



**Revision history**

Rev	Date	Description
01.03	20110307	Replaced missing figure 10. Also small updates and corrections.
01.02	20110225	Removed SPIFI to reflect initial product release. Also small updates and corrections.
01.01	20110125	Many small updates and corrections.
01.00	20101022	First release of the LPC178x/7x User Manual.

**Contact information**

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

## 1.1 Introduction

The LPC178x/177x is an ARM Cortex-M3 based microcontroller for embedded applications requiring a high level of integration and low power dissipation.

The Cortex-M3 is a next generation core that offers better performance than the ARM7 at the same clock rate, and offers other system enhancements such as modernized debug features and a higher level of support block integration. The Cortex-M3 CPU incorporates a 3-stage pipeline and has a Harvard architecture with separate local instruction and data buses, as well as a third bus with slightly lower performance for peripherals. The Cortex-M3 CPU also includes an internal prefetch unit that supports speculative branches. The LPC178x/177x adds a specialized flash memory accelerator to give optimal performance when executing code from flash. The LPC178x/177x is targeted to operate at up to a 100 MHz CPU frequency under worst case commercial conditions.

The peripheral complement of the LPC178x/177x includes up to 512 kB of Flash memory, up to 96 kB of data memory, 4 kB of EEPROM memory, an External Memory Controller for SDRAM and static memory access, an LCD panel controller, an Ethernet MAC, a General Purpose DMA controller, a USB device/host/OTG interface, 5 UARTs, 3 SSP controllers, 3 I<sup>2</sup>C interfaces, an I<sup>2</sup>S serial audio interface, a 2-channel CAN interface, an SD card interface, an 8 channel 12-bit ADC, a 10-bit DAC, a Motor Control PWM, a Quadrature Encoder Interface, 4 general purpose timers, a 6-output general purpose PWM, an ultra-low power RTC with separate battery supply, a windowed watchdog timer, a CRC calculation engine, up to 165 general purpose I/O pins, and more. The pinout of LPC178x/177x is intended to allow pin function compatibility with the LPC24xx and LPC23xx.

## 1.2 Features

Refer to [Section 1.4](#) for details of features for specific part numbers.

- Functional replacement for LPC23xx and 24xx family devices.
- ARM Cortex-M3 processor, running at frequencies of up to 100 MHz. The Cortex-M3 executes the Thumb®-2 instruction set for optimal performance and code size, including hardware division, single cycle multiply, and bit-field manipulation. A Memory Protection Unit (MPU) supporting eight regions is included.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash program memory with In-System Programming (ISP) and In-Application Programming (IAP) capabilities. The combination of an enhanced flash memory accelerator and location of the flash memory on the CPU local code/data bus provides high code performance from flash.
- Up to 96 kB on-chip SRAM includes:
  - Up to 64 kB of SRAM on the CPU with local code/data bus for high-performance CPU access.
  - Up to two 16 kB SRAM blocks with separate access paths for higher throughput. These SRAM blocks may be used for Ethernet, USB, LCD, and DMA memory, as well as for general purpose instruction and data storage.
  - 4kB on-chip EEPROM.
- External Memory Controller provides support for asynchronous static memory devices such as RAM, ROM and Flash up to 64 MB, as well as dynamic memories such as Single Data Rate SDRAM.
- Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with the SSP, I<sup>2</sup>S, UART, SD/MMC, CRC engine, Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, GPIO, and for memory-to-memory transfers.
- Multilayer AHB matrix interconnect provides a separate bus for each AHB master. AHB masters include the CPU, General Purpose DMA controller, Ethernet MAC, LCD controller, and the USB interface. This interconnect provides communication with no arbitration delays unless two masters attempt to access the same slave at the same time.
- Split APB bus allows for higher throughput with fewer stalls between the CPU and DMA. A single level of write buffering allows the CPU to continue without waiting for completion of APB writes if the APB was not already busy.
- LCD controller, supporting both Super-Twisted Nematic (STN) and Thin-Film Transistor (TFT) displays.
  - Dedicated DMA controller.
  - Selectable display resolution (up to 1024 × 768 pixels).
  - Supports up to 24-bit true-color mode.
- Serial interfaces:
  - Ethernet MAC with MII/RMII interface and dedicated DMA controller.
  - USB 2.0 full-speed controller that can be configured for either device, Host, or OTG operation with an on-chip PHY for device and Host functions and a dedicated DMA controller.

- Five UARTs with fractional baud rate generation, internal FIFOs, IrDA, DMA support, and RS-485/EIA-485 support. UART1 also has a full set of modem handshaking signals. UART4 includes a synchronous mode and a Smart Card mode supporting ISO 7816-3. Devices in the 144-pin package provide 4 UARTs.
- Three SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.
- Three enhanced I<sup>2</sup>C-bus interfaces, one with an open-drain output supporting the full I<sup>2</sup>C specification and Fast mode Plus with data rates of 1Mbit/s, two with standard port pins. Enhancements include multiple address recognition and monitor mode.
- Two-channel CAN controller.
- I<sup>2</sup>S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control. The I<sup>2</sup>S interface can be used with the GPDMA. The I<sup>2</sup>S interface supports 3-wire data transmit and receive or 4-wire combined transmit and receive connections, as well as master clock output.
- Other peripherals:
  - SD card interface that also supports MMC cards.
  - General Purpose I/O (GPIO) pins with configurable pull-up/down resistors, open drain mode, and repeater mode. All GPIOs are located on an AHB bus for fast access, and support Cortex-M3 bit-banding. GPIOs can be accessed by the General Purpose DMA Controller. Any pin of ports 0 and 2 can be used to generate an interrupt. There are 165 GPIOs on 208-pin packages, 141 GPIOs on 180-pin packages, and 109 GPIOs on 144-pin packages.
  - 12-bit Analog-to-Digital Converter (ADC) with input multiplexing among eight pins, conversion rates up to 400 kHz, and multiple result registers. The 12-bit ADC can be used with the GPDMA controller.
  - 10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer and DMA support.
  - Four general purpose timers/counters, with a total of eight capture inputs and ten compare outputs. Each timer block has an external count input. Specific timer events can be selected to generate DMA requests.
  - One motor control PWM with support for three-phase motor control.
  - Quadrature encoder interface that can monitor one external quadrature encoder.
  - Two standard PWM/timer blocks with external count input option.
  - Real-Time Clock (RTC) with a separate power domain. The RTC is clocked by a dedicated RTC oscillator. The RTC block includes 20 bytes of battery-powered backup registers, allowing system status to be stored when the rest of the chip is powered off. Battery power can be supplied from a standard 3 V Lithium button cell. The RTC will continue working when the battery voltage drops to as low as 2.1 V. An RTC interrupt can wake up the CPU from any reduced power mode.
  - Windowed Watchdog Timer (WWDT). Windowed operation, dedicated internal oscillator, watchdog warning interrupt, and safety features.
  - CRC Engine block can calculate a CRC on supplied data using 1 of 3 standard polynomials. The CRC engine can be used in conjunction with the DMA controller to generate a CRC without CPU involvement in the data transfer.
  - Cortex-M3 system tick timer, including an external clock input option.

- Standard JTAG test/debug interface as well as Serial Wire Debug and Serial Wire Trace Port options.
- Emulation trace module supports real-time trace.
- Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep Power-down.
- Single 3.3 V power supply (2.4 V to 3.6 V). Temperature range of -40 °C to 85 °C.
- Four external interrupt inputs configurable as edge/level sensitive. All pins on PORT0 and PORT2 can be used as edge sensitive interrupt sources.
- Non-maskable Interrupt (NMI) input.
- Clock output function that can reflect the main oscillator clock, IRC clock, RTC clock, CPU clock, USB clock, or the watchdog timer clock.
- The Wakeup Interrupt Controller (WIC) allows the CPU to automatically wake up from any priority interrupt that can occur while the clocks are stopped in deep sleep, Power-down, and Deep Power-down modes.
- Processor wake-up from Power-down mode via any interrupt able to operate during Power-down mode (includes external interrupts, RTC interrupt, USB activity, Ethernet wake-up interrupt, CAN bus activity, PORT0/2 pin interrupt, and NMI).
- Brownout detect with separate threshold for interrupt and forced reset.
- On-chip Power-On Reset (POR).
- On-chip crystal oscillator with an operating range of 1 MHz to 25 MHz.
- 12 MHz Internal RC oscillator (IRC) trimmed to 1% accuracy that can optionally be used as a system clock.
- An on-chip PLL allows CPU operation up to the maximum CPU rate without the need for a high-frequency crystal. May be run from the main oscillator or the internal RC oscillator.
- A second, dedicated PLL may be used for the USB interface in order to allow added flexibility for the Main PLL settings.
- Versatile pin function selection feature allows many possibilities for using on-chip peripheral functions.
- Boundary scan for simplified board testing.
- Unique device serial number for identification purposes.
- Available as 208-pin LQFP, 208-pin TFBGA, 180-pin TFBGA, and 144-pin LQFP packages.

## 1.3 Applications

- Industrial control and networking
- Medical systems
- Protocol converter
- Communications
- Electronic Metering
- Lighting
- Alarm systems
- White goods
- Motor control

## 1.4 Ordering information

Table 1. Ordering information

Type number	Package		Version
	Name	Description	
<b>LPC1788</b>			
LPC1788FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
<b>LPC1788</b>			
LPC1788FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
LPC1788FET208	TFBGA208	plastic thin fine-pitch ball grid array package; 208 balls; body 15 × 15 × 0.7 mm	SOT950-1
LPC1788FET180	TFBGA180	thin fine-pitch ball grid array package; 180 balls; body 12 × 12 × 0.8 mm	SOT570-2
LPC1788FBD144	LQFP144	plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm	SOT486-1
<b>LPC1787</b>			
LPC1787FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
<b>LPC1786</b>			
LPC1786FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
<b>LPC1785</b>			
LPC1785FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
<b>LPC1778</b>			
LPC1778FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
LPC1778FET208	TFBGA208	plastic thin fine-pitch ball grid array package; 208 balls; body 15 × 15 × 0.7 mm	SOT950-1
LPC1778FET180	TFBGA180	thin fine-pitch ball grid array package; 180 balls; body 12 × 12 × 0.8 mm	SOT570-2
LPC1778FBD144	LQFP144	plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm	SOT486-1
<b>LPC1777</b>			
LPC1777FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
<b>LPC1776</b>			
LPC1776FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
LPC1776FET180	TFBGA180	thin fine-pitch ball grid array package; 180 balls; body 12 × 12 × 0.8 mm	SOT570-2
<b>LPC1774</b>			
LPC1774FBD208	LQFP208	plastic low profile quad flat package; 208 leads; body 28 × 28 × 1.4 mm	SOT459-1
LPC1774FBD144	LQFP144	plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm	SOT486-1



1.4.1 Part options summary

Table 2. Ordering options for LPC178x/177x parts

Type number <sup>[1]</sup>	Flash (kB)	Total SRAM (kB)	EEPROM (kB)	Ethernet	USB	UART	Ex. Mem Bus <sup>[2]</sup>	LCD	QEI	SD
<b>LPC178x</b>										
LPC1788	512	96 <sup>[5]</sup>	4	Y	H/O/D	5	32-bit/ 16-bit <sup>[3]</sup> / 8-bit <sup>[4]</sup>	Y	Y	Y
LPC1787	512	96 <sup>[5]</sup>	4	N	H/O/D	5	32-bit	Y	Y	Y
LPC1786	256	80 <sup>[6]</sup>	4	Y	H/O/D	5	32-bit	Y	Y	Y
LPC1785	256	80 <sup>[6]</sup>	4	N	H/O/D	5	32-bit	Y	N	Y
<b>LPC177x</b>										
LPC1778	512	96 <sup>[5]</sup>	4	Y	H/O/D	5	32-bit/ 16-bit <sup>[3]</sup> / 8-bit <sup>[4]</sup>	N	Y	Y
LPC1777	512	96 <sup>[5]</sup>	4	N	H/O/D	5	32-bit	N	Y	Y
LPC1776	256	80 <sup>[6]</sup>	4	Y	H/O/D	5	32-bit/ 16-bit <sup>[3]</sup>	N	Y	Y
LPC1774	128	40 <sup>[7]</sup>	2	N	D	5/4 <sup>[4]</sup>	32-bit/ 8-bit <sup>[4]</sup>	N	N	N

- [1] All types include 2 CAN channels, 3 SSP interfaces, 3 I<sup>2</sup>C interfaces, I2S, DAC, and an 8-channel 12-bit ADC.
- [2] Maximum data bus width for available packages. Smaller widths may be used. See [Section 10.1](#) for details of external bus usage on different packages.
- [3] On 180-pin packages, the external bus is limited to 16 bits.
- [4] On 144-pin packages, UART4 and the DAC are not available, and the external bus is limited to 8 bits.
- [5] 64 kB CPU SRAM plus 32 kB peripheral SRAM.
- [6] 64 kB CPU SRAM plus 16 kB peripheral SRAM.
- [7] 32 kB CPU SRAM plus 8 kB peripheral SRAM.
- [8] 16 kB CPU SRAM plus 8 kB peripheral SRAM.

5. Simplified block diagram

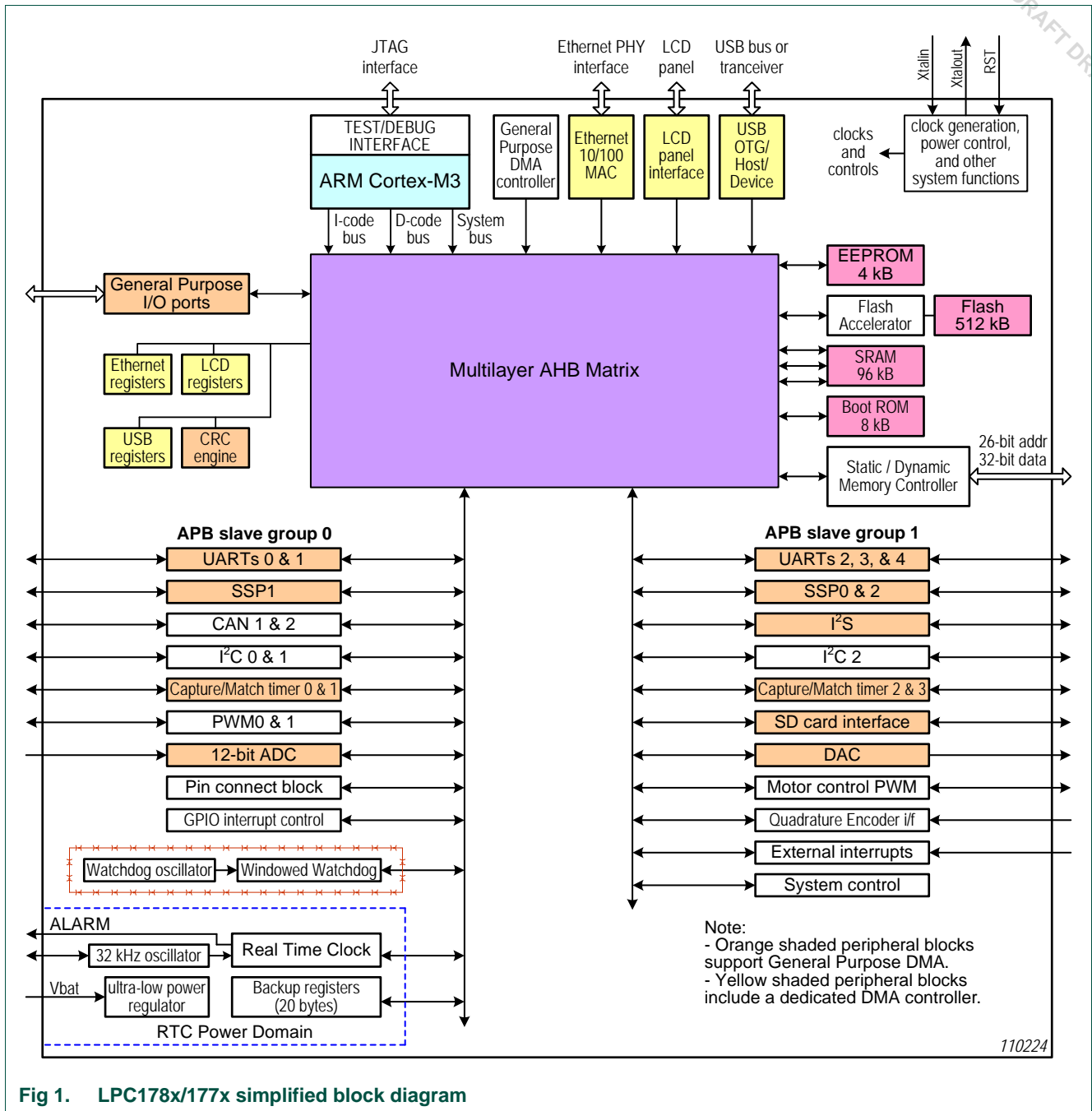


Fig 1. LPC178x/177x simplified block diagram

## 1.6 Architectural overview

The ARM Cortex-M3 includes three AHB-Lite buses, one system bus and the I-code and D-code buses which are faster and are used similarly to Tightly Coupled Memory interfaces: one bus dedicated for instruction fetch (I-code) and one bus for data access (D-code). The use of two core buses allows for simultaneous operations if concurrent operations target different devices.

The LPC178x/177x uses a multi-layer AHB matrix to connect the Cortex-M3 buses and other bus masters to peripherals in a flexible manner that optimizes performance by allowing peripherals on different slaves ports of the matrix to be accessed simultaneously by different bus masters. Details of the multilayer matrix connections are shown in [Figure 2](#).

APB peripherals are connected to the CPU via two APB buses using separate slave ports from the multilayer AHB matrix. This allows for better performance by reducing collisions between the CPU and the DMA controller. The APB bus bridges are configured to buffer writes so that the CPU or DMA controller can write to APB devices without always waiting for APB write completion.

## 1.7 ARM Cortex-M3 processor

The ARM Cortex-M3 is a general purpose 32-bit microprocessor, which offers high performance and very low power consumption. The Cortex-M3 offers many new features, including a Thumb-2 instruction set, low interrupt latency, hardware divide, interruptible/continuable multiple load and store instructions, automatic state save and restore for interrupts, tightly integrated interrupt controller with Wakeup Interrupt Controller, and multiple core buses capable of simultaneous accesses.

Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

The ARM Cortex-M3 processor is described in detail in the Cortex-M3 User Guide that is appended to this manual.

### 1.7.1 Cortex-M3 Configuration Options

The LPC178x/177x uses the r2p0 version of the Cortex-M3 CPU, which includes a number of configurable options, as noted below.

#### System options:

- The Nested Vectored Interrupt Controller (NVIC) is included. The NVIC includes the SYSTICK timer.
- The Wakeup Interrupt Controller (WIC) is included. The WIC allows more powerful options for waking up the CPU from reduced power modes.
- A Memory Protection Unit (MPU) is included.
- A ROM Table is included. The ROM Table provides addresses of debug components to external debug systems.

#### Debug related options:

- A JTAG debug interface is included.
- Serial Wire Debug is included. Serial Wire Debug allows debug operations using only 2 wires, simple trace functions can be added with a third wire.
- The Embedded Trace Macrocell (ETM) is included. The ETM provides instruction trace capabilities.
- The Data Watchpoint and Trace (DWT) unit is included. The DWT allows data address or data value matches to be trace information or trigger other events. The DWT includes 4 comparators and counters for certain internal events.
- An Instrumentation Trace Macrocell (ITM) is included. Software can write to the ITM in order to send messages to the trace port.
- The Trace Port Interface Unit (TPIU) is included. The TPIU encodes and provides trace information to the outside world. This can be on the Serial Wire Viewer pin or the 4-bit parallel trace port.
- A Flash Patch and Breakpoint (FPB) is included. The FPB can generate hardware breakpoints and remap specific addresses in code space to SRAM as a temporary method of altering non-volatile code. The FPB includes 2 literal comparators and 6 instruction comparators.

## 1.8 On-chip flash memory system

The LPC178x/177x contains up to 512 kB of on-chip flash memory. A flash memory accelerator maximizes performance for CPU accesses. This memory may be used for both code and data storage. Programming of the flash memory may be accomplished in several ways. It may be programmed In System via the serial port. The application program may also erase and/or program the flash while the application is running, allowing a great degree of flexibility for data storage field firmware upgrades, etc.

## 1.9 On-chip Static RAM

The LPC178x/177x contains up to 96 kB of on-chip static RAM memory. Up to 64 kB of SRAM, accessible by the CPU and the General Purpose DMA controller, is on a higher-speed bus. Up to 32 kB SRAM is provided in up to two additional 16 kB SRAM blocks for use primarily for peripheral data. When both SRAMs are present, they are situated on separate slave ports on the AHB multilayer matrix.

This architecture allows the possibility for CPU and DMA accesses to be separated in such a way that there are few or no delays for the bus masters. It also allows separation of data for different peripherals functions, in order to improve system performance. For example, LCD DMA can be occurring in one SRAM while Ethernet DMA is occurring in another, all while the CPU is using the system SRAM for data and/or instruction access.

## 1.10 On-chip EEPROM

The LPC178x/177x contains up to 4 kB of on-chip EEPROM memory. The EEPROM is accessible only by the CPU.

1.11 Detailed block diagram

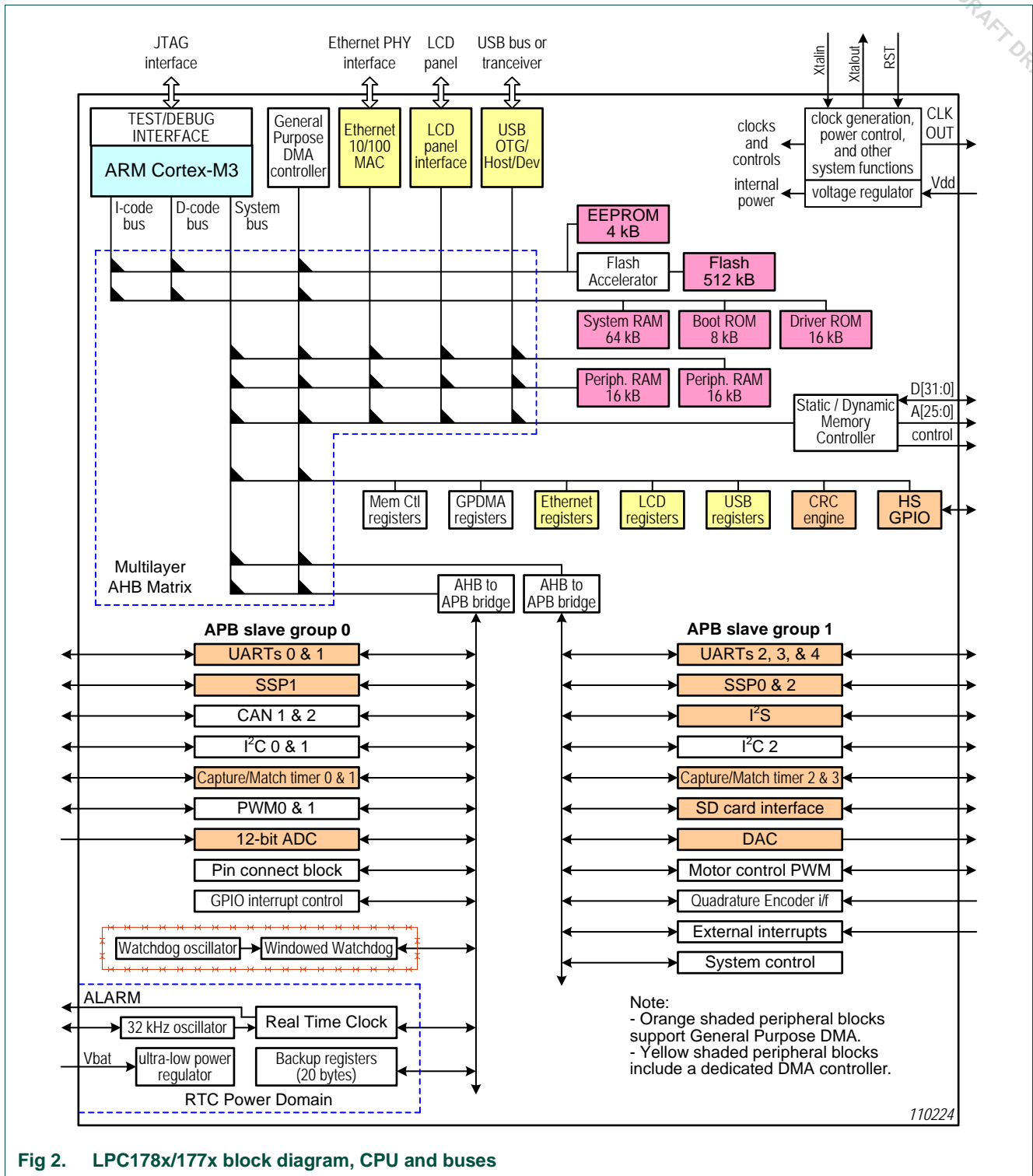


Fig 2. LPC178x/177x block diagram, CPU and buses

### 2.1 Memory map and peripheral addressing

The ARM Cortex-M3 processor has a single 4 GB address space. The following table shows how this space is used on the LPC178x/177x.

**Table 3. LPC178x/177x memory usage and details**

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 FFFF	For devices with 64 kB of local SRAM.
		0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.	
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2000 0000 - 0x2000 1FFF	Peripheral RAM - bank 0 (first 8 kB)
		0x2002 0000 - 0x2000 3FFF	Peripheral RAM - bank 0 (second 8 kB)
		0x2000 4000 - 0x2000 7FFF	Peripheral RAM - bank 1 (16 kB)
0x4000 0000 to 0x7FFF FFFF	AHB peripherals	0x2008 0000 - 0x200B FFFF	See <a href="#">Section 2.3.1</a> for details
		0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks of 16 kB each.
0x4000 0000 to 0x7FFF FFFF	APB Peripherals	0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks of 16 kB each.
		0x8000 0000 to 0xDFFF FFFF	Off-chip Memory via the External Memory Controller
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.

[1] Can be up to 256 MB, upper address 0x8FFF FFFF, if the address shift mode is enabled. See SCS register bit 0 ([Section 3.8.1](#)).

[2] Can be up to 128 MB, upper address 0x97FF FFFF, if the address shift mode is enabled. See SCS register bit 0 ([Section 3.8.1](#)).

## 2.2 Memory maps

The LPC178x/177x incorporates several distinct memory regions, shown in the following figures. [Figure 3](#) shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address remapping, which is described later in this section.

[Figure 3](#) and [Table 5](#) show different views of the peripheral address space. The AHB peripheral area is 2 megabyte in size, and is divided to allow for up to 128 peripherals. The APB peripheral area is 1 megabyte in size and is divided to allow for up to 64 peripherals. Each peripheral of either type is allocated 16 kilobytes of space. This allows simplifying the address decoding for each peripheral.

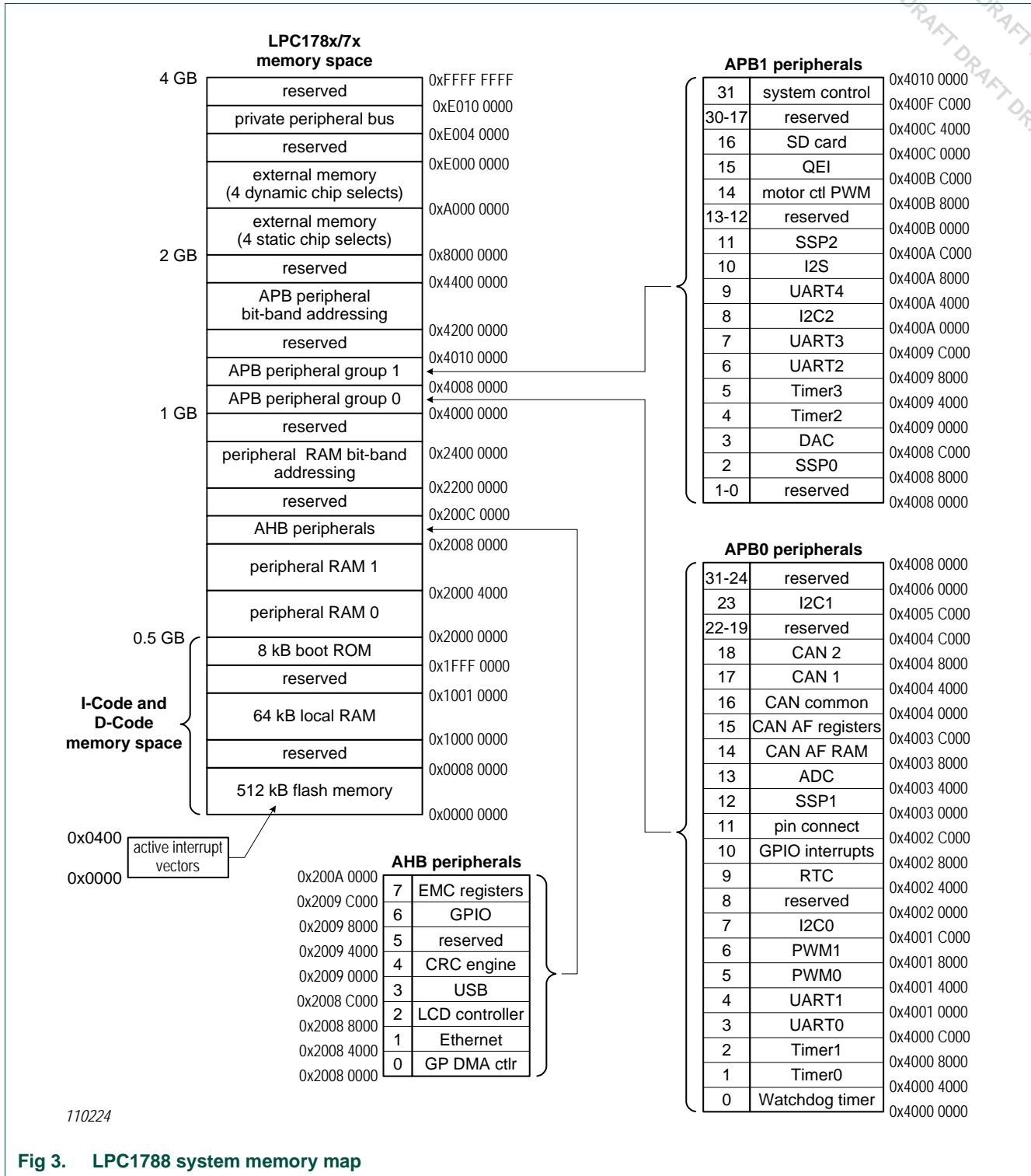


Fig 3. LPC1788 system memory map



## 2.3 On-chip peripherals

All peripheral register addresses are word aligned (to 32-bit boundaries) regardless of their size. This eliminates the need for byte lane mapping hardware that would be required to allow byte (8-bit) or half-word (16-bit) accesses to occur at smaller boundaries. An implication of this is that word and half-word registers must be accessed all at once. For example, it is not possible to read or write the upper byte of a word register separately.

### 2.3.1 AHB peripherals

The following table shows the addresses of peripheral functions that reside directly on the AHB bus matrix. Complete register descriptions may be found in the relevant chapters.

**Table 4. AHB peripherals and base addresses**

AHB peripheral	Address range	Peripheral name
0	0x2008 0000 to 0x2008 3FFF	General Purpose DMA controller
1	0x2008 4000 to 0x2008 7FFF	Ethernet MAC
2	0x2008 8000 to 0x2008 BFFF	LCD controller
3	0x2008 C000 to 0x2008 FFFF	USB interface
4	0x2009 0000 to 0x2009 3FFF	CRC engine
5	0x2009 4000 to 0x2009 7FFF	reserved
6	0x2009 8000 to 0x2009 BFFF	GPIO
7	0x2009 C000 to 0x2009 FFFF	External Memory Controller
8 to 15	0x200A 0000 to 0x200B FFFF	reserved

### 2.3.2 APB peripheral addresses

The following table shows the address maps of the 2 APB buses. APB peripherals do not use all of the 16 kB space allocated to them. Typically each device's registers are "aliased" or repeated at multiple locations within each 16 kB range.

**Table 5. APB0 peripherals and base addresses**

APB0 peripheral	Base address	Peripheral name
0	0x4000 0000	Watchdog Timer
1	0x4000 4000	Timer 0
2	0x4000 8000	Timer 1
3	0x4000 C000	UART0
4	0x4001 0000	UART1
5	0x4001 4000	PWM0
6	0x4001 8000	PWM1
7	0x4001 C000	I <sup>2</sup> C0
8	0x4002 0000	reserved
9	0x4002 4000	RTC
10	0x4002 8000	GPIO interrupts
11	0x4002 C000	Pin Connect Block
12	0x4003 0000	SSP1
13	0x4003 4000	ADC

**Table 5. APB0 peripherals and base addresses**

APB0 peripheral	Base address	Peripheral name
14	0x4003 8000	CAN Acceptance Filter RAM
15	0x4003 C000	CAN Acceptance Filter Registers
16	0x4004 0000	CAN Common Registers
17	0x4004 4000	CAN Controller 1
18	0x4004 8000	CAN Controller 2
19 to 22	0x4004 C000 to 0x4005 8000	reserved
23	0x4005 C000	I <sup>2</sup> C1
24 to 31	0x4006 0000 to 0x4007 C000	reserved

**Table 6. APB1 peripherals and base addresses**

APB1 peripheral	Base address	Peripheral name
0 to 1	0x4008 0000 to 0x4008 4000	reserved
2	0x4008 8000	SSP0
3	0x4008 C000	DAC
4	0x4009 0000	Timer 2
5	0x4009 4000	Timer 3
6	0x4009 8000	UART2
7	0x4009 C000	UART3
8	0x400A 0000	I <sup>2</sup> C2
9	0x400A 4000	UART4
10	0x400A 8000	I <sup>2</sup> S
11	0x400A C000	SSP2
12 to 13	0x400B 0000 to 0x400B 4000	reserved
14	0x400B 8000	Motor control PWM
15	0x400B C000	Quadrature Encoder Interface
16	0x400C 0000	SD card interface
17 to 30	0x400D 0000 to 0x400F 8000	reserved
31	0x400F C000	System control

## 2.4 Memory re-mapping

The Cortex-M3 incorporates a mechanism that allows remapping the interrupt vector table to alternate locations in the memory map. This is controlled via the Vector Table Offset Register contained in the Cortex-M3. Refer to the NVIC description [Section 6.4](#) and [Section 38.4.3.5](#) of the Cortex-M3 User Guide appended to this manual for details of the Vector Table Offset feature.

In the case of flashless operation, user program starting at address 0x8000 0000 must set up an interrupt vector table. For more on flashless operation, see [Section 7.2 “Boot control”](#).

### Boot ROM re-mapping

Following a hardware reset, the Boot ROM is temporarily mapped to address 0. This is normally transparent to the user. However, if execution is halted immediately after reset by a debugger, it should correct the mapping for the user. See [Section 37.8](#).

## 2.5 AHB arbitration

The Multilayer AHB Matrix arbitrates between several masters, only if they attempt to access the same matrix slave port at the same time. By default, the Cortex-M3 D-code bus has the highest priority, followed by the I-Code bus. All other masters share a lower priority.

The default priority can be altered by the user if care is taken. This may be particularly useful if the LCD interface is used and it has difficulty getting sufficient data.

### 2.5.1 Matrix Arbitration register (Matrix\_Arb - 0x400F C188)

The Matrix Arbitration register provides the ability to change the default AHB Matrix arbitration priorities.

**Table 7. Matrix Arbitration register (Matrix\_Arb - 0x400F C188) bit description**

Bit	Symbol	Description	Reset value
1:0	PRI_ICODE	I-Code bus priority. Should be lower than PRI_DCODE for proper operation.	0x1
3:2	PRI_DCODE	D-Code bus priority.	0x3
5:4	PRI_SYS	System bus priority.	0
7:6	PRI_GPDMA	General Purpose DMA controller priority.	0
9:8	PRI_ETH	Ethernet DMA priority.	0
11:10	PRI_LCD	LCD DMA priority.	0
13:12	PRI_USB	USB DMA priority.	0
15:14	-	Reserved. Read value is undefined, only zero should be written.	NA
16	ROM_LAT	ROM latency select. Should always be 0.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

The values used for the various priorities are 3 = highest, 0 = lowest.

An example of a way to give priority to the LCD DMA is to use the value 0x0000 0C09. This gives the LCD highest priority, D-code second priority, I-Code third priority, and all others lowest priority.

Where in the memory space code and various types of data are located can be managed to help minimize the need for arbitration and possible starvation of any of the bus masters, as well as a need for changing the default priorities. For instance, LCD refresh from off-chip memory connected to the EMC, while also executing off-chip code via the EMC can cause a great deal of arbitration.

### 3.1 Introduction

The system control block includes several system features and control registers for a number of functions that are not related to specific peripheral devices. These include:

- Chip Reset (see [Section 3.4](#))
- Peripheral Reset control (see [Section 3.5](#))
- Brown-Out Detection (see [Section 3.6](#))
- External Interrupt Inputs (see [Section 3.7](#))
- Miscellaneous System Controls and Status (see [Section 3.8](#))

Each type of function has its own register(s) if any are required and unneeded bits are defined as reserved in order to allow future expansion.

### 3.2 Pin description

[Table 8](#) shows pins that are associated with System Control block functions.

**Table 8. Pin summary**

Pin name	Pin direction	Pin description
EINT0	Input	<b>External Interrupt Input 0</b> - An active low/high level or falling/rising edge general purpose interrupt input. This pin may be used to wake up the processor from Sleep, Deep-sleep, or Power-down modes.
EINT1	Input	<b>External Interrupt Input 1</b> - See the EINT0 description above.
EINT2	Input	<b>External Interrupt Input 2</b> - See the EINT0 description above.
EINT3	Input	<b>External Interrupt Input 3</b> - See the EINT0 description above.
RESET	Input	<b>External Reset input</b> - A LOW on this pin resets the chip, causing I/O ports and peripherals to take on their default states, and the processor to begin execution at address 0x0000 0000.

### 3.3 Register description

All registers, regardless of size, are on word address boundaries. Details of the registers appear in the description of each function.

**Table 9. Summary of system control registers**

Name	Description	Access	Reset value	Address	Table
<b>Reset</b>					
RSID	Reset Source Identification Register	R/W	see <a href="#">Table 10</a>	0x400F C180	<a href="#">10</a>
<b>Peripheral Reset Control</b>					
RSTCON0	Individual peripheral reset control bits	R/W	0	0x400F C1CC	<a href="#">11</a>
RSTCON1	Individual peripheral reset control bits	R/W	0	0x400F C1D0	<a href="#">12</a>
<b>External Interrupts</b>					
EXTINT	External Interrupt Flag Register	R/W	0	0x400F C140	<a href="#">14</a>
EXTMODE	External Interrupt Mode register	R/W	0	0x400F C148	<a href="#">15</a>
EXTPOLAR	External Interrupt Polarity Register	R/W	0	0x400F C14C	<a href="#">16</a>
<b>Syscon Miscellaneous Registers</b>					
SCS	System Control and Status	R/W	0	0x400F C1A0	<a href="#">17</a>

### 3.4 Chip reset

Reset has 6 sources on the LPC178x/177x: the  $\overline{\text{RESET}}$  pin, Watchdog Reset, Power On Reset (POR), Brown Out Detect (BOD), system reset, and lockup.

The  $\overline{\text{RESET}}$  pin is a Schmitt trigger input pin. Assertion of chip Reset by any source, once the operating voltage attains a usable level, starts the wake-up timer (see description in [Section 4.8 “Wake-up timer”](#) in this chapter), causing reset to remain asserted until the external Reset is de-asserted, the oscillator is running, a fixed number of clocks have passed, and the flash controller has completed its initialization. The reset logic is shown in the following block diagram (see [Figure 4](#)).

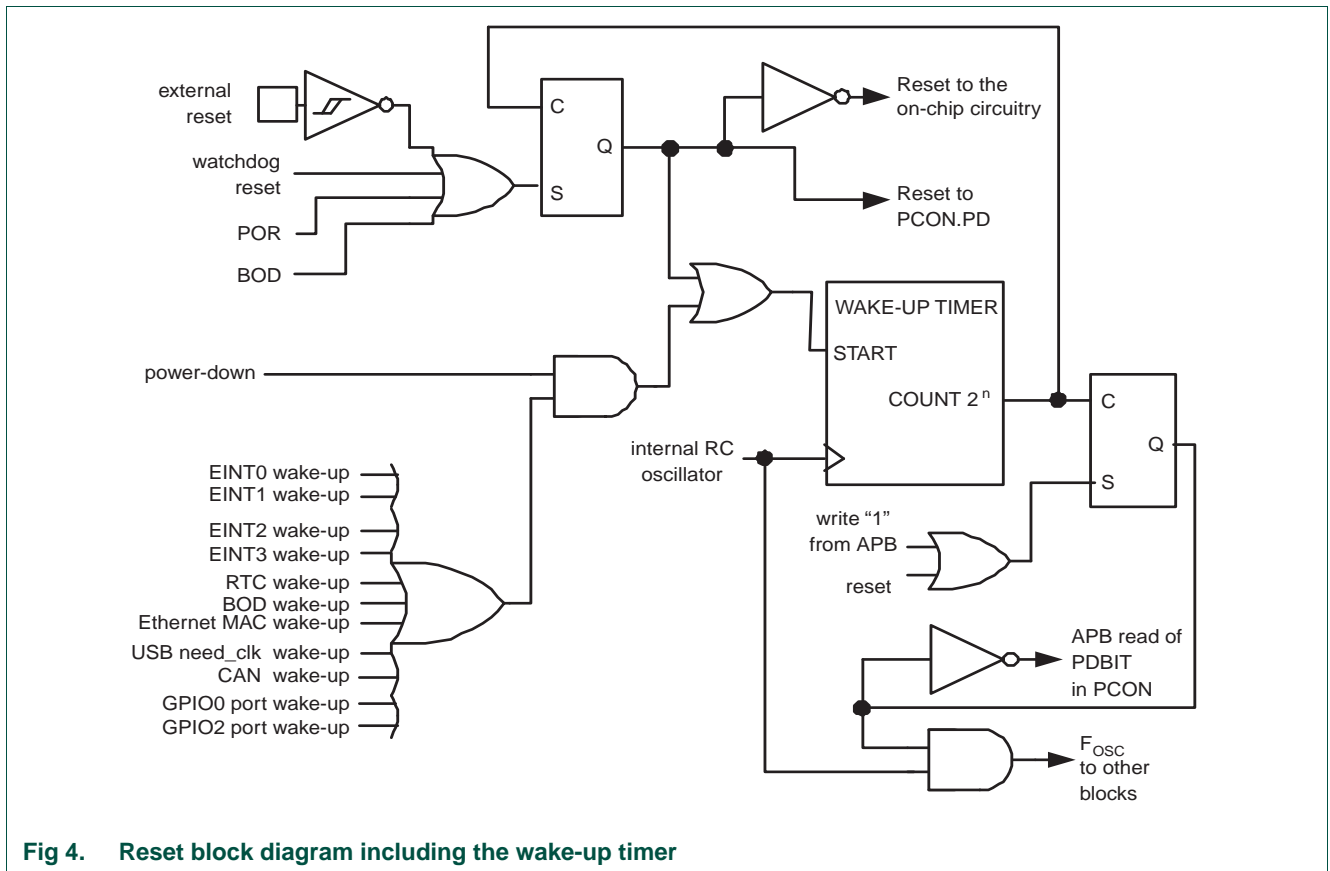


Fig 4. Reset block diagram including the wake-up timer

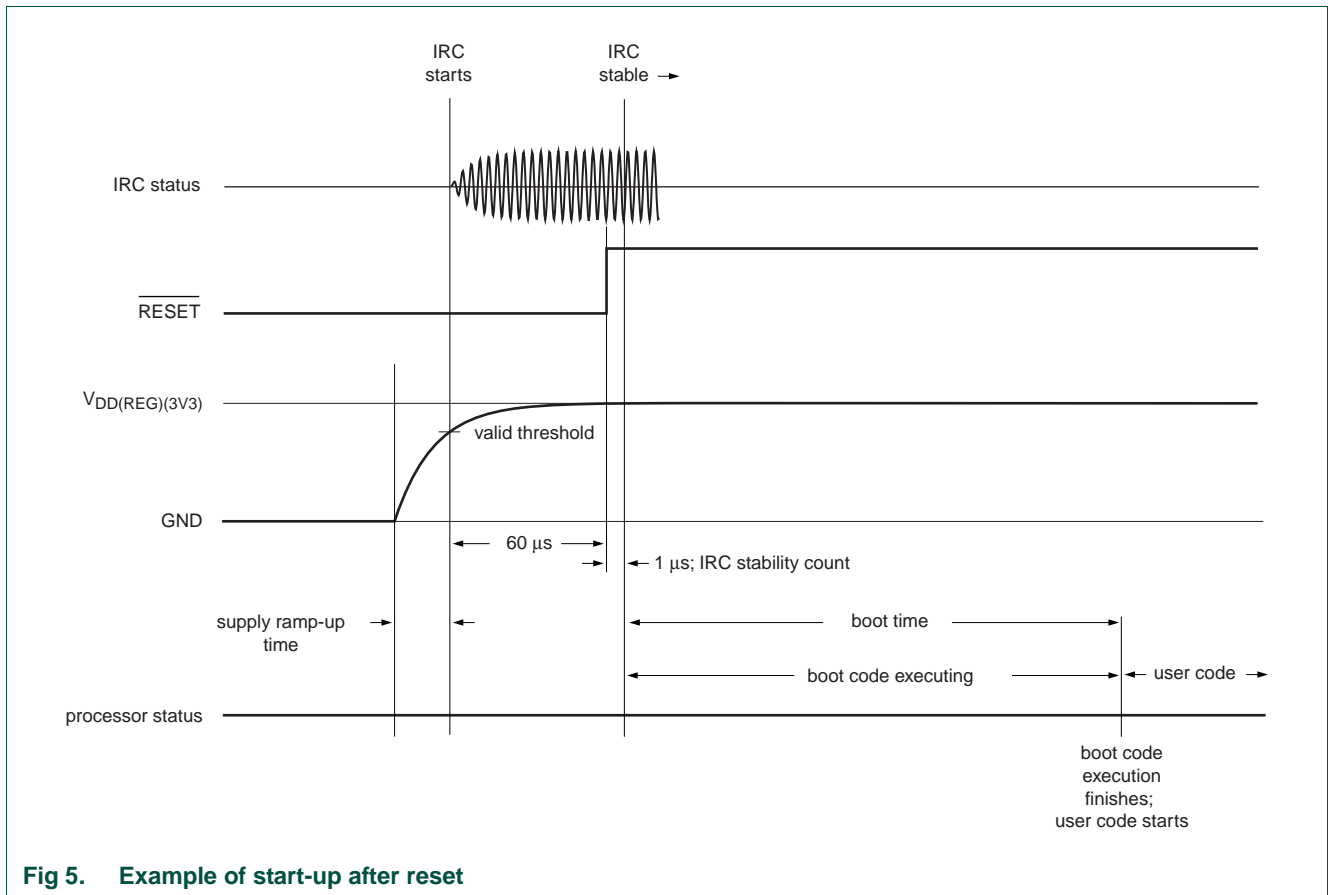
On the assertion of a reset source external to the Cortex-M3 CPU (POR, BOD reset, External reset, and Watchdog reset), the IRC starts up. After the IRC-start-up time (maximum of 60  $\mu\text{s}$  on power-up) and after the IRC provides a stable clock output, the reset signal is latched and synchronized on the IRC clock. Then the following two sequences start simultaneously:

1. The 2-bit IRC wake-up timer starts counting when the synchronized reset is de-asserted. The boot code in the ROM starts when the 2-bit IRC wake-up timer times out. The boot code performs the boot tasks and may jump to the flash. If the flash is not ready to access, the Flash Accelerator will insert wait cycles until the flash is ready.

2. The flash wake-up timer (9-bit) starts counting when the synchronized reset is de-asserted. The flash wakeup-timer generates the 100  $\mu\text{s}$  flash start-up time. Once it times out, the flash initialization sequence is started, which takes about 250 cycles. When it's done, the Flash Accelerator will be granted access to the flash.

When the internal Reset is removed, the processor begins executing at address 0, which is initially the Reset vector mapped from the Boot Block. At that point, all of the processor and peripheral registers have been initialized to predetermined values.

[Figure 5](#) shows an example of the relationship between the  $\overline{\text{RESET}}$ , the IRC, and the processor status when the LPC178x/177x starts up after reset. See [Section 4.3.2 "Main oscillator"](#) for start-up of the main oscillator if selected by the user code.



**Fig 5. Example of start-up after reset**

### 3.4.1 Reset Source Identification Register (RSID - 0x400F C180)

This register contains one bit for each source of Reset. Writing a 1 to any of these bits clears the corresponding read-side bit to 0. The interactions among the four sources are described below.

**Table 10. Reset Source Identification register (RSID - address 0x400F C180) bit description**

Bit	Symbol	Description	Reset value
0	POR	Assertion of the POR signal sets this bit, and clears all of the other bits in this register. But if another Reset signal (e.g., External Reset) remains asserted after the POR signal is negated, then its bit is set. This bit is not affected by any of the other sources of Reset.	See description
1	EXTR	Assertion of the external $\overline{\text{RESET}}$ signal sets this bit. This bit is cleared only by software or POR.	See description
2	WDTR	This bit is set when the Watchdog Timer times out and the WDTRESET bit in the Watchdog Mode Register is 1. This bit is cleared only by software or POR.	See description
3	BODR	<p>This bit is set when the <math>V_{\text{DD(REG)(3V3)}}</math> voltage reaches a level below the BOD reset trip level (typically 1.85 V under nominal room temperature conditions).</p> <p>If the <math>V_{\text{DD(REG)(3V3)}}</math> voltage dips from the normal operating range to below the BOD reset trip level and recovers, the BODR bit will be set to 1.</p> <p>If the <math>V_{\text{DD(REG)(3V3)}}</math> voltage dips from the normal operating range to below the BOD reset trip level and continues to decline to the level at which POR is asserted (nominally 1 V), the BODR bit is cleared.</p> <p>If the <math>V_{\text{DD(REG)(3V3)}}</math> voltage rises continuously from below 1 V to a level above the BOD reset trip level, the BODR will be set to 1.</p> <p>This bit is cleared only by software or POR.</p> <p><b>Note:</b> Only in the case where a reset occurs and the POR = 0, the BODR bit indicates if the <math>V_{\text{DD(REG)(3V3)}}</math> voltage was below the BOD reset trip level or not.</p>	See description
4	SYSRESET	This bit is set if the processor has been reset due to a system reset request, as described in <a href="#">Section 38.4.3.6 “Application Interrupt and Reset Control Register”</a> . Setting the SYSRESETREQ bit in the Cortex-M3 AIRCR register causes a chip reset in the LPC178x/177x. This bit is cleared only by software or POR.	See description
5	LOCKUP	This bit is set if the processor has been reset due to a “lockup”, as described in <a href="#">Section 38.3.4.4 “Lockup”</a> . The lockup state causes a chip reset in the LPC178x/177x. This bit is cleared only by software or POR.	See description
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA



## 3.5 Peripheral reset control

Most peripheral functions on the LPC17M6x can have a hardware reset initiated by software by setting appropriate bits in the RSTCON0 and RSTCON1 registers. Software must clear the RSTCON register after this in order to allow the peripheral to function. A peripheral remains in a hardware reset state as long as the corresponding bit in RSTCON = 1.

### 3.5.1 Reset control register 0 (RSTCON0 - 0x400F C1CC)

Many peripherals may be given a hardware reset using the RSTCON0 register. These are described in [Table 11](#) below. Some additional peripherals may be reset using the RSTCON1 register following.

**Table 11. Reset control register 0 (RSTCON0 - - 0x400F C1CC) bit description**

Bit	Symbol	Description	Reset value
0	RSTLCD	LCD controller reset control bit.	0
1	RSTTIM0	Timer/Counter 0 reset control bit.	0
2	RSTTIM1	Timer/Counter 1 reset control bit.	0
3	RSTUART0	UART0 reset control bit.	0
4	RSTUART1	UART1 reset control bit.	0
5	RSTPWM0	PWM0 reset control bit.	0
6	RSTPWM1	PWM1 reset control bit.	0
7	RSTI2C0	The I <sup>2</sup> C0 interface reset control bit.	0
8	RSTUART4	UART4 reset control bit.	0
9	-	Reserved. Read value is undefined, only zero should be written.	NA
10	RSTSSP1	The SSP 1 interface reset control bit.	0
11	RSTEMC	External Memory Controller reset control bit.	0
12	RSTADC	A/D converter (ADC) reset control bit.	0
13	RSTCAN1	CAN Controller 1 reset control bit. Note: The CAN acceptance filter may be reset by a separate bit in the RSTCON1 register.	0
14	RSTCAN2	CAN Controller 2 reset control bit. Note: The CAN acceptance filter may be reset by a separate bit in the RSTCON1 register.	0
15	RSTGPIO	Reset control bit for GPIO, and GPIO interrupts. Note: IOCON may be reset by a separate bit in the RSTCON1 register.	0
16	-	Reserved. Read value is undefined, only zero should be written.	NA
17	RSTMCPWM	Motor Control PWM reset control bit.	0
18	RSTQEI	Quadrature Encoder Interface reset control bit.	0
19	RSTI2C1	The I <sup>2</sup> C1 interface reset control bit.	0
20	RSTSSP2	The SSP2 interface reset control bit.	0
21	RSTSSP0	The SSP0 interface reset control bit.	0
22	RSTTIM2	Timer 2 reset control bit.	0
23	RSTTIM3	Timer 3 reset control bit.	0
24	RSTUART2	UART 2 reset control bit.	0
25	RSTUART3	UART 3 reset control bit.	0
26	RSTI2C2	I <sup>2</sup> C interface 2 reset control bit.	0

**Table 11. Reset control register 0 (RSTCON0 - 0x400F C1CC) bit description**

Bit	Symbol	Description	Reset value
27	RSTI2S	I <sup>2</sup> S interface reset control bit.	0
28	RSTSDC	SD Card interface reset control bit.	0
29	RSTGPDMA	GPDMA function reset control bit.	0
30	RSTENET	Ethernet block reset control bit.	0
31	RSTUSB	USB interface reset control bit.	0

### 3.5.2 Reset control register 1 (RSTCON1 - 0x400F C1D0)

Some additional peripherals may be given a hardware reset using the RSTCON1 register, as shown in [Table 12](#) below.

**Table 12. Reset control register 1 (RSTCON1 - 0x400F C1D0) bit description**

Bit	Symbol	Description	Reset value
0	RSTIOCON	Reset control bit for the IOCON registers.	0
1	RSTDAC	D/A converter (DAC) reset control bit.	0
2	RSTCANACC	CAN acceptance filter reset control bit.	0
31:3	-	Reserved. Read value is undefined, only zero should be written.	NA

## 3.6 Brown-out detection

The LPC178x/177x includes a Brown-Out Detector (BOD) that provides 2-stage monitoring of the voltage on the  $V_{DD(REG)(3V3)}$  pins. If this voltage falls below the BOD interrupt trip level (typically 2.2 V under nominal room temperature conditions), the BOD asserts an interrupt signal to the NVIC. This signal can be enabled for interrupt in the Interrupt Enable Register in the NVIC in order to cause a CPU interrupt; if not, software can monitor the signal by reading the Raw Interrupt Status Register.

The second stage of low-voltage detection asserts Reset to inactivate the LPC178x/177x when the voltage on the  $V_{DD(REG)(3V3)}$  pins falls below the BOD reset trip level (typically 1.85 V under nominal room temperature conditions). This Reset prevents alteration of the flash as operation of the various elements of the chip would otherwise become unreliable due to low voltage. The BOD circuit maintains this reset down below 1 V, at which point the Power-On Reset circuitry maintains the overall Reset.

Both the BOD reset interrupt level and the BOD reset trip level thresholds include some hysteresis. In normal operation, this hysteresis allows the BOD reset interrupt level detection to reliably interrupt, or a regularly-executed event loop to sense the condition.

But when Brown-Out Detection is enabled to bring the LPC178x/177x out of Power-down mode (which is itself not a guaranteed operation -- see [Section 4.7.7 "Power Mode Control register \(PCON - 0x400F C0C0\)"](#)), the supply voltage may recover from a transient before the wake-up timer has completed its delay. In this case, the net result of the transient BOD is that the part wakes up and continues operation after the instructions that set Power-down mode, without any interrupt occurring and with the BOD bit in the RSID being 0. Since all other wake-up conditions have latching flags (see [Section 3.7.2 "External Interrupt flag register \(EXTINT - 0x400F C140\)"](#) and [Section 29.6.2](#)), a wake-up of this type, without any apparent cause, can be assumed to be a Brown-Out that has gone away.

### 3.7 External interrupt inputs

The LPC178x/177x includes four External Interrupt Inputs as selectable pin functions. The logic of an individual external interrupt is represented in [Figure 6](#). In addition, external interrupts have the ability to wake up the CPU from Power-down mode. Refer to [Section 4.7.8 “Wake-up from Reduced Power Modes”](#) for details.

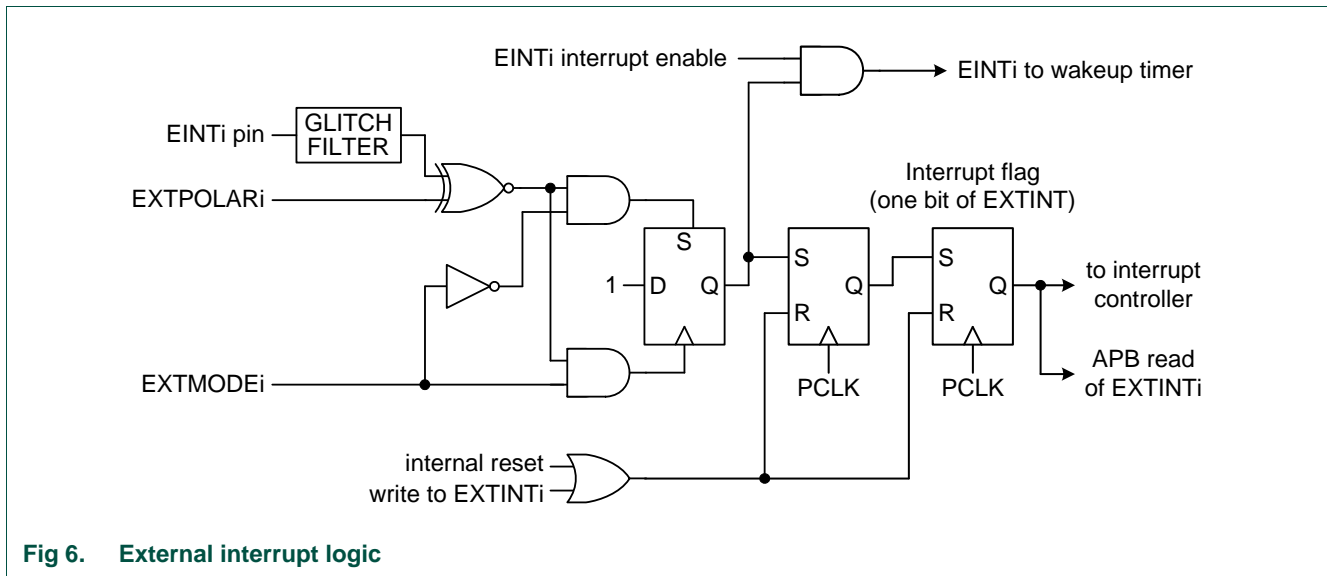


Fig 6. External interrupt logic

### 3.7.1 Register description

The external interrupt function has four registers associated with it. The EXTINT register contains the interrupt flags. The EXTMODE and EXTPOLAR registers specify the level and edge sensitivity parameters.

**Table 13. External Interrupt registers**

Name	Description	Access	Reset value <sup>[1]</sup>	Address
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3. See <a href="#">Table 14</a> .	R/W	0x00	0x400F C140
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. See <a href="#">Table 15</a> .	R/W	0x00	0x400F C148
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt. See <a href="#">Table 16</a> .	R/W	0x00	0x400F C14C

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 3.7.2 External Interrupt flag register (EXTINT - 0x400F C140)

When a pin is selected for its external interrupt function, the level or edge on that pin (selected by its bits in the EXTPOLAR and EXTMODE registers) will set its interrupt flag in this register. This asserts the corresponding interrupt request to the NVIC, which will cause an interrupt if interrupts from the pin are enabled.

Writing ones to bits EINT0 through EINT3 in EXTINT register clears the corresponding bits. In level-sensitive mode the interrupt is cleared only when the pin is in its inactive state.

Once a bit from EINT0 to EINT3 is set and an appropriate code starts to execute (handling wake-up and/or external interrupt), this bit in EXTINT register must be cleared. Otherwise event that was just triggered by activity on the EINT pin will not be recognized in future.

**Important: whenever a change of external interrupt operating mode (i.e. active level/edge) is performed (including the initialization of an external interrupt), the corresponding bit in the EXTINT register must be cleared! For details see [Section 3.7.3 “External Interrupt Mode register \(EXTMODE - 0x400F C148\)”](#) and [Section 3.7.4 “External Interrupt Polarity register \(EXTPOLAR - 0x400F C14C\)”](#).**

For example, if a system wakes up from Power-down using low level on external interrupt 0 pin, its post wake-up code must reset EINT0 bit in order to allow future entry into the Power-down mode. If EINT0 bit is left set to 1, subsequent attempt(s) to invoke Power-down mode will fail. The same goes for external interrupt handling.

More details on Power-down mode will be discussed in the following chapters.

**Table 14. External Interrupt Flag register (EXTINT - address 0x400F C140) bit description**

Bit	Symbol	Description	Reset value
0	EINT0	In level-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the selected edge occurs on the pin.  This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. <sup>[1]</sup>	0
1	EINT1	In level-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the selected edge occurs on the pin.  This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. <sup>[1]</sup>	0
2	EINT2	In level-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the selected edge occurs on the pin.  This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. <sup>[1]</sup>	0
3	EINT3	In level-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT3 function is selected for its pin, and the selected edge occurs on the pin.  This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state. <sup>[1]</sup>	0
31:4	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Example: e.g. if the EINTx is selected to be low level sensitive and low level is present on corresponding pin, this bit can not be cleared; this bit can be cleared only when signal on the pin becomes high.

### 3.7.3 External Interrupt Mode register (EXTMODE - 0x400F C148)

The bits in this register select whether each EINT pin is level- or edge-sensitive. Only pins that are selected for the EINT function (see [Section 8.3](#)) and enabled in the appropriate NVIC register) can cause interrupts from the External Interrupt function (though of course pins selected for other functions may cause interrupts from those functions).

**Note: Software should only change a bit in this register when its interrupt is disabled in the NVIC (state readable in the ISERN/ICERN registers), and should write the corresponding 1 to EXTINT before enabling (initializing) or re-enabling the interrupt. An extraneous interrupt(s) could be set by changing the mode and not having the EXTINT cleared.**

**Table 15. External Interrupt Mode register (EXTMODE - address 0x400F C148) bit description**

Bit	Symbol	Value	Description	Reset value
0	EXTMODE0	0	Level-sensitivity is selected for $\overline{\text{EINT0}}$ .	0
		1	$\overline{\text{EINT0}}$ is edge sensitive.	
1	EXTMODE1	0	Level-sensitivity is selected for $\overline{\text{EINT1}}$ .	0
		1	$\overline{\text{EINT1}}$ is edge sensitive.	
2	EXTMODE2	0	Level-sensitivity is selected for $\overline{\text{EINT2}}$ .	0
		1	$\overline{\text{EINT2}}$ is edge sensitive.	

Table 15. External Interrupt Mode register (EXTMODE - address 0x400F C148) bit description

Bit	Symbol	Value	Description	Reset value
3	EXTMODE3	0	Level-sensitivity is selected for $\overline{\text{EINT3}}$ .	0
		1	$\overline{\text{EINT3}}$ is edge sensitive.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 3.7.4 External Interrupt Polarity register (EXTPOLAR - 0x400F C14C)

In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. Only pins that are selected for the EINT function (see [Section 8.3](#)) and enabled in the appropriate NVIC register) can cause interrupts from the External Interrupt function (though of course pins selected for other functions may cause interrupts from those functions).

**Note:** Software should only change a bit in this register when its interrupt is disabled in the NVIC (state readable in the ISERN/ICERN registers), and should write the corresponding 1 to EXTINT before enabling (initializing) or re-enabling the interrupt. An extraneous interrupt(s) could be set by changing the polarity and not having the EXTINT cleared.

Table 16. External Interrupt Polarity register (EXTPOLAR - address 0x400F C14C) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTPOLAR0	0	$\overline{\text{EINT0}}$ is low-active or falling-edge sensitive (depending on EXTMODE0).	0
		1	$\overline{\text{EINT0}}$ is high-active or rising-edge sensitive (depending on EXTMODE0).	
1	EXTPOLAR1	0	$\overline{\text{EINT1}}$ is low-active or falling-edge sensitive (depending on EXTMODE1).	0
		1	$\overline{\text{EINT1}}$ is high-active or rising-edge sensitive (depending on EXTMODE1).	
2	EXTPOLAR2	0	$\overline{\text{EINT2}}$ is low-active or falling-edge sensitive (depending on EXTMODE2).	0
		1	$\overline{\text{EINT2}}$ is high-active or rising-edge sensitive (depending on EXTMODE2).	
3	EXTPOLAR3	0	$\overline{\text{EINT3}}$ is low-active or falling-edge sensitive (depending on EXTMODE3).	0
		1	$\overline{\text{EINT3}}$ is high-active or rising-edge sensitive (depending on EXTMODE3).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

## 3.8 Other system controls and status flags

Some aspects of controlling LPC178x/177x operation that do not fit into peripheral or other registers are grouped here.

### 3.8.1 System Controls and Status register (SCS - 0x400F C1A0)

The SCS register contains special control and status bits related to various aspects of chip operation. These functions are described in [Table 17](#).

Several of these bits apply to the main oscillator. Since chip operation always begins using the Internal RC Oscillator, and the main oscillator may not be used at all in some applications, it will only be started by software request. This is accomplished by setting the OSCEN bit in the SCS register, as described in Table 3-13. The main oscillator provides a status flag (the OSCSTAT bit in the SCS register) so that software can determine when the oscillator is running and stable. At that point, software can control switching to the main oscillator as a clock source. Prior to starting the main oscillator, a frequency range must be selected by configuring the OSCRANGE bit in the SCS register.

**Table 17. System Controls and Status register (SCS - address 0x400F C1A0) bit description**

Bit	Function	Value	Description	Access	Reset value
0	EMC Shift Control		Controls how addresses are output on the EMC address pins for static memories. Also see <a href="#">Section 10.9</a> in the EMC chapter.	R/W	1
		0	Static memory addresses are shifted to match the data bus width. For example, when accessing a 32-bit wide data bus, the address is shifted right 2 places such that bit 2 is the LSB. In this mode, address bit 0 for the this device is connected to address bit 0 of the memory device, thus simplifying memory connections. This also makes a larger memory address range possible, because additional upper address bits can appear on the higher address pins due to the shift.		
		1	Static memory addresses are always output as byte addresses regardless of the data bus width. For example, when word data is accessed on a 32-bit bus, address bits 1 and 0 will always be 0. In this mode, one or both lower address bits may not be connected to memories that are part of a bus that is wider than 8 bits. This mode matches the operation of LPC23xx and LPC24xx devices.		
1	EMC Reset Disable <sup>1)</sup>		External Memory Controller Reset Disable. Also see <a href="#">Section 10.8</a> in the EMC chapter.	R/W	0
		0	Both EMC resets are asserted when any type of chip reset event occurs. In this mode, all registers and functions of the EMC are initialized upon any reset condition.		
		1	Many portions of the EMC are only reset by a power-on or brown-out event, in order to allow the EMC to retain its state through a warm reset (external reset or watchdog reset). If the EMC is configured correctly, auto-refresh can be maintained through a warm reset.		
2	EMC Burst Control		External Memory Controller burst control. Also see <a href="#">Section 10.10</a> in the EMC chapter.	R/W	0
		0	Burst enabled.		
		1	Burst disabled. This mode can be used to prevent multiple sequential accesses to memory mapped I/O devices connected to EMC static memory chip selects. These unrequested accesses can cause issues with some I/O devices.		



Table 17. System Controls and Status register (SCS - address 0x400F C1A0) bit description

Bit	Function	Value	Description	Access	Reset value
3	MCIPWR Active Level <sup>[1]</sup>		Selects the active level of the SD card interface signal SD_PWR.	R/W	1
		0	SD_PWR is active low (inverted output of the SD Card interface block).		
		1	SD_PWR is active high (follows the output of the SD Card interface block).		
4	Oscillator Range Select		Main oscillator range select.	R/W	0
		0	The frequency range of the main oscillator is 1 MHz to 20 MHz.		
		1	The frequency range of the main oscillator is 15 MHz to 25 MHz.		
5	Oscillator Enable		Main oscillator enable.	R/W	0
		0	The main oscillator is disabled.		
		1	The main oscillator is enabled, and will start up if the correct external circuitry is connected to the XTAL1 and XTAL2 pins.		
6	Oscillator Status		Main oscillator status.	RO	0
		0	The main oscillator is not ready to be used as a clock source.		
		1	The main oscillator is ready to be used as a clock source. The main oscillator must be enabled via the OSCEN bit.		
31:7	-		Reserved. Read value is undefined, only zero should be written.	-	NA

[1] The state of this bit is preserved through a software reset, and only a POR or a BOD event will reset it to its default value.

### 4.1 Summary of clocking and power control functions

---

This section describes the generation of the various clocks needed by the LPC178x/177x and options of clock source selection, as well as power control and wake-up from reduced power modes. Functions described in the following subsections include:

- Oscillators (see [Section 4.3](#))
- PLLs (see [Section 4.5](#))
- Clock selection and dividers (see [Section 4.6](#))
- Power control (see [Section 4.7](#))
- Wake-up timer (see [Section 4.8](#))
- External clock output (see [Section 4.9](#))

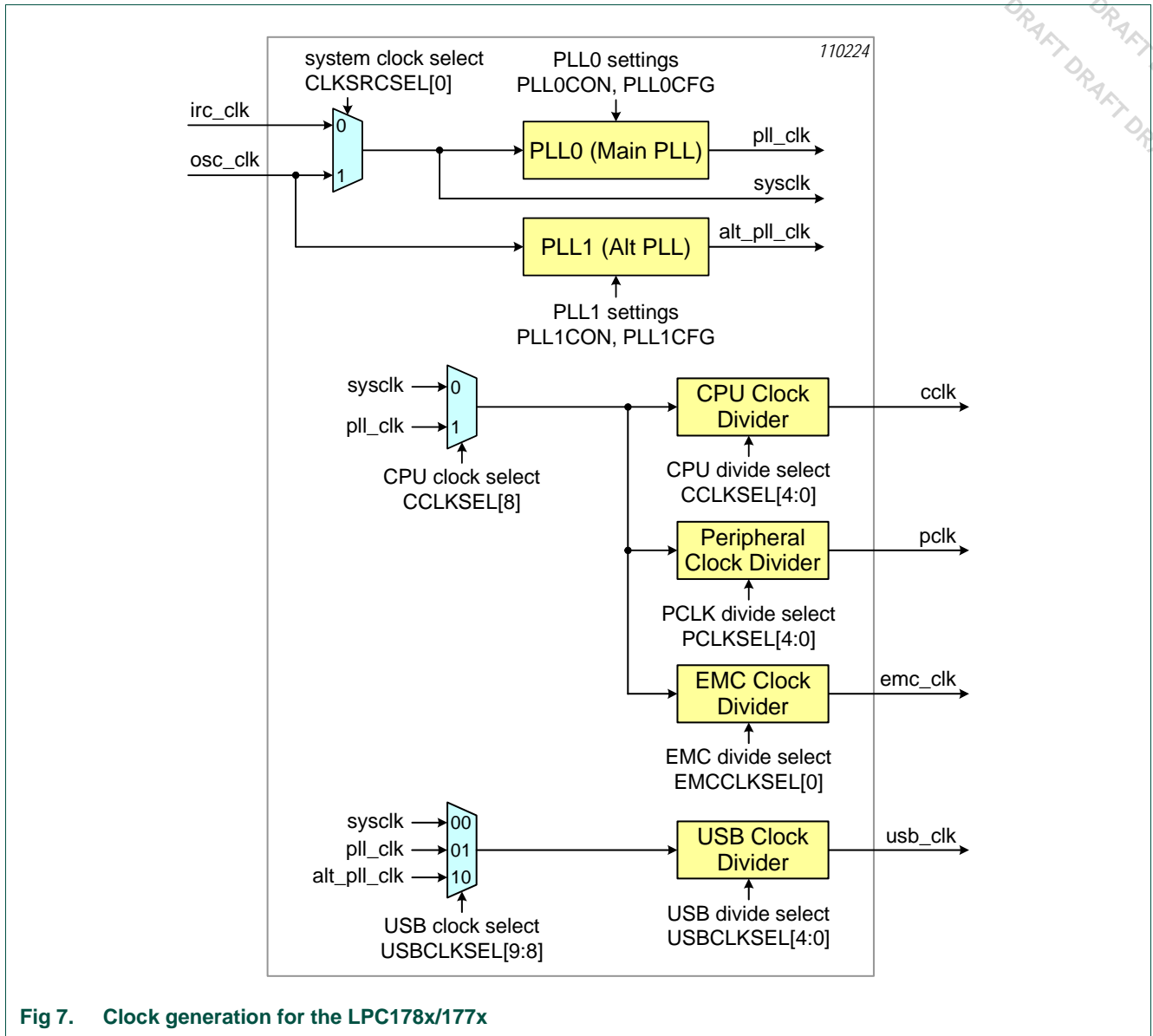


Fig 7. Clock generation for the LPC178x/177x

## 4.2 Register description

All registers, regardless of size, are on word address boundaries. Details of the registers appear in the description of each function.

**Table 18. Summary of system control registers**

Name	Description	Access	Reset value	Address	Table
<b>System clock source selection</b>					
CLKSRCSEL	Clock Source Select Register	R/W	0	0x400F C10C	<a href="#">21</a>
<b>Phase Locked Loop 0 (PLL0, Main PLL)</b>					
PLL0CON	PLL0 Control register	R/W	0	0x400F C080	<a href="#">23</a>
PLL0CFG	PLL0 Configuration register	R/W	0	0x400F C084	<a href="#">24</a>
PLL0STAT	PLL0 Status register	RO	0	0x400F C088	<a href="#">25</a>
PLL0FEED	PLL0 Feed register	WO	NA	0x400F C08C	<a href="#">26</a>
<b>Phase Locked Loop 1 (PLL1, Alt PLL)</b>					
PLL1CON	PLL1 Control register	R/W	0	0x400F C0A0	<a href="#">23</a>
PLL1CFG	PLL1 Configuration register	R/W	0	0x400F C0A4	<a href="#">24</a>
PLL1STAT	PLL1 Status register	RO	0	0x400F C0A8	<a href="#">25</a>
PLL1FEED	PLL1 Feed register	WO	NA	0x400F C0AC	<a href="#">26</a>
<b>Clock dividers</b>					
CCLKSEL	CPU Clock Selection register	R/W	1	0x400F C104	<a href="#">30</a>
USBCLKSEL	USB Clock Selection register	R/W	0	0x400F C108	<a href="#">31</a>
EMCCLKSEL	External Memory Controller Clock Selection register	R/W	0	0X400F C100	<a href="#">32</a>
PCLKSEL	Peripheral Clock Selection register	R/W	0x10	0x400F C1A8	<a href="#">33</a>
<b>Power control</b>					
PCON	Power Control register	R/W	0	0x400F C0C0	<a href="#">35</a>
PCONP	Power Control for Peripherals	R/W	0x0408 829E	0x400F C0C4	<a href="#">37</a>
<b>Utility</b>					
CLKOUTCFG	Clock Output Configuration register	R/W	0	0x400F C1C8	<a href="#">38</a>

## 4.3 Oscillators

The LPC178x/177x includes three independent oscillators. These are the Main Oscillator, the Internal RC Oscillator, and the RTC oscillator. Each oscillator can be used for more than one purpose as required in a particular application. This can be seen in [Figure 7](#).

Following Reset, the LPC178x/177x will operate from the Internal RC Oscillator until switched by software. This allows systems to operate without any external crystal, and allows the boot loader code to operate at a known frequency.

### 4.3.1 Internal RC oscillator

The Internal RC Oscillator (IRC) may be used as the clock source for the watchdog timer, and/or as the clock that drives PLL0 and subsequently the CPU. The precision of the IRC does not allow for use of the USB interface, which requires a much more precise time base in order to comply with the USB specification (only the main oscillator can meet that specification). Also, the IRC should not be used with the CAN1/2 block if the CAN baud rate is higher than 100 kbit/s. The IRC frequency is 12 MHz, factory trimmed to within  $\pm 1\%$  accuracy.

Upon power-up or any chip reset, the LPC178x/177x uses the IRC as the clock source. Software may later switch to one of the other available clock sources.

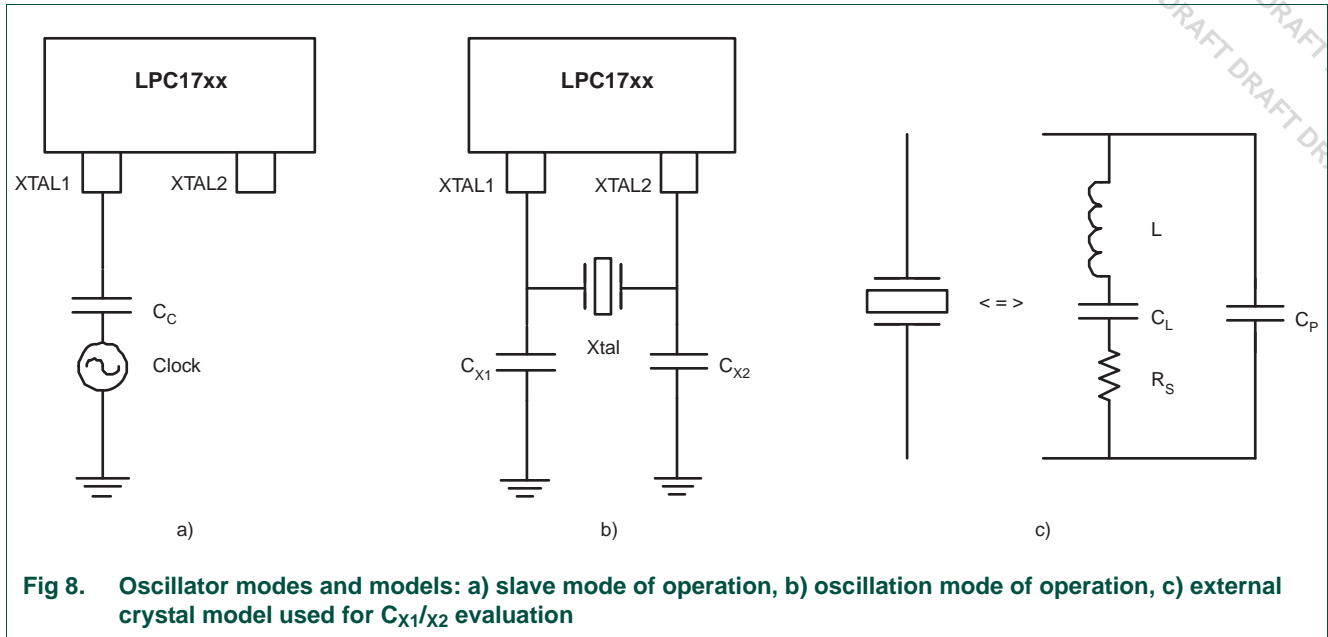
### 4.3.2 Main oscillator

The main oscillator can be used as the clock source for the CPU, with or without using PLL0. The main oscillator operates at frequencies of 1 MHz to 25 MHz. This frequency can be boosted to a higher frequency, up to the maximum CPU operating frequency, by the Main PLL (PLL0). The oscillator output is called OSC\_CLK. The clock selected as the PLL0 input is PLLCLKIN and the ARM processor clock frequency is referred to as CCLK for purposes of rate equations, etc. elsewhere in this document. The frequencies of PLLCLKIN and CCLK are the same value unless the PLL0 is active and connected. Refer to [Section 4.5](#) for details.

The on-board oscillator in the LPC178x/177x can operate in one of two modes: slave mode and oscillation mode.

In slave mode the input clock signal should be coupled by means of a capacitor of 100 pF ( $C_C$  in [Figure 8](#), drawing a), with an amplitude between 200 mVrms and 1000 mVrms. This corresponds to a square wave signal with a signal swing of between 280 mV and 1.4 V. The XTAL2 pin in this configuration can be left unconnected.

External components and models used in oscillation mode are shown in [Figure 8](#), drawings b and c, and in [Table 19](#) and [Table 20](#). Since the feedback resistance is integrated on chip, only a crystal and the capacitances  $C_{X1}$  and  $C_{X2}$  need to be connected externally in case of fundamental mode oscillation (the fundamental frequency is represented by  $L$ ,  $C_L$  and  $R_S$ ). Capacitance  $C_P$  in [Figure 8](#), drawing c, represents the parallel package capacitance and should not be larger than 7 pF. Parameters  $F_C$ ,  $C_L$ ,  $R_S$  and  $C_P$  are supplied by the crystal manufacturer.



**Table 19. Recommended values for  $C_{X1/X2}$  in oscillation mode (crystal and external components parameters) low frequency mode (OSCRANGE = 0, see Table 17)**

Fundamental oscillation frequency $F_{Osc}$	Crystal load capacitance $C_L$	Maximum crystal series resistance $R_S$	External load capacitors $C_{X1}, C_{X2}$
1 MHz - 5 MHz	10 pF	< 300 $\Omega$	18 pF, 18 pF
	20 pF	< 300 $\Omega$	39 pF, 39 pF
	30 pF	< 300 $\Omega$	57 pF, 57 pF
5 MHz - 10 MHz	10 pF	< 300 $\Omega$	18 pF, 18 pF
	20 pF	< 200 $\Omega$	39 pF, 39 pF
	30 pF	< 100 $\Omega$	57 pF, 57 pF
10 MHz - 15 MHz	10 pF	< 160 $\Omega$	18 pF, 18 pF
	20 pF	< 60 $\Omega$	39 pF, 39 pF
15 MHz - 20 MHz	10 pF	< 80 $\Omega$	18 pF, 18 pF

**Table 20. Recommended values for  $C_{X1/X2}$  in oscillation mode (crystal and external components parameters) high frequency mode (OSCRANGE = 1, see Table 17)**

Fundamental oscillation frequency $F_{Osc}$	Crystal load capacitance $C_L$	Maximum crystal series resistance $R_S$	External load capacitors $C_{X1}, C_{X2}$
15 MHz - 20 MHz	10 pF	< 180 $\Omega$	18 pF, 18 pF
	20 pF	< 100 $\Omega$	39 pF, 39 pF
20 MHz - 25 MHz	10 pF	< 160 $\Omega$	18 pF, 18 pF
	20 pF	< 80 $\Omega$	39 pF, 39 pF

#### 4.3.2.1 Main oscillator startup

Since chip operation always begins using the Internal RC Oscillator, and the main oscillator may not be used at all in some applications, it will only be started by software request. This is accomplished by setting the OSCEN bit in the SCS register, as described in [Table 17](#). The main oscillator provides a status flag (the OSCSTAT bit in the SCS register) so that software can determine when the oscillator is running and stable. At that point, software can control switching to the main oscillator as a clock source. Prior to starting the main oscillator, a frequency range must be selected by configuring the OSCRANGE bit in the SCS register.

#### 4.3.3 RTC oscillator

The RTC oscillator provides a 1 Hz clock to the RTC and a 32 kHz clock output that can be output on the CLKOUT pin in order to allow trimming the RTC oscillator without interference from a probe.

#### 4.3.4 Watchdog oscillator

The Watchdog Timer has a dedicated oscillator that provides a 500 kHz clock to the Watchdog Timer that is always running if the Watchdog Timer is enabled. The Watchdog oscillator clock can be output on the CLKOUT pin in order to allow observe its frequency.

In order to allow Watchdog Timer operation with minimum power consumption, which can be important in reduced power modes, the Watchdog oscillator frequency is not tightly controlled. The Watchdog oscillator frequency will vary over temperature and power supply within a particular part, and may vary by processing across different parts. This variation should be taken into account when determining Watchdog reload values.

Within a particular part, temperature and power supply variations can produce up to a  $\pm 17\%$  frequency variation. Frequency variation between devices under the same operating conditions can be up to  $\pm 30\%$ .

## 4.4 Clock source selection multiplexer

Two clock sources may be chosen to drive the system clock (sysclk) and PLL0. These are the Internal RC oscillator and the main oscillator.

The clock source selection should only be changed safely when PLL0 is not connected. For a detailed description of how to change the clock source in a system using PLL0 see [Section 4.5.11 "PLL configuration sequence"](#).

### 4.4.1 Clock Source Selection register (CLKSRCSEL - 0x400F C10C)

The CLKSRCSEL register controls selection of the clock used for sysclk and PLL0.

**Table 21. Clock Source Selection register (CLKSRCSEL - 0x400F C10C) bit description**

Bit	Symbol	Value	Description	Reset value
0	CLKSRC		Selects the clock source for sysclk and PLL0 as follows:	0
		0	Selects the Internal RC oscillator as the sysclk and PLL0 clock source (default).	
		1	Selects the main oscillator as the sysclk and PLL0 clock source.	
31:1	-		Reserved. Read value is undefined, only zero should be written.	NA



## 4.5 PLL0 and PLL1 (Phase Locked Loops)

PLL0 (also called the Main PLL) and PLL1 (also called the Alt PLL) are functionally identical, but have somewhat different input possibilities and output connections. These possibilities are shown in [Figure 7](#). The Main PLL can receive its input from either the IRC or the Main Oscillator, and can potentially be used to provide the clocks to nearly everything on the device. The Alt PLL receives its input only from the main oscillator and is intended to be used as an alternate source of clocking to the USB. This peripheral has timing needs that may not always be filled by the Main PLL.

Both PLLs are disabled and powered off on reset. If the Alt PLL is left disabled, the USB clock can be supplied by PLL0 if everything is set up to provide 48 MHz to the USB clock through that route. The source for each clock must be selected via the CLKSEL registers (see [Section 4.6](#)), and can be further reduced by clock dividers as needed.

PLL activation is controlled via the PLLCON registers. PLL multiplier and divider values are controlled by the PLLCFG registers. The PLLCFG registers are protected in order to prevent accidental deactivation of PLLs or accidental alteration PLL operating parameters. The protection is accomplished by a feed sequence similar to that of the Watchdog Timer. Details are provided in the descriptions of the PLLFEED registers.

PLL0 accepts an input clock frequency from either the IRC or the main oscillator. If only the Main PLL is used, then its output frequency must be an integer multiple of all other clocks needed in the system. PLL1 takes its input only from the main oscillator, requiring an external crystal in the range of 10 to 25 MHz. In each PLL, the Current Controlled Oscillator (CCO) operates in the range of 156 MHz to 320 MHz, so there are additional dividers to bring the output down to the desired frequencies. The minimum output divider value is 2, insuring that the output of the PLLs have a 50% duty cycle. [Figure 9](#) shows a block diagram of PLL internal connections.

If the USB is used, the possibilities for the CPU clock and other clocks will be limited by the requirements that the frequency be precise and very low jitter, and that the PLL0 output must be a multiple of 48 MHz. Even multiples of 48 Mhz that are within the operating range of the PLL are 192 and 288 MHz. Also, only the main oscillator in conjunction with the PLL can meet the precision and jitter specifications for USB. It is due to these limitations that the Alt PLL is provided.

The Alt PLL accepts an input clock frequency from the main oscillator in the range of 10 MHz to 25 MHz only. When used as the USB clock, the input frequency is multiplied up to a multiple of 48 MHz (192 or 288 MHz as described above).

### 4.5.1 PLL and startup/boot code interaction

When there is no valid user code (determined by the checksum word) in the user flash or the ISP enable pin (P2[10]) is pulled low on startup, the ISP mode will be entered and the boot code will setup the Main PLL with the IRC. Therefore it can not be assumed that the Main PLL is disabled when the user opens a debug session to debug the application code. The user startup code must follow the steps described in this chapter to disconnect the Main PLL.

### 4.5.2 PLL register description

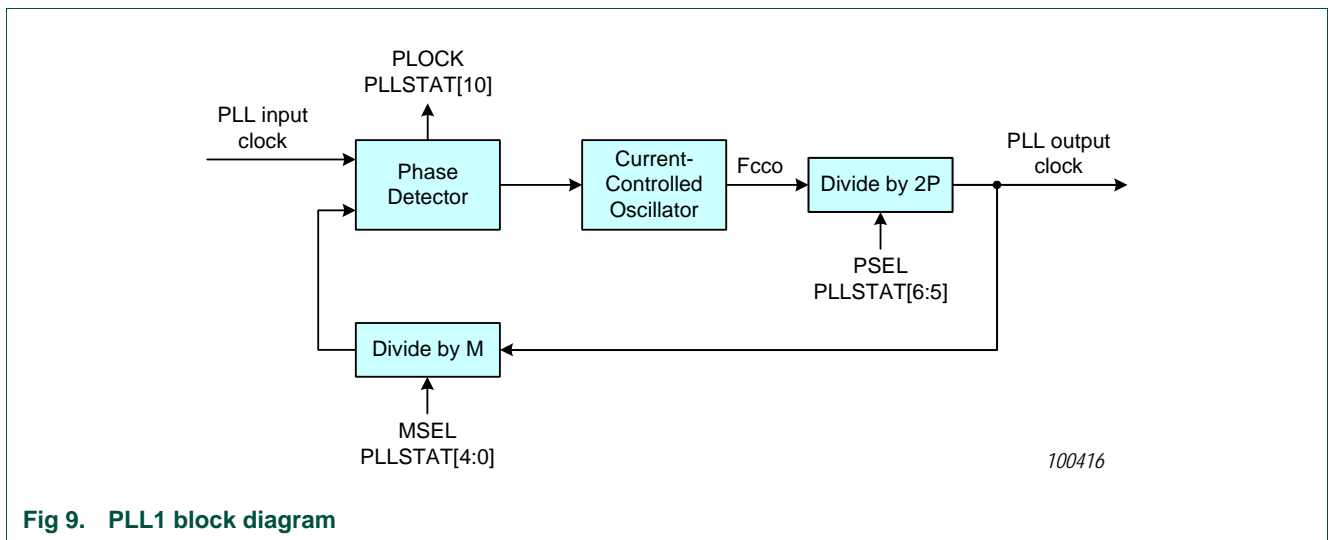
The PLLs are controlled by the registers shown in [Table 22](#). More detailed descriptions follow.

**Warning: Improper setting of PLL values may result in incorrect operation of the USB subsystem!**

**Table 22. PLL1 registers**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	PLLn Register Name and Address	Table
PLLCON	PLL Control Register. Holding register for updating PLL control bits. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W	0	PLL0CON - 0x400F C080 PLL1CON - 0x400F C0A0	<a href="#">23</a>
PLLCFG	PLL Configuration Register. Holding register for updating PLL configuration values. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W	0	PLL0CFG - 0x400F C084 PLL1CFG - 0x400F C0A4	<a href="#">24</a>
PLLSTAT	PLL Status Register. Read-back register for PLL control and configuration information. If PLLCON or PLLCFG have been written to, but a PLL feed sequence has not yet occurred, they will not reflect the current PLL state. Reading this register provides the actual values controlling PLL, as well as PLL status.	RO	0	PLL0STAT - 0x400F C088 PLL1STAT - 0x400F C0A8	<a href="#">25</a>
PLLFEED	PLL Feed Register. This register enables loading of PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation.	WO	NA	PLL0FEED - 0x400F C08C PLL1FEED - 0x400F C0AC	<a href="#">26</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.



**Fig 9. PLL1 block diagram**

### 4.5.3 PLL Control registers (PLL0CON - 0x400F C080, PLL1CON - 0x400F C0A0)

The PLLCON registers contains the bits that enable and connect each PLL. Enabling a PLL allows it to attempt to lock to the current settings of the multiplier and divider values. Changes to a PLLCON register do not take effect until a correct PLL feed sequence has been given for that PLL (see [Section 4.5.7](#) and [Section 4.5.4](#)).

**Table 23. PLL Control registers (PLL0CON - 0x400F C080, PLL1CON - 0x400F C0A0) bit description**

Bit	Symbol	Description	Reset value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the related PLL and allow it to lock to the requested frequency. See PLLSTAT register, <a href="#">Table 25</a> .	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	NA

Each PLL must be set up, enabled, and lock established before it may be used as a clock source. The hardware does not insure that the PLL is locked before it is selected nor does it automatically disconnect the PLL if lock is lost during operation.

### 4.5.4 PLL Configuration registers (PLL0CFG - 0x400F C084, PLL1CFG - 0x400F C0A4)

The PLLCFG register contains the PLL multiplier and divider values. Changes to the PLLCFG register do not take effect until a correct PLL feed sequence has been given (see [Section 4.5.7](#)). Calculations for the PLL frequency, and multiplier and divider values are found in [Section 4.5.10](#).

**Table 24. PLL Configuration registers (PLL0CFG - 0x400F C084, PLL1CFG - 0x400F C0A4) bit description**

Bit	Symbol	Description	Reset value
4:0	MSEL	PLL Multiplier value. Supplies the value "M" in the PLL frequency calculations. <b>Note:</b> For details on selecting the right value for MSEL see <a href="#">Section 4.5.9</a> .	0
6:5	PSEL	PLL Divider value. Supplies the value "P" in the PLL frequency calculations. <b>Note:</b> For details on selecting the right value for PSEL see <a href="#">Section 4.5.9</a> .	0
31:7	-	Reserved. Read value is undefined, only zero should be written.	NA

### 4.5.5 PLL Status registers (PLL0STAT - 0x400F C088, PLL1STAT - 0x400F C0A8)

The read-only PLLSTAT register provides the actual PLL parameters that are in effect at the time it is read, as well as the PLL status. PLLSTAT may disagree with values found in PLLCON and PLLCFG because changes to those registers do not take effect until a proper PLL feed has occurred (see [Section 4.5.7 "PLL Feed registers \(PLL0FEED - 0x400F C08C, PLL1FEED - 0x400F C0AC\)"](#)).

**Table 25. PLL Status registers (PLL0STAT - 0x400F C088, PLL1STAT - 0x400F C0A8) bit description**

Bit	Symbol	Description	Reset value
4:0	MSEL	Read-back for the PLL Multiplier value. This is the value currently used by the related PLL.	0
6:5	PSEL	Read-back for the PLL Divider value. This is the value currently used by the related PLL.	0
7	-	Reserved. The value read from a reserved bit is not defined.	NA

**Table 25. PLL Status registers (PLL0STAT - 0x400F C088, PLL1STAT - 0x400F C0A8) bit description**

Bit	Symbol	Description	Reset value
8	PLLE_STAT	Read-back for the PLL Enable bit. When one, the related PLL is currently activated. When zero, the related PLL is turned off. This bit is automatically cleared when Power-down mode is activated.	0
9	-	Reserved. The value read from a reserved bit is not defined.	NA
10	PLOCK	Reflects the PLL Lock status. When zero, the related PLL is not locked. When one, the related PLL is locked onto the requested frequency.	0
31:11	-	Reserved. The value read from a reserved bit is not defined.	NA

#### 4.5.6 PLL Interrupts: PLOCK0 and PLOCK1

The PLOCK bit in the PLLSTAT register reflects the lock status of the related PLL1. When the PLL is first enabled, or when its parameters are changed, the PLL requires some time to establish lock under the new conditions. The related PLOCK bit can be monitored to determine when the PLL may be connected for use.

Each PLOCK bit is connected to the interrupt controller. This allows for software to turn on the PLL and continue with other functions without having to wait for the PLL to achieve lock. When the interrupt occurs, the PLL may be selected as a clock source, and the interrupt disabled. PLOCK0 and PLOCK1 appear as exception numbers 32 and 48 respectively in [Table 41](#). Note that each PLOCK bit remains asserted whenever the related PLL is locked, so if the interrupt is used, the interrupt service routine must disable the interrupt prior to exiting.

#### 4.5.7 PLL Feed registers (PLL0FEED - 0x400F C08C, PLL1FEED - 0x400F C0AC)

A correct feed sequence must be written to the related PLLFEED register in order for changes to the related PLLCON and PLLCFG registers to take effect. The feed sequence is:

1. Write the value 0xAA to PLLFEED.
2. Write the value 0x55 to PLLFEED.

The two writes must be in the correct sequence, and there must be no other register access in the same address space (0x400F C000 to 0x400F FFFF) between them. Because of this, it may be necessary to disable interrupts for the duration of the PLL feed operation, if there is a possibility that an interrupt service routine could write to another register in that space. If either of the feed values is incorrect, or one of the previously mentioned conditions is not met, any changes to the PLLCON or PLLCFG register will not become effective.

**Table 26. PLL Feed registers (PLL0FEED - 0x400F C08C, PLL1FEED - 0x400F C0AC) bit description**

Bit	Symbol	Description	Reset value
7:0	PLLFEED	The PLL feed sequence must be written to this register in order for the related PLL's configuration and control register changes to take effect.	0x00
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 4.5.8 PLLs and Power-down mode

Power-down mode automatically turns off and disconnects activated PLLs, while subsequent wake-up from Power-down mode does not automatically restore PLL settings. This must be done in software. Typically, a routine to activate the PLL(s), wait for lock, and then select the PLL(s) can be called at the beginning of any interrupt service routine that might be called due to the wake-up.

If activity on the USB data lines is not selected to wake the microcontroller from Power-down mode (see [Section 4.7.8](#) for details of wake up from reduced modes), both the Main PLL (PLL0) and the Alt PLL (PLL1) will be automatically be turned off and disconnected when Power-down mode is invoked, as described above. However, if the USB activity interrupt is enabled and `USB_NEED_CLK = 1` (see [Table 253](#) for a description of `USB_NEED_CLK`), it is not possible to go into Power-down mode and any attempt to set the PD bit will fail, leaving the PLLs in the current state.

### 4.5.9 PLL frequency calculation

Equations for both the Main and Alt PLLs use the following parameters:

**Table 27. Elements determining PLL frequency**

Element	Description
pll_in_clk	the frequency of the input to the PLL
F <sub>CCO</sub>	the frequency of the PLL current controlled oscillator
pll_out_clk	the PLL output frequency
M	PLL Multiplier value from the MSEL bits in the PLLCFG register
P	PLL Divider value from the PSEL bits in the PLLCFG register

The PLL output frequency (when the PLL is active and locked) is given by:

$$\text{pll\_out\_clk} = M \times \text{pll\_in\_clk} \quad \text{- or -} \quad \text{pll\_out\_clk} = F_{\text{CCO}} / (2 \times P)$$

The CCO frequency can be computed as:

$$F_{\text{CCO}} = \text{pll\_out\_clk} \times 2 \times P \quad \text{- or -} \quad F_{\text{CCO}} = \text{pll\_in\_clk} \times M \times 2 \times P$$

The PLL inputs and settings must meet the following criteria:

- M is in the range of 1 to 32.
- P is one of 1, 2, 4, 8.
- pll\_in\_clk is in the range of 10 MHz to 25 MHz.
- F<sub>CCO</sub> is in the range of 156 MHz to 320 MHz.
- pll\_out\_clk is in the range of 9.75 MHz to 160 MHz.

### 4.5.10 Procedure for determining PLL settings

In general, PLL configuration values may be found as follows:

1. Based on the desired PLL output frequency, choose an oscillator frequency (F<sub>OSC</sub>). If the USB interface is to be used, an external crystal of either 12 MHz, 16 MHz, or 24 MHz must be provided. 12 MHz is recommended for this purpose in order to save power and have more flexibility with PLL settings.
2. If the USB interface is used in the system, and if a PLL output of 96 MHz or 144 MHz can provide the desired CPU clock frequency, it is probably possible to use only PLL0.
3. Calculate the value of M to configure the MSEL1 bits to obtain the desired PLL output frequency.  $M = \text{pll\_out\_clk} / \text{pll\_in\_clk}$ . The value written to the MSEL bits in the PLLCFG register is M – 1 (or see [Table 28](#)). This is done for both PLLs if they are both used.

- Find a value for P to configure the PSEL bits, such that  $F_{CCO}$  is within its defined operating frequency limits of 156 MHz to 320 MHz.  $F_{CCO}$  is calculated using  $F_{CCO} = pll\_out\_clk \times 2 \times P$ . The value written to the PSEL bits in PLLCFG can be found in [Table 29](#).

**Table 28. PLL Multiplier values**

Value of M	MSEL Bits (PLLCFG bits [4:0])	MSEL hex
1	00000	0
2	00001	0x01
3	00010	0x02
4	00011	0x03
5	00100	0x04
6	00101	0x05
7	00110	0x06
8	00111	0x07
9	01000	0x08
10	01001	0x09
11	01010	0x0A
12	01011	0x0B
13	01100	0x0C
14	01101	0x0D
15	01110	0x0E
16	01111	0x0F
17	10000	0x10
18	10001	0x11
19	10010	0x12
20	10011	0x13
21	10100	0x14
22	10101	0x15
23	10110	0x16
24	10111	0x17
25	11000	0x18
26	11001	0x19
27	11010	0x1A
28	11011	0x1B
29	11100	0x1C
30	11101	0x1D
31	11110	0x1E
32	11111	0x1F

Table 29. PLL Divider values

Value of P	PSEL Bits (PLLCFG bits [6:5])	PSEL hex
1	00	0
2	01	0x1
4	10	0x2
8	11	0x3

#### 4.5.11 PLL configuration sequence

The following discussions refer to PLLs and PLL related registers generically (e.g. PLLCFG rather than PLL0CFG or PLL1CFG). The instructions have to be adapted to the specific case being addressed in the application.

##### To set up a PLL and switch clocks to its output:

1. Make sure that the PLL output is not already being used. The CCLKSEL and USBCLKSEL registers must not select the PLL being set up (see [“To switch clocks away from a PLL output:”](#) below). Clock dividers included in these registers may also be set up at this time if writing to any of the noted registers.
2. If the main PLL is being set up, and the main clock source is being changed (IRC versus main oscillator), change this first by writing the correct value to the CLKSRCSEL register.
3. Write PLL new setup values to the PLLCFG register. Write a 1 to the PLLE bit in the PLLCON register. Perform a PLL feed sequence by writing first the value 0xAA, then the value 0x55 to the PLLFEED register.
4. Set up the necessary clock dividers. These may include the CCLKSEL, PCLKSEL, EMCCCLKSEL, and the USBCLKSEL registers.
5. Wait for the PLL to lock. This may be accomplished by polling the PLLSTAT register and testing for PLOCK = 1, or by using the PLL lock interrupt.
6. Connect the PLL by selecting its output in the appropriate places. This may include the CCLKSEL and USBCLKSEL registers.

##### To switch clocks away from a PLL output:

1. To switch back to the mode of not using a PLL, write values to any or all of the CCLKSEL and USBCLKSEL registers in order to select a different clock source.
2. The related PLL may now be turned off by writing to the PLLCON register and performing a PLL feed sequence, reconfigured by writing to the PLLCFG register, etc.

#### 4.5.12 PLL configuration examples

The following examples illustrate selecting PLL values based on different system requirements.

##### Example 1).

Assumptions:

- The system design is planned to use the IRC to generate the CPU clock.
- A frequency as close to 80 MHz as possible is desired for the CPU clock.



Of the two PLLs, only PLL0 can supply the CPU clock, so this example is for PLL0. The nearest multiple of the 12 MHz IRC frequency to 80 MHz is 84 MHz. Since  $\text{pll\_out\_clk} = M \times \text{pll\_in\_clk}$ ,  $M = \text{pll\_out\_clk} / \text{pll\_in\_clk} = 84 / 12 = 7$ .

Now a value for P must be found that puts  $F_{\text{CCO}}$  within the PLL operating range of 156 MHz to 320 MHz.  $F_{\text{CCO}} = \text{pll\_out\_clk} \times 2 \times P$ . Start by finding the value of  $F_{\text{CCO}}$  with  $P = 1$ , which is  $84 \text{ MHz} \times 2 = 168 \text{ MHz}$ . Since that is within the PLL operating range, no further work is needed.

Set up the PLL for  $M = 7$  and  $P = 1$ . This requires putting the value 6 ( $M - 1$ , or see [Table 28 "PLL Multiplier values"](#)) in the MSEL field of the PLL0CFG register. A value of 0 (see [Table 29 "PLL Divider values"](#)) is needed in the PSEL field of PLL0CFG. A single write of both values would be  $\text{PLL0CFG} = 0x06$ . See [Section 4.5.11](#) for a description of the PLL setup sequence.

### Example 2).

Assumptions:

- The system design is planned to use a 12 MHz crystal generate both the CPU clock and the USB clock.
- A frequency close to 100 MHz is desired for the CPU clock.

Of the two PLLs, only PLL0 can supply both the CPU clock and the USB clock, so this example is for PLL0. The PLL output must be an even integer multiple of 48 MHz for the USB to operate correctly (i.e. a multiple of 96 MHz). Two multiples of 96 MHz fit within the PLL operating range: 192 MHz ( $2 \times 96 \text{ MHz}$ ), and 288 MHz ( $3 \times 96 \text{ MHz}$ ). Of these, only 192 MHz can produce a CPU clock near 100 MHz (96 MHz). So, a 96 MHz PLL output can be used to obtain the 2 needed frequencies. Since  $\text{pll\_out\_clk} = M \times \text{pll\_in\_clk}$ ,  $M = \text{pll\_out\_clk} / \text{pll\_in\_clk} = 96 / 12 = 8$ .

Now a value for P must be found that puts  $F_{\text{CCO}}$  within the PLL operating range of 156 MHz to 320 MHz.  $F_{\text{CCO}} = \text{pll\_out\_clk} \times 2 \times P$ . Start by finding the value of  $F_{\text{CCO}}$  with  $P = 1$ , which is  $96 \text{ MHz} \times 2 = 192 \text{ MHz}$ . Since that is within the PLL operating range, no further work is needed.

Set up the PLL for  $M = 8$  and  $P = 1$ . This requires putting the value 7 ( $M - 1$ , or see [Table 28](#)) in the MSEL field of the PLL0CFG register. A value of 0 (see [Table 29](#)) is needed in the PSEL field of PLL0CFG. A single write of both values would be  $\text{PLL0CFG} = 0x07$ . See [Section 4.5.11](#) for a description of the PLL setup sequence.

### Example 3)

Assumptions:

- The system design will use the USB interface.
- It is desired that the CPU clock remain flexible and able to operate at frequencies unrelated to the USB clock.

In order to keep the CPU clock separate from the USB clock, the CPU will use PLL0. For USB, PLL1 may be configured with the same values used in the last example. PLL0 can be operated from either the IRC or the main oscillator to obtain whatever frequency is needed, and the PLL0 setup can be changed without compromising USB operation.

## 4.6 Clock selection and division

The output of each PLL that is used must be divided down to whatever frequency is needed by each subsystem. On the LPC178x/177x, there are separate clocks for the CPU, External Memory Controller, USB interface, and peripherals on the APB buses. Separate clock selection multiplexers and clock dividers provide flexibility in the generation of these clocks.

### 4.6.1 CPU Clock Selection register (CCLKSEL - 0x400F C104)

The CCLKSEL register controls selection of the clock used as the Main PLL input, and also controls the division of the PLL0 output before it is used by the CPU. When PLL0 is bypassed, the division may be by 1. When PLL0 is running, the output must be divided in order to bring the CPU clock frequency (CCLK) within operating limits. A 5-bit divider allows a range of options, including slowing CPU operation to a low rate for temporary power savings without turning off PLL0. Note that the CPU clock rate should not be set lower than the peripheral clock rate.

The two clock sources that may be chosen to drive PLL0 and ultimately the CPU and on-chip peripheral devices are the main oscillator and the Internal RC oscillator.

The clock source selection for PLL0 can only be changed safely when PLL0 is not being used. For a detailed description of how to change the clock source in a system using PLL0 see [Section 4.5.12 “PLL configuration examples”](#).

Note the following restrictions regarding the choice of clock sources:

- The IRC oscillator should not be used (via PLL0) as the clock source for the USB subsystem.
- The IRC oscillator should not be used (via PLL0) as the clock source for the CAN controllers if the CAN baud rate is higher than 100 kbit/s.

**Table 30. CPU Clock Selection register (CCLKSEL - address 0x400F C104) bit description**

Bit	Symbol	Value	Description	Reset value
4:0	CCLKDIV		Selects the divide value for creating the CPU clock (CCLK) from the selected clock source.	0x01
		0	The divider is turned off., no clock will be provided to the CPU. This setting should typically not be used, the CPU will be halted and a reset will be required to restore operation.	
		1	The input clock is divided by 1 to produce the CPU clock.	
		2	The input clock is divided by 2 to produce the CPU clock.	
		3	The input clock is divided by 3 to produce the CPU clock.	
		:	:	
		31	The input clock is divided by 31 to produce the CPU clock.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA
8	CCLKSEL		Selects the input clock for the CPU clock divider.	0
		0	Sysclk is used as the input to the CPU clock divider.	
		1	The output of the Main PLL is used as the input to the CPU clock divider.	
31:9	-		Reserved. Read value is undefined, only zero should be written.	NA

### 4.6.2 USB Clock Selection register (USBCLKSEL - 0x400F C108)

The USBCLKSEL register controls selection of the clock used for the USB subsystem, and also controls the division of the that clock before it is used by the USB. The output of the selected PLL must be divided in order to bring the USB clock frequency to 48 MHz with a 50% duty cycle. A divider allows obtaining the correct USB clock from any even multiple of 48 MHz (i.e. any multiple of 96 MHz) within the PLL operating range.

**Remark:** A clock derived from the Internal RC oscillator should not be used to clock the USB subsystem.

**Table 31. USB Clock Selection register (USBCLKSEL - address 0x400F C108) bit description**

Bit	Symbol	Value	Description	Reset value
4:0	USBDIV		Selects the divide value for creating the USB clock from the PLL0 output. Only the values shown below can produce even number multiples of 48 MHz from the PLL0 output.  <b>Warning:</b> Improper setting of this value will result in incorrect operation of the USB interface. Only the main oscillator in conjunction with either PLL0 or PLL1 can provide a clock that meets USB accuracy and jitter specifications.	0
		0	The divider is turned off., no clock will be provided to the USB subsystem.	
		4	PLL0 output is divided by 4. PLL0 output must be 192 MHz.	
		6	PLL0 output is divided by 6. PLL0 output must be 288 MHz.	
		Others	Other values cannot produce the 48 MHz clock required for USB operation.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA
9:8	USBSEL		Selects the input clock for the USB clock divider.	0
		00	Sysclk is used as the input to the USB clock divider. When this clock is selected, the USB can be accessed by software but cannot perform USB functions.	
		01	The output of the Main PLL is used as the input to the USB clock divider.	
		10	The output of the Alt PLL is used as the input to the USB clock divider.	
		11	Reserved, this setting should not be used.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 4.6.3 EMC Clock Selection register (EMCCLKSEL - 0x400F C100)

The EMCCLKSEL register controls division of the clock before it is used by the EMC. The EMC uses the same base clock as the CPU and the APB peripherals. The EMC clock can be the same as the CPU clock, or half that. This is intended to be used primarily when the CPU is running faster than the external bus can support.

**Table 32. EMC Clock Selection register (EMCCLKSEL - address 0x400F C100) bit description**

Bit	Symbol	Value	Description	Reset value
0	EMCDIV		Selects the EMC clock rate relative to the CPU clock.	0
		0	The EMC uses the same clock as the CPU.	
		1	The EMC uses a clock at half the rate of the CPU.	
31:1	-		Reserved. Read value is undefined, only zero should be written.	NA

**4.6.4 Peripheral Clock Selection register (PCLKSEL - 0x400F C1A8)**

The PCLKSEL register controls the base clock used for all APB peripherals. A 5-bit divider allows a range of frequencies to be used. Note that the peripheral clock rate should not be set higher than the CPU clock rate.

**Table 33. Peripheral Clock Selection register (PCLKSEL - address 0x400F C1A8) bit description**

Bit	Symbol	Value	Description	Reset value
4:0	PCLKDIV		Selects the divide value for the clock used for all APB peripherals.	0x04
		0	The divider is turned off., no clock will be provided to APB peripherals.	
		1	The input clock is divided by 1 to produce the APB peripheral clock.	
		2	The input clock is divided by 2 to produce the APB peripheral clock.	
		3	The input clock is divided by 3 to produce the APB peripheral clock.	
		4	The input clock is divided by 4 to produce the APB peripheral clock.	
		:	:	
		31	The input clock is divided by 31 to produce the APB peripheral clock.	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

## 4.7 Power control

The LPC178x/177x supports a variety of power control features: Sleep mode, Deep Sleep mode, Power-down mode, and Deep Power-down mode. The CPU clock rate may also be controlled as needed by changing clock sources, re-configuring PLL values, and/or altering the CPU clock divider value. This allows a trade-off of power versus processing speed based on application requirements. In addition, Peripheral Power Control allows shutting down the clocks to individual on-chip peripherals, allowing fine tuning of power consumption by eliminating all dynamic power use in any peripherals that are not required for the application.

Entry to any reduced power mode begins with the execution of either a WFI (Wait For Interrupt) or WFE (Wait For Exception) instruction by the Cortex-M3. The Cortex-M3 internally supports two reduced power modes: Sleep and Deep Sleep. These are selected by the SLEEPDEEP bit in the cortex-M3 System Control Register. Power-down and Deep Power-down modes are selected by bits in the PCON register. See [Table 35](#). The same register contains flags that indicate whether entry into each reduced power mode actually occurred.

The LPC178x/177x also implements a separate power domain in order to allow turning off power to the bulk of the device while maintaining operation of the Real Time Clock.

Reduced power modes have some limitation during debug, see [Section 37.7](#) for more information.

### 4.7.1 Sleep mode

**Note:** Sleep mode on the LPC178x/177x corresponds to the Idle mode on LPC2xxx series devices. The name is changed because ARM has incorporated portions of reduced power mode control into the Cortex-M3. LPC178x/177x documentation uses the Cortex-M3 terminology where applicable.

When Sleep mode is entered, the clock to the core is stopped, and the SMFLAG bit in PCON is set, see [Table 35](#). Resumption from the Sleep mode does not need any special sequence but re-enabling the clock to the ARM core.

In Sleep mode, execution of instructions is suspended until either a Reset or an interrupt occurs. Peripheral functions continue operation during Sleep mode and may generate interrupts to cause the processor to resume execution. Sleep mode eliminates dynamic power used by the processor itself, memory systems and related controllers, and internal buses.

The DMA controller can continue to work in Sleep mode, and has access to the peripheral RAMs and all peripheral registers. The flash memory and the main SRAM are not available in Sleep mode, they are disabled in order to save power.

Wake-up from Sleep mode will occur whenever any enabled interrupt occurs.

### 4.7.2 Deep Sleep mode

**Note:** Deep Sleep mode on the LPC178x/177x corresponds to the Sleep mode on LPC23xx and LPC24xx series devices. The name is changed because ARM has incorporated portions of reduced power mode control into the Cortex-M3. LPC178x/177x documentation uses the Cortex-M3 terminology where applicable.

When the chip enters the Deep Sleep mode, the main oscillator is powered down, nearly all clocks are stopped, and the DSFLAG bit in PCON is set, see [Table 35](#). The IRC remains running for fast startup. The 32 kHz RTC oscillator is not stopped and RTC interrupts may be used as a wake-up source. The flash is left in the standby mode allowing a quick wake-up. The PLLs are automatically turned off and the clock selection multiplexers are set to use sysclk (the reset state). The clock divider control registers are automatically reset to zero.

The processor state and registers, peripheral registers, and internal SRAM values are preserved throughout Deep Sleep mode and the logic levels of chip pins remain static. The Deep Sleep mode can be terminated and normal operation resumed by either a Reset or certain specific interrupts that are able to function without clocks. Since all dynamic operation of the chip is suspended, Deep Sleep mode reduces chip power consumption to a very low value.

On the wake-up of Deep Sleep mode, if the IRC was used before entering Deep Sleep mode, a 2-bit IRC timer starts counting and the code execution and peripherals activities will resume after the timer expires (4 cycles). If the main oscillator is used, the 12-bit main oscillator timer starts counting and the code execution will resume when the timer expires (4096 cycles). The user must remember to re-configure any required PLLs and clock dividers after the wake-up.

Wake-up from Deep Sleep mode can be brought about by NMI, External Interrupts EINT0 through EINT3, GPIO interrupts, the Ethernet Wake-on-LAN interrupt, Brownout Detect, an RTC Alarm interrupt, a USB input pin transition (USB activity interrupt), a CAN input pin transition, or a Watchdog Timer timeout, when the related interrupt is enabled. Wake-up will occur whenever any enabled interrupt occurs.

### 4.7.3 Power-down mode

Power-down mode does everything that Deep Sleep mode does, but also turns off the flash memory. Entry to Power-down mode causes the PDFLAG bit in PCON to be set, see [Table 35](#). This saves more power, but requires waiting for resumption of flash operation before execution of code or data access in the flash memory can be accomplished.

When the chip enters Power-down mode, the IRC, the main oscillator, and all clocks are stopped. The RTC remains running if it has been enabled and RTC interrupts may be used to wake up the CPU. The flash is forced into Power-down mode. The PLLs are automatically turned off and the clock selection multiplexers are set to use sysclk (the reset state). The clock divider control registers are automatically reset to zero. If the Watchdog timer is running, it will continue running in Power-down mode.

Upon wake-up from Power-down mode, if the IRC was used before entering Power-down mode, after IRC-start-up time (about 60  $\mu$ s), the 2-bit IRC timer starts counting and expiring in 4 cycles. Code execution can then be resumed immediately following the expiration of the IRC timer if the code was running from SRAM. In the meantime, the flash wake-up timer measures flash start-up time of about 100  $\mu$ s. When it times out, access to the flash is enabled. The user must remember to re-configure any required PLLs and clock dividers after the wake-up.

Wake-up from Power-down mode can be brought about by NMI, External Interrupts EINT0 through EINT3, GPIO interrupts, the Ethernet Wake-on-LAN interrupt, Brownout Detect, an RTC Alarm interrupt, a USB input pin transition (USB activity interrupt), or a CAN input pin transition, when the related interrupt is enabled.

#### 4.7.4 Deep Power-down mode

In Deep Power-down mode, power is shut off to the entire chip with the exception of the Real-Time Clock, the RESET pin, the WIC, and the RTC backup registers. Entry to Deep Power-down mode causes the DPDFLAG bit in PCON to be set, see [Table 35](#).

To optimize power conservation, the user has the additional option of turning off or retaining power to the 32 kHz oscillator. It is also possible to use external circuitry to turn off power to the on-chip regulator via the V<sub>DD(REG)(3V3)</sub> pins and/or the I/O power via the V<sub>DD(3V3)</sub> pins after entering Deep Power-down mode. Power must be restored before device operation can be restarted.

Wake-up from Deep Power-down mode will occur when an external reset signal is applied, or the RTC interrupt is enabled and an RTC interrupt is generated.

#### 4.7.5 Peripheral power control

A Power Control for Peripherals feature allows individual peripherals to be turned off if they are not needed in the application, resulting in additional power savings. This is detailed in the description of the PCONP register.

#### 4.7.6 Register description

The Power Control function uses registers shown in [Table 34](#). More detailed descriptions follow.

**Table 34. Power Control registers**

Name	Description	Access	Reset value <sup>[1]</sup>	Address
PCON	Power Control Register. This register contains control bits that enable some reduced power operating modes of the LPC178x/177x. See <a href="#">Table 35</a> .	R/W	0	0x400F C0C0
PCONP	Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, allowing elimination of power consumption by peripherals that are not needed. See <a href="#">Table 37</a> .	R/W	0x0408 829E	0x400F C0C4

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 4.7.7 Power Mode Control register (PCON - 0x400F C0C0)

Controls for some reduced power modes and other power related controls are contained in the PCON register, as described in [Table 35](#).

**Table 35. Power Mode Control register (PCON - address 0x400F C0C0) bit description**

Bit	Symbol	Description	Reset value
0	PM0	Power mode control bit 0. This bit controls entry to the Power-down mode. See <a href="#">Section 4.7.7.1</a> below for details.	0
1	PM1	Power mode control bit 1. This bit controls entry to the Deep Power-down mode. See <a href="#">Section 4.7.7.1</a> below for details.	0
2	BODRPM	Brown-Out Reduced Power Mode. When BODRPM is 1, the Brown-Out Detect circuitry will be turned off when chip Power-down mode or Deep Sleep mode is entered, resulting in a further reduction in power usage. However, the possibility of using Brown-Out Detect as a wake-up source from the reduced power mode will be lost. When 0, the Brown-Out Detect function remains active during Power-down and Deep Sleep modes. See the System Control Block chapter for details of Brown-Out detection.	0
3	BOGD	Brown-Out Global Disable. When BOGD is 1, the Brown-Out Detect circuitry is fully disabled at all times, and does not consume power. When 0, the Brown-Out Detect circuitry is enabled. See the System Control Block chapter for details of Brown-Out detection.	0
4	BORD	Brown-Out Reset Disable. When BORD is 1, the BOD will not reset the device when the $V_{DD(REG)(3V3)}$ voltage dips goes below the BOD reset trip level. The Brown-Out interrupt is not affected. When BORD is 0, the BOD reset is enabled. See the <a href="#">Section 3.6</a> for details of Brown-Out detection.	0
7:3	-	Reserved. Read value is undefined, only zero should be written.	NA
8	SMFLAG	Sleep Mode entry flag. Set when the Sleep mode is successfully entered. Cleared by software writing a one to this bit.	0 <a href="#">[1]</a> <a href="#">[2]</a>
9	DSFLAG	Deep Sleep entry flag. Set when the Deep Sleep mode is successfully entered. Cleared by software writing a one to this bit.	0 <a href="#">[1]</a> <a href="#">[2]</a>
10	PDFLAG	Power-down entry flag. Set when the Power-down mode is successfully entered. Cleared by software writing a one to this bit.	0 <a href="#">[1]</a> <a href="#">[2]</a>
11	DPDFLAG	Deep Power-down entry flag. Set when the Deep Power-down mode is successfully entered. Cleared by software writing a one to this bit.	0 <a href="#">[1]</a> <a href="#">[3]</a>
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Only one of these flags will be valid at a specific time.

[2] Hardware reset value only for a power-up of core power or by a brownout detect event.

[3] Hardware reset value only for a power-up event on Vbat.

#### 4.7.7.1 Encoding of Reduced Power Modes

The PM1 and PM0 bits in PCON allow entering reduced power modes as needed. The encoding of these bits allows backward compatibility with devices that previously only supported Sleep and Power-down modes. [Table 36](#) below shows the encoding for the three reduced power modes supported by the LPC178x/177x.



**Table 36. Encoding of reduced power modes**

PM1, PM0	Description
00	Execution of WFI or WFE enters either Sleep or Deep Sleep mode as defined by the SLEEPDEEP bit in the Cortex-M3 System Control Register.
01	Execution of WFI or WFE enters Power-down mode if the SLEEPDEEP bit in the Cortex-M3 System Control Register is 1.
10	Reserved, this setting should not be used.
11	Execution of WFI or WFE enters Deep Power-down mode if the SLEEPDEEP bit in the Cortex-M3 System Control Register is 1.

### 4.7.8 Wake-up from Reduced Power Modes

Any enabled interrupt can wake up the CPU from Sleep mode. Certain interrupts can wake up the processor if it is in either Deep Sleep mode or Power-down mode.

Interrupts that can occur during Deep Sleep or Power-down mode will wake up the CPU if the interrupt is enabled. After wake-up, execution will continue to the appropriate interrupt service routine. These interrupts are NMI, External Interrupts EINT0 through EINT3, GPIO interrupts, Ethernet Wake-on-LAN interrupt, Brownout Detect, RTC Alarm, CAN Activity Interrupt, and USB Activity Interrupt. For the wake-up process to take place the corresponding interrupt must be enabled in the NVIC. For pin-related peripheral functions, the related functions must also be mapped to pins.

The CAN Activity Interrupt is generated by activity on the CAN bus pins, and the USB Activity Interrupt is generated by activity on the USB bus pins. These interrupts are only useful to wake up the CPU when it is on Deep Sleep or Power-down mode, when the peripheral functions are powered up, but not active. Typically, if these interrupts are used, their flags should be polled just before enabling the interrupt and entering the desired reduced power mode. This can save time and power by avoiding an immediate wake-up. Upon wake-up, the interrupt service can turn off the related activity interrupt, do any application specific setup, and exit to await a normal peripheral interrupt.

In Deep Power-down mode, internal power to most of the device is removed, which limits the possibilities for waking up from this mode. On the LPC178x/177x, external reset can wake-up the device. Also, if the RTC is running and has been set up to cause an interrupt, that event can wake-up the device.

### 4.7.9 Power Control for Peripherals register (PCONP - 0x400F C0C4)

The PCONP register allows turning off selected peripheral functions for the purpose of saving power. This is accomplished by gating off the clock source to the specified peripheral blocks. A few peripheral functions cannot be turned off (i.e. the Watchdog timer and the System Control block).

Some peripherals, particularly those that include analog functions, may consume power that is not clock dependent. These peripherals may contain a separate disable control that turns off additional circuitry to reduce power. When this is the case, the peripheral should be disabled internally first, then turned off using PCONP, in order to get the greatest power savings. Information on peripheral specific power saving features may be found in the chapter describing that peripheral.

Each bit in PCONP controls one peripheral as shown in [Table 37](#).

If a peripheral control bit is 1, that peripheral is enabled. If a peripheral control bit is 0, that peripheral's clock is disabled (gated off) to conserve power. For example if bit 19 is 1, the I<sup>2</sup>C1 interface is enabled. If bit 19 is 0, the I<sup>2</sup>C1 interface is disabled.

**Important: valid read from a peripheral register and valid write to a peripheral register is possible only if that peripheral is enabled in the PCONP register!**

**Table 37. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description**

Bit	Symbol	Description	Reset value
0	PCLCD	LCD controller power/clock control bit.	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	0
6	PCPWM1	PWM1 power/clock control bit.	0
7	PCI2C0	I <sup>2</sup> C0 interface power/clock control bit.	1
8	PCUART4	UART4 power/clock control bit.	0
9	PCRTC	RTC power/clock control bit.	1
10	PCSSP1	SSP 1 interface power/clock control bit.	0
11	PCEMC	External Memory Controller power/clock control bit.	0
12	PCADC	A/D converter (ADC) power/clock control bit. <b>Note:</b> Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before attempting to set PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
15	PCGPIO	Power/clock control bit for IOCON, GPIO, and GPIO interrupts.	1
16	-	Reserved. Read value is undefined, only zero should be written.	NA
17	PCMCPWM	Motor Control PWM power/clock control bit.	0
18	PCQEI	Quadrature Encoder Interface power/clock control bit.	0
19	PCI2C1	I <sup>2</sup> C1 interface power/clock control bit.	1
20	PCSSP2	SSP2 interface power/clock control bit.	0
21	PCSSP0	SSP0 interface power/clock control bit.	0
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I <sup>2</sup> C interface 2 power/clock control bit.	1
27	PCI2S	I <sup>2</sup> S interface power/clock control bit.	0
28	PCSDC	SD Card interface power/clock control bit.	0
29	PCGPDMA	GPDMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

Note that the DAC peripheral does not have a control bit in PCONP. To enable the DAC, its output must be selected to appear on the related pin, P0[26], by configuring the relevant IOCON register. See [Section 8.4.1](#).

#### 4.7.10 Power control usage notes

After every reset, the PCONP register contains the value that enables selected interfaces and peripherals controlled by the PCONP to be enabled. Therefore, apart from proper configuring via peripheral dedicated registers, the user's application might have to access the PCONP in order to start using some of the on-board peripherals.

Power saving oriented systems should have 1s in the PCONP register only in positions that match peripherals really used in the application. All other bits, declared to be "Reserved" or dedicated to the peripherals not used in the current application, must be cleared to 0.

#### 4.7.11 Power domains

The LPC178x/177x provides two independent power domains that allow the bulk of the device to have power removed while maintaining operation of the Real Time Clock.

The VBAT pin supplies power only to the RTC domain. The RTC requires a minimum of power to operate, which can be supplied by an external battery. Whenever the device core power is present, that power is used to operate the RTC, causing no power drain from a battery when main power is available.

## 4.8 Wake-up timer

The LPC178x/177x begins operation at power-up and when awakened from Power-down mode by using the 12 MHz IRC oscillator as the clock source. This allows chip operation to begin quickly. If the main oscillator or one or both PLLs are needed by the application, software will need to enable these features and wait for them to stabilize before they are used as a clock source.

When the main oscillator is initially activated, the wake-up timer allows software to ensure that the main oscillator is fully functional before the processor uses it as a clock source and starts to execute instructions. This is important at power-on, all types of Reset, and whenever any of the aforementioned functions are turned off for any reason. Since the oscillator and other functions are turned off during Power-down mode, any wake-up of the processor from Power-down mode makes use of the Wake-up Timer.

The Wake-up Timer monitors the crystal oscillator as the means of checking whether it is safe to begin code execution. When power is applied to the chip, or some event caused the chip to exit Power-down mode, some time is required for the oscillator to produce a signal of sufficient amplitude to drive the clock logic. The amount of time depends on many factors, including the rate of  $V_{DD(REG)(3V3)}$  ramp (in the case of power on), the type of crystal and its electrical characteristics (if a quartz crystal is used), as well as any other external circuitry (e.g. capacitors), and the characteristics of the oscillator itself under the existing ambient conditions.

Once a clock is detected, the Wake-up Timer counts a fixed number of clocks (4,096), then sets the flag (OSCSTAT bit in the SCS register) that indicates that the main oscillator is ready for use. Software can then switch to the main oscillator and start any required PLLs. Refer to the Main Oscillator description in this chapter for details.

## 4.9 External clock output pin

For system test and development purposes, any one of several internal clocks may be brought out on the CLKOUT function available on the P1[25] or P1[27] pins, as shown in [Figure 10](#).

Clocks that may be observed via CLKOUT are the CPU clock (cclk), the main oscillator (osc\_clk), the internal RC oscillator (irc\_osc), the USB clock (usb\_clk), the RTC clock (rtc\_clk), and the Watchdog oscillator (wdt\_clk).

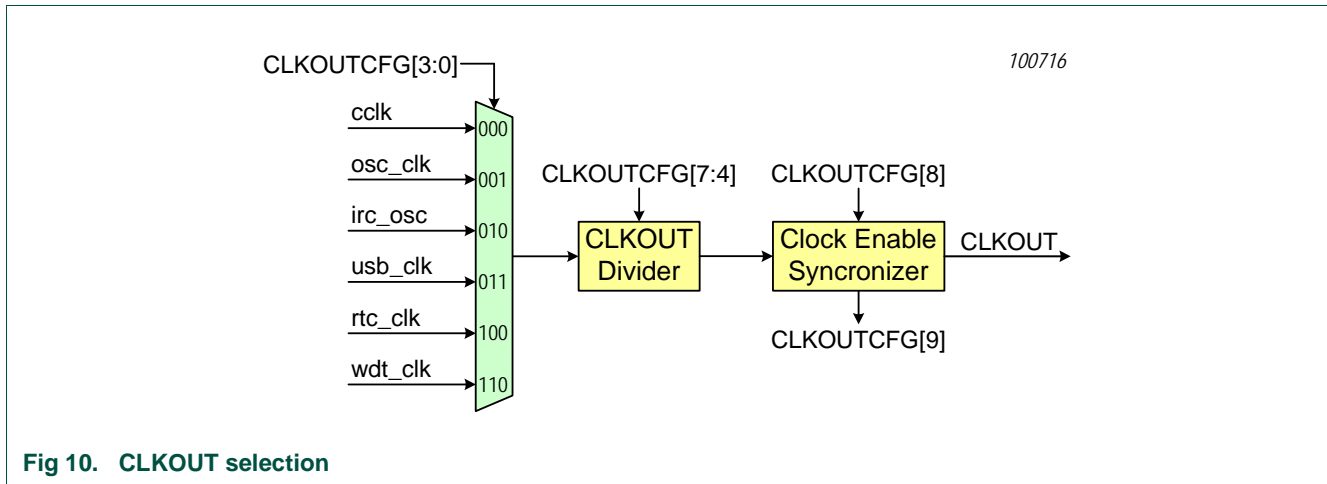


Fig 10. CLKOUT selection

### 4.9.1 Clock Output Configuration register (CLKOUTCFG - 0x400F C1C8)

The CLKOUTCFG register controls the selection of the internal clock that appears on the CLKOUT pin and allows dividing the clock by an integer value up to 16. The divider can be used to produce a system clock that is related to one of the on-chip clocks. For most clock sources, the division may be by 1. When the CPU clock is selected and is higher than approximately 50 MHz, the output must be divided in order to bring the frequency within the ability of the pin to switch with reasonable logic levels. If a clock is selected that is not running, there will be no signal on CLKOUT.

Note: The CLKOUT multiplexer is designed to switch cleanly, without glitches, between the possible clock sources. The divider is also designed to allow changing the divide value without glitches.

**Table 38. Clock Output Configuration register (CLKOUTCFG - 0x400F C1C8) bit description**

Bit	Symbol	Value	Description	Reset value
3:0	CLKOUTSEL		Selects the clock source for the CLKOUT function.	0
		0000	Selects the CPU clock as the CLKOUT source.	
		0001	Selects the main oscillator as the CLKOUT source.	
		0010	Selects the Internal RC oscillator as the CLKOUT source.	
		0011	Selects the USB clock as the CLKOUT source.	
		0100	Selects the RTC oscillator as the CLKOUT source.	
		0110	Selects the Watchdog oscillator as the CLKOUT source.	
		others	Reserved, do not use these settings.	
7:4	CLKOUTDIV		Integer value to divide the output clock by, minus one.	0
		0000	Clock is divided by 1.	
		0001	Clock is divided by 2.	
		0010	Clock is divided by 3.	
		1111	Clock is divided by 16.	
8	CLKOUT_EN		CLKOUT enable control, allows switching the CLKOUT source without glitches. Clear to stop CLKOUT on the next falling edge. Set to enable CLKOUT.	0
9	CLKOUT_ACT		CLKOUT activity indication. Reads as 1 when CLKOUT is enabled. Read as 0 when CLKOUT has been disabled via the CLKOUT_EN bit and the clock has completed being stopped.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 5.1 Introduction

The flash accelerator block in the LPC178x/177x allows maximization of the performance of the Cortex-M3 processor when it is running code from flash memory, while also saving power. The flash accelerator also provides speed and power improvements for data accesses to the flash memory.

### 5.2 Flash accelerator blocks

The flash accelerator is divided into several functional blocks:

- AHB-Lite bus interface, accessible by the Cortex-M3 I-code and D-code buses, as well as by the General Purpose DMA Controller
- An array of eight 128-bit buffers
- Flash accelerator control logic, including address compare and flash control
- A flash memory interface

Figure 11 shows a simplified diagram of the flash accelerator blocks and data paths.

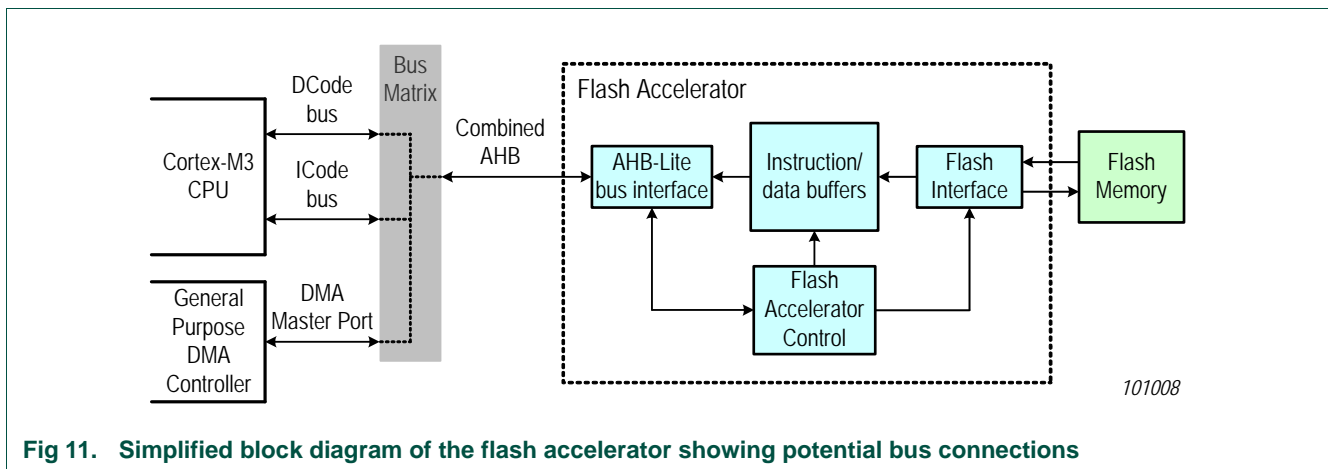


Fig 11. Simplified block diagram of the flash accelerator showing potential bus connections

In the following descriptions, the term “fetch” applies to an explicit flash read request from the CPU. “Prefetch” is used to denote a flash read of instructions beyond the current processor fetch address.

#### 5.2.1 Flash memory bank

There is one bank of flash memory controlled by the LPC178x/177x flash accelerator.

Flash programming operations are not controlled by the flash accelerator, but are handled as a separate function. A Boot ROM contains flash programming algorithms that may be called as part of the application program, and a loader that may be run to allow programming of the flash memory.

### 5.2.2 Flash programming Issues

Since the flash memory does not allow accesses during programming and erase operations, it is necessary for the flash accelerator to force the CPU to wait if a memory access to a flash address is requested while the flash memory is busy with a programming operation. Under some conditions, this delay could result in a Watchdog time-out. The user will need to be aware of this possibility and take steps to insure that an unwanted Watchdog reset does not cause a system failure while programming or erasing the flash memory.

In order to preclude the possibility of stale data being read from the flash memory, the LPC178x/177x flash accelerator buffers are automatically invalidated at the beginning of any flash programming or erase operation. Any subsequent read from a flash address will cause a new fetch to be initiated after the flash operation has completed.

## 5.3 Register description

The flash accelerator is controlled by the register shown in [Table 39](#). More detailed descriptions follow.

**Table 39. Summary of flash accelerator registers**

Name	Description	Access	Reset value <sup>[1]</sup>	Address
FLASHCFG	Flash Accelerator Configuration Register. Controls flash access timing. See <a href="#">Table 40</a> .	R/W	0x303A	0x400F C000

[1] Reset Value reflects the data stored in defined bits only. It does not include reserved bits content.



## 5.4 Flash Accelerator Configuration register (FLASHCFG - 0x400F C000)

Configuration bits select the flash access time, as shown in [Table 40](#). **The lower bits of FLASHCFG control internal flash accelerator functions and should not be altered.**

Following reset, flash accelerator functions are enabled and flash access timing is set to a default value of 4 clocks.

Changing the FLASHCFG register value causes the flash accelerator to invalidate all of the holding latches, resulting in new reads of flash information as required. This guarantees synchronization of the flash accelerator to CPU operation.

**Table 40. Flash Accelerator Configuration register (FLASHCFG - address 0x400F C000) bit description**

Bit	Symbol	Value	Description	Reset value
11:0	-	-	Reserved, user software should not change these bits from the reset value.	0x03A
15:12	FLASHTIM		Flash access time. The value of this field plus 1 gives the number of CPU clocks used for a flash access. <b>Warning:</b> improper setting of this value may result in incorrect operation of the device.	0x3
		0000	Flash accesses use 1 CPU clock. Use for up to 20 MHz CPU clock.	
		0001	Flash accesses use 2 CPU clocks. Use for up to 40 MHz CPU clock.	
		0010	Flash accesses use 3 CPU clocks. Use for up to 60 MHz CPU clock.	
		0011	Flash accesses use 4 CPU clocks. Use for up to 80 MHz CPU clock.	
		0100	Flash accesses use 5 CPU clocks. Use for up to 100 MHz CPU clock.	
		0101	Flash accesses use 6 CPU clocks. This "safe" setting will work under any conditions.	
		Other	Intended for potential future higher speed devices.	
31:16	-		Reserved. Read value is undefined, only zero should be written.	NA

## 5.5 Operation

Simply put, the flash accelerator attempts to have the next Cortex-M3 instruction that will be needed in its latches in time to prevent CPU fetch stalls. The LPC178x/177x uses one bank of flash memory. The flash accelerator includes an array of eight 128-bit buffers to store both instructions and data in a configurable manner. Each 128-bit buffer in the array can include four 32-bit instructions, eight 16-bit instructions or some combination of the two. During sequential code execution, a buffer typically contains the current instruction and the entire flash line that contains that instruction, or one flash line of data containing a previously requested address. Buffers are marked according to how they are used (as instruction or data buffers), and when they have been accessed. This information is used to carry out the buffer replacement strategy.

The Cortex-M3 provides a separate bus for instruction access (I-code) and data access (D-code) in the code memory space. These buses, plus the General Purpose DMA Controllers's master port, are arbitrated by the AHB multilayer matrix. Any access to the flash memory's address space is presented to the flash accelerator.

If a flash instruction fetch and a flash data access from the CPU occur at the same time, the multilayer matrix gives precedence to the data access. This is because a stalled data access always slows down execution, while a stalled instruction fetch often does not. When the flash data access is concluded, any flash fetch or prefetch that had been in progress is re-initiated.

Branches and other program flow changes cause a break in the sequential flow of instruction fetches described above. Buffer replacement strategy in the flash accelerator attempts to maximize the chances that potentially reusable information is retained until it is needed again.

If an attempt is made to write directly to the flash memory without using the normal flash programming interface (via Boot ROM function calls), the flash accelerator generates an error condition. The CPU treats this error as a data abort. The GPDMA handles error conditions as described in [Section 33.4.1.6.3](#).

When an Instruction Fetch is not satisfied by existing contents of the buffer array, nor has a prefetch been initiated for that flash line, the CPU will be stalled while a fetch is initiated for the related 128-bit flash line. If a prefetch has been initiated but not yet completed, the CPU is stalled for a shorter time since the required flash access is already in progress.

Typically, a flash prefetch is begun whenever an access is made to a just prefetched address, or to a buffer whose immediate successor is not already in another buffer. A prefetch in progress may be aborted by a data access, in order to minimize CPU stalls.

A prefetched flash line is latched within the flash memory, but the flash accelerator does not capture the line in a buffer until the CPU presents an address that is contained within the prefetched flash line. If the core presents an instruction address that is not already buffered and is not contained in the prefetched flash line, the prefetched line will be discarded.

Some special cases include the possibility that the CPU will request a data access to an address already contained in an instruction buffer. In this case, the data will be read from the buffer as if it was a data buffer. The reverse case, if the CPU requests an instruction address that can be satisfied from an existing data buffer, causes the instruction to be supplied from the data buffer, and the buffer to be changed into an instruction buffer. This causes the buffer to be handled differently when the flash accelerator is determining which buffer is to be overwritten next.

### 6.1 Features

---

- Nested Vectored Interrupt Controller that is an integral part of the ARM Cortex-M3
- Tightly coupled interrupt controller provides low interrupt latency
- Controls system exceptions and peripheral interrupts
- In the LPC178x/177x, the NVIC supports 40 vectored interrupts
- 32 programmable interrupt priority levels, with hardware priority level masking
- Relocatable vector table
- Non-Maskable Interrupt
- Software interrupt generation

### 6.2 Description

---

The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M3. The tight coupling to the CPU allows for low interrupt latency and efficient processing of late arriving interrupts. The NVIC handles interrupts in addition to system exceptions. Exceptions include Reset, NMI, Hard Fault, MemManage Fault, Bus Fault, Usage Fault, SVCcall, Debug Monitor, PendSV, and SysTick.

Refer to the Cortex-M3 User Guide [Section 38.4.2](#) for details of NVIC operation.

### 6.3 Interrupt sources

---

[Table 41](#) lists the interrupt sources for each peripheral function. Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller. Each line may represent more than one interrupt source, as noted.

Exception numbers relate to where entries are stored in the exception vector table. Interrupt numbers are used in some other contexts, such as software interrupts.

Note that system exceptions are hard-wired into the Cortex-M3 and are not shown in the table. Some other information about the SysTick interrupt can be found in the System Tick Timer chapter, [Section 25.1](#)

In addition, the NVIC handles the Non-Maskable Interrupt (NMI). In order for NMI to operate from an external signal, the NMI function must be connected to the related device pin (P2[10] / EINT0n / NMI). When connected, a logic 1 on the pin will cause the NMI to be processed. For details, refer to the Cortex-M3 User Guide that is an appendix to this User Manual.

**Table 41. Connection of interrupt sources to the Vectored Interrupt Controller**

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
0	16	0x40	WDT	Watchdog Interrupt (WDINT)
1	17	0x44	Timer 0	Match 0 - 1 (MR0, MR1) Capture 0 - 1 (CR0, CR1)
2	18	0x48	Timer 1	Match 0 - 2 (MR0, MR1, MR2) Capture 0 - 1 (CR0, CR1)
3	19	0x4C	Timer 2	Match 0-3 Capture 0-1
4	20	0x50	Timer 3	Match 0-3 Capture 0-1
5	21	0x54	UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
6	22	0x58	UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Control Change End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
7	23	0x5C	UART 2	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
8	24	0x60	UART 3	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
9	25	0x64	PWM1	Match 0 - 6 of PWM1 Capture 0-1 of PWM1
10	26	0x68	I <sup>2</sup> C0	SI (state change)
11	27	0x6C	I <sup>2</sup> C1	SI (state change)
12	28	0x70	I <sup>2</sup> C2	SI (state change)
13	29	0x74	(unused)	-

Table 41. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
14	30	0x78	SSP0	Tx FIFO half empty of SSP0 Rx FIFO half full of SSP0 Rx Timeout of SSP0 Rx Overrun of SSP0
15	31	0x7C	SSP 1	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
16	32	0x80	PLL0 (Main PLL)	PLL0 Lock (PLOCK0)
17	33	0x84	RTC	Counter Increment (RTCCIF), Alarm (RTCALF)
18	34	0x88	External Interrupt	External Interrupt 0 (EINT0)
19	35	0x8C	External Interrupt	External Interrupt 1 (EINT1)
20	36	0x90	External Interrupt	External Interrupt 2 (EINT2)
21	37	0x94	External Interrupt	External Interrupt 3 (EINT3)
22	38	0x98	ADC	A/D Converter end of conversion
23	39	0x9C	BOD	Brown Out detect
24	40	0xA0	USB	USB_INT_REQ_LP, USB_INT_REQ_HP, USB_INT_REQ_DMA, USB_HOST_INT, USB_ATX_INT, USB_OTG_INT, USB_I2C_INT
25	41	0xA4	CAN	CAN Common, CAN 0 Tx, CAN 0 Rx, CAN 1 Tx, CAN 1 Rx
26	42	0xA8	DMA Controller	Interrupt status of all DMA channels
27	43	0xAC	I <sup>2</sup> S	irq, dmareq1, dmareq2
28	44	0xB0	Ethernet	WakeupInt, SoftInt, TxDoneInt, TxFinishedInt, TxErrorInt, TxUnderrunInt, RxDoneInt, RxFinishedInt, RxErrorInt, RxOverrunInt.
29	45	0xB4	SD Card Interface	RxDATAAvlBl, TxDATAAvlBl, RxFifoEmpty, TxFifoEmpty, RxFifoFull, TxFifoFull, RxFifoHalfFull, TxFifoHalfEmpty, RxActive, TxActive, CmdActive, DataBlockEnd, StartBitErr, DataEnd, CmdSent, CmdRespEnd, RxOverrun, TxUnderrun, DataTimeOut, CmdTimeOut, DataCrcFail, CmdCrcFail
30	46	0xB8	Motor Control PWM	IPER[2:0], IPW[2:0], ICAP[2:0], FES
31	47	0xBC	Quadrature Encoder	INX_Int, TIM_Int, VELC_Int, DIR_Int, ERR_Int, ENCLK_Int, POS0_Int, POS1_Int, POS2_Int, REV_Int, POS0REV_Int, POS1REV_Int, POS2REV_Int
32	48	0xC0	PLL1 (Alt PLL)	PLL1 Lock (PLOCK1)
33	49	0xC4	USB Activity Interrupt	USB_NEED_CLK
34	50	0xC8	CAN Activity Interrupt	CAN1WAKE, CAN2WAKE
35	51	0xCC	UART4	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)

Table 41. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
36	52	0xD0	SSP2	Tx FIFO half empty of SSP2 Rx FIFO half full of SSP2 Rx Timeout of SSP2 Rx Overrun of SSP2
37	53	0xD4	LCD controller	BER, VCompl, LNBUI, FUF1, Crsr1
38	54	0xD8	GPIO interrupts	P0xREI, P2xREI, P0xFEI, P2xFEI
39	55	0xDC	PWM0	Match 0 - 6 of PWM0 Capture 0-1 of PWM0
40	56	0xE0	EEPROM	EE_PROG_DONE, EE_RW_DONE

## 6.4 Vector table remapping

The Cortex-M3 incorporates a mechanism that allows remapping the interrupt vector table to alternate locations in the memory map. This is controlled via the Vector Table Offset Register (VTOR) contained in the Cortex-M3.

The vector table may be located anywhere within the bottom 1 GB of Cortex-M3 address space. The vector table should be located on a 256 word (1024 byte) boundary to insure alignment on LPC178x/177x family devices. Refer to [Section 38.4.3.5](#) of the Cortex-M3 User Guide appended to this manual for details of the Vector Table Offset feature.

ARM describes bit 29 of the VTOR (TBLOFF) as selecting a memory region, either code or SRAM. For simplicity, this bit can be thought as simply part of the address offset since the split between the “code” space and the “SRAM” space occurs at the location corresponding to bit 29 in a memory address.

### Examples:

To place the vector table at the beginning of the “local” static RAM, starting at address 0x1000 0000, place the value 0x1000 0000 in the VTOR register. This indicates address 0x1000 0000 in the code space, since bit 29 of the VTOR equals 0.

To place the vector table at the beginning of the peripheral RAM, starting at address 0x2000 0000, place the value 0x2000 0000 in the VTOR register. This indicates address 0x2000 0000 in the SRAM space, since bit 29 of the VTOR equals 1.

## 6.5 Register description

The following table summarizes the registers in the NVIC as implemented in the LPC178x/177x. The Cortex-M3 User Guide [Section 38.4.2](#) provides a functional description of the NVIC.

**Table 42. NVIC register map**

Name	Description	Access	Reset value	Address	Table
ISER0 to ISER1	Interrupt Set-Enable Registers. These registers allow enabling interrupts and reading back the interrupt enables for specific peripheral functions.	RW	0	ISER0 - 0xE000 E100 ISER1 - 0xE000 E104	<a href="#">43</a> <a href="#">44</a>
ICER0 to ICER1	Interrupt Clear-Enable Registers. These registers allow disabling interrupts and reading back the interrupt enables for specific peripheral functions.	RW	0	ICER0 - 0xE000 E180 ICER1 - 0xE000 E184	<a href="#">45</a> <a href="#">46</a>
ISPR0 to ISPR1	Interrupt Set-Pending Registers. These registers allow changing the interrupt state to pending and reading back the interrupt pending state for specific peripheral functions.	RW	0	ISPR0 - 0xE000 E200 ISPR1 - 0xE000 E204	<a href="#">47</a> <a href="#">48</a>
ICPR0 to ICPR1	Interrupt Clear-Pending Registers. These registers allow changing the interrupt state to not pending and reading back the interrupt pending state for specific peripheral functions.	RW	0	ICPR0 - 0xE000 E280 ICPR1 - 0xE000 E284	<a href="#">49</a> <a href="#">50</a>
IABR0 to IABR1	Interrupt Active Bit Registers. These registers allow reading the current interrupt active state for specific peripheral functions.	RO	0	IABR0 - 0xE000 E300 IABR1 - 0xE000 E304	<a href="#">51</a> <a href="#">52</a>
IPR0 to IPR10	Interrupt Priority Registers. These registers allow assigning a priority to each interrupt. Each register contains the 5-bit priority fields for 4 interrupts.	RW	0	IPR0 - 0xE000 E400 IPR1 - 0xE000 E404 IPR2 - 0xE000 E408 IPR3 - 0xE000 E40C IPR4 - 0xE000 E410 IPR5 - 0xE000 E414 IPR6 - 0xE000 E418 IPR7 - 0xE000 E41C IPR8 - 0xE000 E420 IPR9 - 0xE000 E424 IPR10 - 0xE000 E428	<a href="#">53</a> <a href="#">54</a> <a href="#">55</a> <a href="#">56</a> <a href="#">57</a> <a href="#">58</a> <a href="#">59</a> <a href="#">60</a> <a href="#">61</a> <a href="#">62</a> <a href="#">63</a>
STIR	Software Trigger Interrupt Register. This register allows software to generate an interrupt.	WO	0	STIR - 0xE000 EF00	<a href="#">64</a>

### 6.5.1 Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)

The ISER0 register allows enabling the first 32 peripheral interrupts, or for reading the enabled state of those interrupts. The remaining interrupts are enabled via the ISER1 register ([Section 6.5.2](#)). Disabling interrupts is done through the ICER0 and ICER1 registers ([Section 6.5.3](#) and [Section 5.4](#)).

**Table 43. Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)**

Bit	Name	Function
0	ISE_WDT	Watchdog Timer interrupt enable. Write: writing 0 has no effect, writing 1 enables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ISE_TIMER0	Timer 0 interrupt enable. See functional description for bit 0.
2	ISE_TIMER1	Timer 1 interrupt enable. See functional description for bit 0.
3	ISE_TIMER2	Timer 2 interrupt enable. See functional description for bit 0.
4	ISE_TIMER3	Timer 3 interrupt enable. See functional description for bit 0.
5	ISE_UART0	UART0 interrupt enable. See functional description for bit 0.
6	ISE_UART1	UART1 interrupt enable. See functional description for bit 0.
7	ISE_UART2	UART2 interrupt enable. See functional description for bit 0.
8	ISE_UART3	UART3 interrupt enable. See functional description for bit 0.
9	ISE_PWM1	PWM1 interrupt enable. See functional description for bit 0.
10	ISE_I2C0	I <sup>2</sup> C0 interrupt enable. See functional description for bit 0.
11	ISE_I2C1	I <sup>2</sup> C1 interrupt enable. See functional description for bit 0.
12	ISE_I2C2	I <sup>2</sup> C2 interrupt enable. See functional description for bit 0.
13	-	Reserved. Read value is undefined, only zero should be written.
14	ISE_SSP0	SSP0 interrupt enable. See functional description for bit 0.
15	ISE_SSP1	SSP1 interrupt enable. See functional description for bit 0.
16	ISE_PLL0	PLL0 (Main PLL) interrupt enable. See functional description for bit 0.
17	ISE_RTC	Real Time Clock (RTC) interrupt enable. See functional description for bit 0.
18	ISE_EINT0	External Interrupt 0 interrupt enable. See functional description for bit 0.
19	ISE_EINT1	External Interrupt 1 interrupt enable. See functional description for bit 0.
20	ISE_EINT2	External Interrupt 2 interrupt enable. See functional description for bit 0.
21	ISE_EINT3	External Interrupt 3 interrupt enable. See functional description for bit 0.
22	ISE_ADC	ADC interrupt enable. See functional description for bit 0.
23	ISE_BOD	BOD interrupt enable. See functional description for bit 0.
24	ISE_USB	USB interrupt enable. See functional description for bit 0.
25	ISE_CAN	CAN interrupt enable. See functional description for bit 0.
26	ISE_DMA	GPDMA interrupt enable. See functional description for bit 0.
27	ISE_I2S	I <sup>2</sup> S interrupt enable. See functional description for bit 0.
28	ISE_ENET	Ethernet interrupt enable. See functional description for bit 0.
29	ISE_SD	SD card interface interrupt enable. See functional description for bit 0.
30	ISE_MCPWM	Motor Control PWM interrupt enable. See functional description for bit 0.
31	ISE_QEI	Quadrature Encoder Interface interrupt enable. See functional description for bit 0.



### 6.5.2 Interrupt Set-Enable Register 1 register (ISER1 - 0xE000 E104)

The ISER1 register allows enabling the second group of peripheral interrupts, or for reading the enabled state of those interrupts. Disabling interrupts is done through the ICER0 and ICER1 registers ([Section 6.5.3](#) and [Section 5.4](#)).

**Table 44. Interrupt Set-Enable Register 1 register (ISER1 - 0xE000 E104)**

Bit	Name	Function
0	ISE_PLL1	PLL1 (Alt PLL) interrupt enable. Write: writing 0 has no effect, writing 1 enables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ISE_USBACT	USB Activity interrupt enable. See functional description for bit 0.
2	ISE_CANACT	CAN Activity interrupt enable. See functional description for bit 0.
3	ISE_UART4	UART4 interrupt enable. See functional description for bit 0.
4	ISE_SSP2	SSP2 interrupt enable. See functional description for bit 0.
5	ISE_LCD	LCD interrupt enable. See functional description for bit 0.
6	ISE_GPIO	GPIO interrupt enable. See functional description for bit 0.
7	ISE_PWM0	PWM0 interrupt enable. See functional description for bit 0.
8	ISE_FLASH	Flash and EEPROM interrupt enable. See functional description for bit 0.
31:9	-	Reserved. Read value is undefined, only zero should be written.

### 6.5.3 Interrupt Clear-Enable Register 0 (ICER0 - 0xE000 E180)

The ICER0 register allows disabling the first 32 peripheral interrupts, or for reading the enabled state of those interrupts. The remaining interrupts are disabled via the ICER1 register ([Section 5.4](#)). Enabling interrupts is done through the ISER0 and ISER1 registers ([Section 6.5.1](#) and [Section 6.5.2](#)).

**Table 45. Interrupt Clear-Enable Register 0 (ICER0 - 0xE000 E180)**

Bit	Name	Function
0	ICE_WDT	Watchdog Timer interrupt disable. Write: writing 0 has no effect, writing 1 disables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ICE_TIMER0	Timer 0 interrupt disable. See functional description for bit 0.
2	ICE_TIMER1	Timer 1 interrupt disable. See functional description for bit 0.
3	ICE_TIMER2	Timer 2 interrupt disable. See functional description for bit 0.
4	ICE_TIMER3	Timer 3 interrupt disable. See functional description for bit 0.
5	ICE_UART0	UART0 interrupt disable. See functional description for bit 0.
6	ICE_UART1	UART1 interrupt disable. See functional description for bit 0.
7	ICE_UART2	UART2 interrupt disable. See functional description for bit 0.
8	ICE_UART3	UART3 interrupt disable. See functional description for bit 0.
9	ICE_PWM1	PWM1 interrupt disable. See functional description for bit 0.
10	ICE_I2C0	I <sup>2</sup> C0 interrupt disable. See functional description for bit 0.
11	ICE_I2C1	I <sup>2</sup> C1 interrupt disable. See functional description for bit 0.
12	ICE_I2C2	I <sup>2</sup> C2 interrupt disable. See functional description for bit 0.
13	-	Reserved. Read value is undefined, only zero should be written.
14	ICE_SSP0	SSP0 interrupt disable. See functional description for bit 0.
15	ICE_SSP1	SSP1 interrupt disable. See functional description for bit 0.
16	ICE_PLL0	PLL0 (Main PLL) interrupt disable. See functional description for bit 0.
17	ICE_RTC	Real Time Clock (RTC) interrupt disable. See functional description for bit 0.
18	ICE_EINT0	External Interrupt 0 interrupt disable. See functional description for bit 0.
19	ICE_EINT1	External Interrupt 1 interrupt disable. See functional description for bit 0.
20	ICE_EINT2	External Interrupt 2 interrupt disable. See functional description for bit 0.
21	ICE_EINT3	External Interrupt 3 interrupt disable. See functional description for bit 0.
22	ICE_ADC	ADC interrupt disable. See functional description for bit 0.
23	ICE_BOD	BOD interrupt disable. See functional description for bit 0.
24	ICE_USB	USB interrupt disable. See functional description for bit 0.
25	ICE_CAN	CAN interrupt disable. See functional description for bit 0.
26	ICE_DMA	GPDMA interrupt disable. See functional description for bit 0.
27	ICE_I2S	I <sup>2</sup> S interrupt disable. See functional description for bit 0.
28	ICE_ENET	Ethernet interrupt disable. See functional description for bit 0.
29	ICE_SD	SD card interface interrupt disable. See functional description for bit 0.
30	ICE_MCPWM	Motor Control PWM interrupt disable. See functional description for bit 0.
31	ICE_QEI	Quadrature Encoder Interface interrupt disable. See functional description for bit 0.

#### 5.4 Interrupt Clear-Enable Register 1 register (ICER1 - 0xE000 E184)

The ICER1 register allows disabling the second group of peripheral interrupts, or for reading the enabled state of those interrupts. Enabling interrupts is done through the ISER0 and ISER1 registers ([Section 6.5.1](#) and [Section 6.5.2](#)).

**Table 46. Interrupt Clear-Enable Register 1 register (ICER1 - 0xE000 E184)**

Bit	Name	Function
0	ICE_PLL1	PLL1 (Alt PLL) interrupt disable. Write: writing 0 has no effect, writing 1 disables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ICE_USBACT	USB Activity interrupt disable. See functional description for bit 0.
2	ICE_CANACT	CAN Activity interrupt disable. See functional description for bit 0.
3	ICE_UART4	UART4 interrupt disable. See functional description for bit 0.
4	ICE_SSP2	SSP2 interrupt disable. See functional description for bit 0.
5	ICE_LCD	LCD interrupt disable. See functional description for bit 0.
6	ICE_GPIO	GPIO interrupt disable. See functional description for bit 0.
7	ICE_PWM0	PWM0 interrupt disable. See functional description for bit 0.
8	ICE_EEPROM	EEPROM interrupt disable. See functional description for bit 0.
31:9	-	Reserved. Read value is undefined, only zero should be written.

### 6.5.5 Interrupt Set-Pending Register 0 register (ISPR0 - 0xE000 E200)

The ISPR0 register allows setting the pending state of the first 32 peripheral interrupts, or for reading the pending state of those interrupts. The remaining interrupts can have their pending state set via the ISPR1 register ([Section 6.5.6](#)). Clearing the pending state of interrupts is done through the ICPR0 and ICPR1 registers ([Section 6.5.7](#) and [Section 6.5.8](#)).

**Table 47. Interrupt Set-Pending Register 0 register (ISPR0 - 0xE000 E200)**

Bit	Name	Function
0	ISP_WDT	Watchdog Timer interrupt pending set. Write: writing 0 has no effect, writing 1 changes the interrupt state to pending. Read: 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.
1	ISP_TIMER0	Timer 0 interrupt pending set. See functional description for bit 0.
2	ISP_TIMER1	Timer 1 interrupt pending set. See functional description for bit 0.
3	ISP_TIMER2	Timer 2 interrupt pending set. See functional description for bit 0.
4	ISP_TIMER3	Timer 3 interrupt pending set. See functional description for bit 0.
5	ISP_UART0	UART0 interrupt pending set. See functional description for bit 0.
6	ISP_UART1	UART1 interrupt pending set. See functional description for bit 0.
7	ISP_UART2	UART2 interrupt pending set. See functional description for bit 0.
8	ISP_UART3	UART3 interrupt pending set. See functional description for bit 0.
9	ISP_PWM1	PWM1 interrupt pending set. See functional description for bit 0.
10	ISP_I2C0	I <sup>2</sup> C0 interrupt pending set. See functional description for bit 0.
11	ISP_I2C1	I <sup>2</sup> C1 interrupt pending set. See functional description for bit 0.
12	ISP_I2C2	I <sup>2</sup> C2 interrupt pending set. See functional description for bit 0.
13	-	Reserved. Read value is undefined, only zero should be written.
14	ISP_SSP0	SSP0 interrupt pending set. See functional description for bit 0.
15	ISP_SSP1	SSP1 interrupt pending set. See functional description for bit 0.
16	ISP_PLL0	PLL0 (Main PLL) interrupt pending set. See functional description for bit 0.
17	ISP_RTC	Real Time Clock (RTC) interrupt pending set. See functional description for bit 0.
18	ISP_EINT0	External Interrupt 0 interrupt pending set. See functional description for bit 0.
19	ISP_EINT1	External Interrupt 1 interrupt pending set. See functional description for bit 0.
20	ISP_EINT2	External Interrupt 2 interrupt pending set. See functional description for bit 0.
21	ISP_EINT3	External Interrupt 3 interrupt pending set. See functional description for bit 0.
22	ISP_ADC	ADC interrupt pending set. See functional description for bit 0.
23	ISP_BOD	BOD interrupt pending set. See functional description for bit 0.
24	ISP_USB	USB interrupt pending set. See functional description for bit 0.
25	ISP_CAN	CAN interrupt pending set. See functional description for bit 0.
26	ISP_DMA	GPDMA interrupt pending set. See functional description for bit 0.
27	ISP_I2S	I <sup>2</sup> S interrupt pending set. See functional description for bit 0.
28	ISP_ENET	Ethernet interrupt pending set. See functional description for bit 0.
29	ISP_SD	SD card interface interrupt pending set. See functional description for bit 0.
30	ISP_MCPWM	Motor Control PWM interrupt pending set. See functional description for bit 0.
31	ISP_QEI	Quadrature Encoder Interface interrupt pending set. See functional description for bit 0.

### 6.5.6 Interrupt Set-Pending Register 1 register (ISPR1 - 0xE000 E204)

The ISPR1 register allows setting the pending state of the second group of peripheral interrupts, or for reading the pending state of those interrupts. Clearing the pending state of interrupts is done through the ICPR0 and ICPR1 registers ([Section 6.5.7](#) and [Section 6.5.8](#)).

**Table 48. Interrupt Set-Pending Register 1 register (ISPR1 - 0xE000 E204)**

Bit	Name	Function
0	ISP_PLL1	PLL1 (Alt PLL) interrupt pending set. Write: writing 0 has no effect, writing 1 changes the interrupt state to pending. Read: 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.
1	ISP_USBACT	USB Activity interrupt pending set. See functional description for bit 0.
2	ISP_CANACT	CAN Activity interrupt pending set. See functional description for bit 0.
3	ISP_UART4	UART4 interrupt pending set. See functional description for bit 0.
4	ISP_SSP2	SSP2 interrupt pending set. See functional description for bit 0.
5	ISP_LCD	LCD interrupt pending set. See functional description for bit 0.
6	ISP_GPIO	GPIO interrupt pending set. See functional description for bit 0.
7	ISP_PWM0	PWM0 interrupt pending set. See functional description for bit 0.
8	ISP_EEPROM	EEPROM interrupt pending set. See functional description for bit 0.
31:9	-	Reserved. Read value is undefined, only zero should be written.

### 6.5.7 Interrupt Clear-Pending Register 0 register (ICPR0 - 0xE000 E280)

The ICPR0 register allows clearing the pending state of the first 32 peripheral interrupts, or for reading the pending state of those interrupts. The remaining interrupts can have their pending state cleared via the ICPR1 register ([Section 6.5.8](#)). Setting the pending state of interrupts is done through the ISPR0 and ISPR1 registers ([Section 6.5.5](#) and [Section 6.5.6](#)).

**Table 49. Interrupt Clear-Pending Register 0 register (ICPR0 - 0xE000 E280)**

Bit	Name	Function
0	ICP_WDT	Watchdog Timer interrupt pending clear. Write: writing 0 has no effect, writing 1 changes the interrupt state to not pending. Read: 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.
1	ICP_TIMER0	Timer 0 interrupt pending clear. See functional description for bit 0.
2	ICP_TIMER1	Timer 1 interrupt pending clear. See functional description for bit 0.
3	ICP_TIMER2	Timer 2 interrupt pending clear. See functional description for bit 0.
4	ICP_TIMER3	Timer 3 interrupt pending clear. See functional description for bit 0.
5	ICP_UART0	UART0 interrupt pending clear. See functional description for bit 0.
6	ICP_UART1	UART1 interrupt pending clear. See functional description for bit 0.
7	ICP_UART2	UART2 interrupt pending clear. See functional description for bit 0.
8	ICP_UART3	UART3 interrupt pending clear. See functional description for bit 0.
9	ICP_PWM1	PWM1 interrupt pending clear. See functional description for bit 0.
10	ICP_I2C0	I <sup>2</sup> C0 interrupt pending clear. See functional description for bit 0.
11	ICP_I2C1	I <sup>2</sup> C1 interrupt pending clear. See functional description for bit 0.
12	ICP_I2C2	I <sup>2</sup> C2 interrupt pending clear. See functional description for bit 0.
13	-	Reserved. Read value is undefined, only zero should be written.
14	ICP_SSP0	SSP0 interrupt pending clear. See functional description for bit 0.
15	ICP_SSP1	SSP1 interrupt pending clear. See functional description for bit 0.
16	ICP_PLL0	PLL0 (Main PLL) interrupt pending clear. See functional description for bit 0.
17	ICP_RTC	Real Time Clock (RTC) interrupt pending clear. See functional description for bit 0.
18	ICP_EINT0	External Interrupt 0 interrupt pending clear. See functional description for bit 0.
19	ICP_EINT1	External Interrupt 1 interrupt pending clear. See functional description for bit 0.
20	ICP_EINT2	External Interrupt 2 interrupt pending clear. See functional description for bit 0.
21	ICP_EINT3	External Interrupt 3 interrupt pending clear. See functional description for bit 0.
22	ICP_ADC	ADC interrupt pending clear. See functional description for bit 0.
23	ICP_BOD	BOD interrupt pending clear. See functional description for bit 0.
24	ICP_USB	USB interrupt pending clear. See functional description for bit 0.
25	ICP_CAN	CAN interrupt pending clear. See functional description for bit 0.
26	ICP_DMA	GPDMA interrupt pending clear. See functional description for bit 0.
27	ICP_I2S	I <sup>2</sup> S interrupt pending clear. See functional description for bit 0.
28	ICP_ENET	Ethernet interrupt pending clear. See functional description for bit 0.
29	ICP_SD	SD Card interface interrupt pending clear. See functional description for bit 0.
30	ICP_MCPWM	Motor Control PWM interrupt pending clear. See functional description for bit 0.
31	ICP_QEI	Quadrature Encoder Interface interrupt pending clear. See functional description for bit 0.

### 6.5.8 Interrupt Clear-Pending Register 1 register (ICPR1 - 0xE000 E284)

The ICPR1 register allows clearing the pending state of the second group of peripheral interrupts, or for reading the pending state of those interrupts. Setting the pending state of interrupts is done through the ISPR0 and ISPR1 registers ([Section 6.5.5](#) and [Section 6.5.6](#)).

**Table 50. Interrupt Clear-Pending Register 1 register (ICPR1 - 0xE000 E284)**

Bit	Name	Function
0	ICP_PLL1	PLL1 (Alt PLL) interrupt pending clear. Write: writing 0 has no effect, writing 1 changes the interrupt state to not pending. Read: 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.
1	ICP_USBACT	USB Activity interrupt pending clear. See functional description for bit 0.
2	ICP_CANACT	CAN Activity interrupt pending clear. See functional description for bit 0.
3	ICP_UART4	UART4 interrupt pending clear. See functional description for bit 0.
4	ICP_SSP2	SSP2 interrupt pending clear. See functional description for bit 0.
5	ICP_LCD	LCD interrupt pending clear. See functional description for bit 0.
6	ICP_GPIO	GPIO interrupt pending clear. See functional description for bit 0.
7	ICP_PWM0	PWM0 interrupt pending clear. See functional description for bit 0.
8	ICP_EEPROM	EEPROM interrupt pending clear. See functional description for bit 0.
31:9	-	Reserved. Read value is undefined, only zero should be written.

### 6.5.9 Interrupt Active Bit Register 0 (IABR0 - 0xE000 E300)

The IABR0 register is a read-only register that allows reading the active state of the first 32 peripheral interrupts. Bits in IABR are set while the corresponding interrupt service routines are in progress. Additional interrupts can have their active state read via the IABR1 register ([Section 6.5.10](#)).

**Table 51. Interrupt Active Bit Register 0 (IABR0 - 0xE000 E300)**

Bit	Name	Function
0	IAB_WDT	Watchdog Timer interrupt active. Read: 0 indicates that the interrupt is not active, 1 indicates that the interrupt is active.
1	IAB_TIMER0	Timer 0 interrupt active. See functional description for bit 0.
2	IAB_TIMER1	Timer 1 interrupt active. See functional description for bit 0.
3	IAB_TIMER2	Timer 2 interrupt active. See functional description for bit 0.
4	IAB_TIMER3	Timer 3 interrupt active. See functional description for bit 0.
5	IAB_UART0	UART0 interrupt active. See functional description for bit 0.
6	IAB_UART1	UART1 interrupt active. See functional description for bit 0.
7	IAB_UART2	UART2 interrupt active. See functional description for bit 0.
8	IAB_UART3	UART3 interrupt active. See functional description for bit 0.
9	IAB_PWM1	PWM1 interrupt active. See functional description for bit 0.
10	IAB_I2C0	I <sup>2</sup> C0 interrupt active. See functional description for bit 0.
11	IAB_I2C1	I <sup>2</sup> C1 interrupt active. See functional description for bit 0.
12	IAB_I2C2	I <sup>2</sup> C2 interrupt active. See functional description for bit 0.
13	-	Reserved. Read value is undefined, only zero should be written.
14	IAB_SSP0	SSP0 interrupt active. See functional description for bit 0.
15	IAB_SSP1	SSP1 interrupt active. See functional description for bit 0.
16	IAB_PLL0	PLL0 (Main PLL) interrupt active. See functional description for bit 0.
17	IAB_RTC	Real Time Clock (RTC) interrupt active. See functional description for bit 0.
18	IAB_EINT0	External Interrupt 0 interrupt active. See functional description for bit 0.
19	IAB_EINT1	External Interrupt 1 interrupt active. See functional description for bit 0.
20	IAB_EINT2	External Interrupt 2 interrupt active. See functional description for bit 0.
21	IAB_EINT3	External Interrupt 3 interrupt active. See functional description for bit 0.
22	IAB_ADC	ADC interrupt active. See functional description for bit 0.
23	IAB_BOD	BOD interrupt active. See functional description for bit 0.
24	IAB_USB	USB interrupt active. See functional description for bit 0.
25	IAB_CAN	CAN interrupt active. See functional description for bit 0.
26	IAB_DMA	GPDMA interrupt active. See functional description for bit 0.
27	IAB_I2S	I <sup>2</sup> S interrupt active. See functional description for bit 0.
28	IAB_ENET	Ethernet interrupt active. See functional description for bit 0.
29	IAB_SD	Repetitive Interrupt Timer interrupt active. See functional description for bit 0.
30	IAB_MCPWM	Motor Control PWM interrupt active. See functional description for bit 0.
31	IAB_QEI	Quadrature Encoder Interface interrupt active. See functional description for bit 0.



**6.5.10 Interrupt Active Bit Register 1 (IABR1 - 0xE000 E304)**

The IABR1 register is a read-only register that allows reading the active state of the second group of peripheral interrupts. Bits in IABR are set while the corresponding interrupt service routines are in progress.

**Table 52. Interrupt Active Bit Register 1 (IABR1 - 0xE000 E304)**

Bit	Name	Function
0	IAB_PLL1	PLL1 (Alt PLL) interrupt active. Read: 0 indicates that the interrupt is not active, 1 indicates that the interrupt is active.
1	IAB_USBACT	USB Activity interrupt active. See functional description for bit 0.
2	IAB_CANACT	CAN Activity interrupt active. See functional description for bit 0.
3	IAB_UART4	UART4 interrupt active. See functional description for bit 0.
4	IAB_SSP2	SSP2 interrupt active. See functional description for bit 0.
5	IAB_LCD	LCD interrupt active. See functional description for bit 0.
6	IAB_GPIO	GPIO interrupt active. See functional description for bit 0.
7	IAB_PWM0	PWM0 interrupt active. See functional description for bit 0.
8	IAB_EEPROM	EEPROM interrupt active. See functional description for bit 0.
31:9	-	Reserved. The value read from a reserved bit is not defined.

### 6.5.11 Interrupt Priority Register 0 (IPR0 - 0xE000 E400)

The IPR0 register controls the priority of the first 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 53. Interrupt Priority Register 0 (IPR0 - 0xE000 E400)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_WDT	Watchdog Timer interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_TIMER0	Timer 0 interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_TIMER1	Timer 1 interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_TIMER2	Timer 2 interrupt priority. See functional description for bits 7-3.

### 6.5.12 Interrupt Priority Register 1 (IPR1 - 0xE000 E404)

The IPR1 register controls the priority of the second group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 54. Interrupt Priority Register 1 (IPR1 - 0xE000 E404)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_TIMER3	Timer 3 interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_UART0	UART0 interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_UART1	UART1 interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_UART2	UART2 interrupt priority. See functional description for bits 7-3.

### 6.5.13 Interrupt Priority Register 2 (IPR2 - 0xE000 E408)

The IPR2 register controls the priority of the third group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 55. Interrupt Priority Register 2 (IPR2 - 0xE000 E408)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_UART3	UART3 interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_PWM1	PWM1 interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_I2C0	I <sup>2</sup> C0 interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_I2C1	I <sup>2</sup> C1 interrupt priority. See functional description for bits 7-3.

### 6.5.14 Interrupt Priority Register 3 (IPR3 - 0xE000 E40C)

The IPR3 register controls the priority of the fourth group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 56. Interrupt Priority Register 3 (IPR3 - 0xE000 E40C)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_I2C2	I <sup>2</sup> C2 interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	-	Reserved. Read value is undefined, only zero should be written.
18:8	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_SSP0	SSP0 interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_SSP1	SSP1 interrupt priority. See functional description for bits 7-3.

### 6.5.15 Interrupt Priority Register 4 (IPR4 - 0xE000 E410)

The IPR4 register controls the priority of the fifth group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 57. Interrupt Priority Register 4 (IPR4 - 0xE000 E410)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_PLL0	PLL0 (Main PLL) interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_RTC	Real Time Clock (RTC) interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_EINT0	External Interrupt 0 interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_EINT1	External Interrupt 1 interrupt priority. See functional description for bits 7-3.

### 6.5.16 Interrupt Priority Register 5 (IPR5 - 0xE000 E414)

The IPR5 register controls the priority of the sixth group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 58. Interrupt Priority Register 5 (IPR5 - 0xE000 E414)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_EINT2	External Interrupt 2 interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_EINT3	External Interrupt 3 interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_ADC	ADC interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_BOD	BOD interrupt priority. See functional description for bits 7-3.

### 6.5.17 Interrupt Priority Register 6 (IPR6 - 0xE000 E418)

The IPR6 register controls the priority of the seventh group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 59. Interrupt Priority Register 6 (IPR6 - 0xE000 E418)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_USB	USB interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_CAN	CAN interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_DMA	GPDMA interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_I2S	I <sup>2</sup> S interrupt priority. See functional description for bits 7-3.

### 6.5.18 Interrupt Priority Register 7 (IPR7 - 0xE000 E41C)

The IPR7 register controls the priority of the eighth group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 60. Interrupt Priority Register 7 (IPR7 - 0xE000 E41C)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_ENET	Ethernet interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_SD	SD Card interface interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_MCPWM	Motor Control PWM interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_QEI	Quadrature Encoder Interface interrupt priority. See functional description for bits 7-3.

### 6.5.19 Interrupt Priority Register 8 (IPR8 - 0xE000 E420)

The IPR8 register controls the priority of the ninth and last group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 61. Interrupt Priority Register 8 (IPR8 - 0xE000 E420)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_PLL1	PLL1 (Alt PLL) interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_USBACT	USB Activity interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_CANACT	CAN Activity interrupt priority. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_UART4	UART4 interrupt priority. See functional description for bits 7-3.

### 6.5.20 Interrupt Priority Register 9 (IPR9 - 0xE000 E424)

The IPR9 register controls the priority of the tenth group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 62. Interrupt Priority Register 9 (IPR9 - 0xE000 E424)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_SSP2	SSP2 interrupt priority. 0 = highest priority. 31 (0x1F) = lowest priority.
10:8	Unimplemented	These bits ignore writes, and read as 0.
15:11	IP_LCD	LCD controller interrupt priority. See functional description for bits 7-3.
18:16	Unimplemented	These bits ignore writes, and read as 0.
23:19	IP_GPIO	Priority of GPIO interrupts. See functional description for bits 7-3.
26:24	Unimplemented	These bits ignore writes, and read as 0.
31:27	IP_PWM0	PWM0 interrupt priority. See functional description for bits 7-3.

### 6.5.21 Interrupt Priority Register 10 (IPR10 - 0xE000 E428)

The IPR10 register controls the priority of the eleventh group of 4 peripheral interrupts. Each interrupt can have one of 32 priorities, where 0 is the highest priority.

**Table 63. Interrupt Priority Register 10 (IPR10 - 0xE000 E428)**

Bit	Name	Function
2:0	Unimplemented	These bits ignore writes, and read as 0.
7:3	IP_EEPROM	EEPROM programming interrupt. 0 = highest priority. 31 (0x1F) = lowest priority.
31:8	Unimplemented	These bits ignore writes, and read as 0.

### 6.5.22 Software Trigger Interrupt Register (STIR - 0xE000 EF00)

The STIR register provides an alternate way for software to generate an interrupt, in addition to using the ISPR registers. This mechanism can only be used to generate peripheral interrupts, not system exceptions.

By default, only privileged software can write to the STIR register. Unprivileged software can be given this ability if privileged software sets the USERSETMPEND bit in the CCR register (see [Section 38.4.2.8](#) and [Section 38.4.3.8](#)).

**Table 64. Software Trigger Interrupt Register (STIR - 0xE000 EF00)**

Bit	Name	Function
8:0	INTID	Writing a value to this field generates an interrupt for the specified Interrupt ID (see <a href="#">Table 41</a> ).
31:9	-	Reserved. Read value is undefined, only zero should be written.

### 7.1 LPC178x/177x pin configuration

For information about the individual LPC178x/177x devices, refer to specific data sheets. [Table 65](#) lists pins in order by pin name, and includes description of each potential pin function.

See the IOCON registers ([Section 8.4.1](#)) to configure the pins of each LPC178x/177x part for the desired function.

I/O pins on the LPC178x/177x are 5V tolerant and have input hysteresis unless otherwise indicated in the table below. Crystal pins, power pins, and reference voltage pins are not 5V tolerant. In addition, when pins are selected to be A to D converter inputs, they are no longer 5V tolerant and must be limited to the voltage at the ADC positive reference pin (VREFP).

**Table 65. LPC178x/177x pin description**

Symbol	Type	IOCON select <sup>[1]</sup>	Description
<b>P0[0] to P0[31]</b>	I/O		<b>Port 0:</b> Port 0 provides up to 32 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.
P0[0]/ CAN_RD1/ U3_TXD/I2C1_SDA/ U0_TXD	I/O	0	<b>P0[0]</b> — General purpose digital input/output pin.
	I	1	<b>CAN_RD1</b> — CAN1 receiver input.
	O	2	<b>U3_TXD</b> — Transmitter output for UART 3.
	I/O	3	<b>I2C1_SDA</b> — I <sup>2</sup> C1 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I/O	4	<b>U0_TXD</b> — Transmitter output for UART 0.
P0[1]/ CAN1_TD/ U3_RXD/I2C1_SCL/ U0_RXD	I/O	0	<b>P0[1]</b> — General purpose digital input/output pin.
	O	1	<b>CAN1_TD</b> — CAN1 transmitter output.
	I	2	<b>U3_RXD</b> — Receiver input for UART 3.
	I/O	3	<b>I2C1_SCL</b> — I <sup>2</sup> C1 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I	4	<b>U0_RXD</b> — Receiver input for UART 0.
P0[2]/ U0_TXD/ U3_TXD	I/O	0	<b>P0[2]</b> — General purpose digital input/output pin.
	O	1	<b>U0_TXD</b> — Transmitter output for UART 0. Used for ISP communication, see <a href="#">Section 36.1</a> .
	O	2	<b>U3_TXD</b> — Transmitter output for UART 3.
P0[3]/ U0_RXD/ U3_RXD	I/O	0	<b>P0[3]</b> — General purpose digital input/output pin.
	I	1	<b>U0_RXD</b> — Receiver input for UART 0. Used for ISP communication, see <a href="#">Section 36.1</a> .
	I	2	<b>U3_RXD</b> — Receiver input for UART 3.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>[1]</sup>	Description
P0[4]/ I2S_RX_SCK/ CAN_RD2/ T2_CAP0/ LCD_VD[0]	I/O	0	<b>P0[4]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_RX_SCK</b> — I <sup>2</sup> S Receive clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the <i>I<sup>2</sup>S-bus specification</i> .
	I	2	<b>CAN_RD2</b> — CAN2 receiver input.
	I	3	<b>T2_CAP0</b> — Capture input for Timer 2, channel 0.
	O	7	<b>LCD_VD[0]</b> — LCD data.
P0[5]/ I2S_RX_WS/ CAN_TD2/ T2_CAP1/ LCD_VD[1]	I/O	0	<b>P0[5]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_RX_WS</b> — I <sup>2</sup> S Receive word select. It is driven by the master and received by the slave. Corresponds to the signal WS in the <i>I<sup>2</sup>S-bus specification</i> .
	O	2	<b>CAN_TD2</b> — CAN2 transmitter output.
	I	3	<b>T2_CAP1</b> — Capture input for Timer 2, channel 1.
	O	7	<b>LCD_VD[1]</b> — LCD data.
P0[6]/ I2S_RX_SDA/ SSP1_SSEL/ T2_MAT0/ U1_RTS/ LCD_VD[8]	I/O	0	<b>P0[6]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_RX_SDA</b> — I <sup>2</sup> S Receive data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the <i>I<sup>2</sup>S-bus specification</i> .
	I/O	2	<b>SSP1_SSEL1</b> — Slave Select for SSP1.
	O	3	<b>T2_MAT0</b> — Match output for Timer 2, channel 0.
	O	4	<b>U1_RTS</b> — Request to Send output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
P0[7]/ I2S_TX_SCK/ SSP1_SCK/ T2_MAT1/ LCD_VD[9]	I/O	0	<b>P0[7]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_TX_SCK</b> — I <sup>2</sup> S transmit clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the <i>I<sup>2</sup>S-bus specification</i> .
	I/O	2	<b>SSP1_SCK</b> — Serial Clock for SSP1.
	O	3	<b>T2_MAT1</b> — Match output for Timer 2, channel 1.
	O	7	<b>LCD_VD[9]</b> — LCD data.
P0[8]/ I2S_TX_WS/ SSP1_MISO/ T2_MAT2/ LCD_VD[16]	I/O	0	<b>P0[8]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_TX_WS</b> — I <sup>2</sup> S Transmit word select. It is driven by the master and received by the slave. Corresponds to the signal WS in the <i>I<sup>2</sup>S-bus specification</i> .
	I/O	2	<b>SSP1_MISO</b> — Master In Slave Out for SSP1.
	O	3	<b>T2_MAT2</b> — Match output for Timer 2, channel 2.
	O	7	<b>LCD_VD[16]</b> — LCD data.
P0[9]/ I2S_TX_SDA/ SSP1_MOSI/ T2_MAT3/ LCD_VD[17]	I/O	0	<b>P0[9]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2S_TX_SDA</b> — I <sup>2</sup> S transmit data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the <i>I<sup>2</sup>S-bus specification</i> .
	I/O	2	<b>SSP1_MOSI</b> — Master Out Slave In for SSP1.
	O	3	<b>T2_MAT3</b> — Match output for Timer 2, channel 3.
	O	7	<b>LCD_VD[17]</b> — LCD data.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P0[10]/ U2_TXD/ I2C2_SDA/ T3_MAT0	I/O	0	<b>P0[10]</b> — General purpose digital input/output pin.
	O	1	<b>U2_TXD</b> — Transmitter output for UART 2.
	I/O	2	<b>I2C2_SDA</b> — I <sup>2</sup> C2 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	O	3	<b>T3_MAT0</b> — Match output for Timer 3, channel 0.
P0[11]/ U2_RXD/ I2C2_SCL/ T3_MAT1	I/O	0	<b>P0[11]</b> — General purpose digital input/output pin.
	I	1	<b>U2_RXD</b> — Receiver input for UART 2.
	I/O	2	<b>I2C2_SCL</b> — I <sup>2</sup> C2 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	O	3	<b>T3_MAT1</b> — Match output for Timer 3, channel 1.
P0[12]/ USB_PPWR2/ SSP1_MISO/ AD0[6]	I/O	0	<b>P0[12]</b> — General purpose digital input/output pin.
	O	1	<b>USB_PPWR2</b> — Port Power enable signal for USB port 2.
	I/O	2	<b>SSP1_MISO</b> — Master In Slave Out for SSP1.
	I	3	<b>AD0[6]</b> — A/D converter 0, input 6. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
P0[13]/ USB_UP_LED2/ SSP1_MOSI/ AD0[7]	I/O	0	<b>P0[13]</b> — General purpose digital input/output pin.
	O	1	<b>USB_UP_LED2</b> — USB port 2 GoodLink LED indicator. It is LOW when device is configured (non-control endpoints enabled). It is HIGH when the device is not configured or during global suspend.
	I/O	2	<b>SSP1_MOSI</b> — Master Out Slave In for SSP1.
	I	3	<b>AD0[7]</b> — A/D converter 0, input 7. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
P0[14]/ USB_HSTEN2/ SSP1_SSEL/ USB_CONNECT2	I/O	0	<b>P0[14]</b> — General purpose digital input/output pin.
	O	1	<b>USB_HSTEN2</b> — Host Enabled status for USB port 2.
	I/O	2	<b>SSP1_SSEL</b> — Slave Select for SSP1.
	O	3	<b>USB_CONNECT2</b> — SoftConnect control for USB port 2. Signal used to switch an external 1.5 k $\Omega$ resistor under software control. Used with the SoftConnect USB feature.
P0[15]/ U1_TXD/ SSP0_SCK	I/O	0	<b>P0[15]</b> — General purpose digital input/output pin.
	O	1	<b>U1_TXD</b> — Transmitter output for UART 1.
	I/O	2	<b>SSP0_SCK</b> — Serial clock for SSP0.
P0[16]/ U1_RXD/ SSP0_SSEL/	I/O	0	<b>P0[16]</b> — General purpose digital input/output pin.
	I	1	<b>U1_RXD</b> — Receiver input for UART 1.
	I/O	2	<b>SSP0_SSEL</b> — Slave Select for SSP0.
P0[17]/ U1_CTS/ SSP0_MISO	I/O	0	<b>P0[17]</b> — General purpose digital input/output pin.
	I	1	<b>U1_CTS</b> — Clear to Send input for UART 1.
	I/O	2	<b>SSP0_MISO</b> — Master In Slave Out for SSP0.
P0[18]/ U1_DCD/ SSP0_MOSI	I/O	0	<b>P0[18]</b> — General purpose digital input/output pin.
	I	1	<b>U1_DCD</b> — Data Carrier Detect input for UART 1.
	I/O	2	<b>SSP0_MOSI</b> — Master Out Slave In for SSP0.



Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P0[19]/ U1_DSR/ SD_CLK/ I2C1_SDA	I/O	0	<b>P0[19]</b> — General purpose digital input/output pin.
	I	1	<b>U1_DSR</b> — Data Set Ready input for UART 1.
	O	2	<b>SD_CLK</b> — Clock output line for SD card interface.
	I/O	3	<b>I2C1_SDA</b> — I <sup>2</sup> C1 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
P0[20]/ U1_DTR/ SD_CMD/ I2C1_SCL	I/O	0	<b>P0[20]</b> — General purpose digital input/output pin.
	O	1	<b>U1_DTR</b> — Data Terminal Ready output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
	I/O	2	<b>SD_CMD</b> — Command line for SD card interface.
	I/O	3	<b>I2C1_SCL</b> — I <sup>2</sup> C1 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
P0[21]/ U1_RI/ SD_PWR/ U4_OE/ CAN_RD1	I/O	0	<b>P0[21]</b> — General purpose digital input/output pin.
	I	1	<b>U1_RI</b> — Ring Indicator input for UART 1.
	O	2	<b>SD_PWR</b> — Power Supply Enable for external SD card power supply.
	O	3	<b>U4_OE</b> — RS-485/EIA-485 output enable signal for UART 4.
	I/O	4	<b>CAN_RD1</b> — CAN1 receiver input.
P0[22]/ U1_RTS/ SD_DAT[0]/ U4_TXD/ CAN_TD1	I/O	0	<b>P0[22]</b> — General purpose digital input/output pin.
	O	1	<b>U1_RTS</b> — Request to Send output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
	I/O	2	<b>SD_DAT[0]</b> — Data line 0 for SD card interface.
	O	3	<b>U4_TXD</b> — Transmitter output for UART 4 (input/output in smart card mode).
	O	4	<b>CAN_TD1</b> — CAN1 transmitter output.
P0[23]/ AD0[0]/ I2S_RX_SCK/ T3_CAP0	I/O	0	<b>P0[23]</b> — General purpose digital input/output pin.
	I	1	<b>AD0[0]</b> — A/D converter 0, input 0. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
	I/O	2	<b>I2S_RX_SCK</b> — Receive Clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the <i>I<sup>2</sup>S-bus specification</i> .
P0[24]/ AD0[1]/ I2S_RX_WS/ T3_CAP1	I	3	<b>T3_CAP0</b> — Capture input for Timer 3, channel 0.
	I/O	0	<b>P0[24]</b> — General purpose digital input/output pin.
	I	1	<b>AD0[1]</b> — A/D converter 0, input 1. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
P0[25]/ AD0[2]/ I2S_RX_SDA/ U3_TXD	I/O	2	<b>I2S_RX_WS</b> — Receive Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in the <i>I<sup>2</sup>S-bus specification</i> .
	I	3	<b>T3_CAP1</b> — Capture input for Timer 3, channel 1.
	I/O	0	<b>P0[25]</b> — General purpose digital input/output pin.
P0[25]/ AD0[2]/ I2S_RX_SDA/ U3_TXD	I	1	<b>AD0[2]</b> — A/D converter 0, input 2. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
	I/O	2	<b>I2S_RX_SDA</b> — Receive data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the <i>I<sup>2</sup>S-bus specification</i> .
	O	3	<b>U3_TXD</b> — Transmitter output for UART 3.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>[1]</sup>	Description
P0[26]/ AD0[3]/ DAC_OUT/ U3_RXD	I/O	0	<b>P0[26]</b> — General purpose digital input/output pin.
	I	1	<b>AD0[3]</b> — A/D converter 0, input 3. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
	O	2	<b>DAC_OUT</b> — D/A converter output. When configured as the DAC output, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ).
	I	3	<b>U3_RXD</b> — Receiver input for UART 3.
P0[27]/ I2C0_SDA/ USB_SDA	I/O	0	<b>P0[27]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2C0_SDA</b> — I <sup>2</sup> C0 data input/output. (this pin uses a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I/O	2	<b>USB_SDA</b> — I <sup>2</sup> C serial data for communication with an external USB transceiver.
P0[28]/ I2C0_SCL/ USB_SCL	I/O	0	<b>P0[28]</b> — General purpose digital input/output pin.
	I/O	1	<b>I2C0_SCL0</b> — I <sup>2</sup> C0 clock input/output (this pin uses a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I/O	2	<b>USB_SCL</b> — I <sup>2</sup> C serial clock for communication with an external USB transceiver.
P0[29]/ USB_D+1/ EINT0	I/O	0	<b>P0[29]</b> — General purpose digital input/output pin. When used as GPIO, P0[29] shares a direction control with P0[30].
	I/O	1	<b>USB_D+1</b> — USB port 1 bidirectional D+ line.
	I	2	<b>EINT_0</b> — External interrupt 0 input.
P0[30]/ USB_D–1/ EINT1	I/O	0	<b>P0[30]</b> — General purpose digital input/output pin. When used as GPIO, P0[30] shares a direction control with P0[29].
	I/O	1	<b>USB_D–1</b> — USB port 1 bidirectional D– line.
	I	2	<b>EINT_1</b> — External interrupt 1 input.
P0[31]/ USB_D+2	I/O	0	<b>P0[31]</b> — General purpose digital input/output pin.
	I/O	1	<b>USB_D+2</b> — USB port 2 bidirectional D+ line.
<b>P1[0] to P1[31]</b>	I/O		<b>Port 1:</b> Port 1 provides up to 32 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.
P1[0]/ ENET_TXD0/ T3_CAP1/ SSP2_SCK	I/O	0	<b>P1[0]</b> — General purpose digital input/output pin.
	O	1	<b>ENET_TXD0</b> — Ethernet transmit data 0 (RMII/MII interface).
	I	3	<b>T3_CAP1</b> — Capture input for Timer 3, channel 1.
	I/O	4	<b>SSP2_SCK</b> — Serial clock for SSP2.
P1[1]/ ENET_TXD1/ T3_MAT3/ SSP2_MOSI	I/O	0	<b>P1[1]</b> — General purpose digital input/output pin.
	O	1	<b>ENET_TXD1</b> — Ethernet transmit data 1 (RMII/MII interface).
	O	3	<b>T3_MAT3</b> — Match output for Timer 3, channel 3.
	I/O	4	<b>SSP2_MOSI</b> — Master Out Slave In for SSP2.
P1[2]/ ENET_TXD2/ SD_CLK/ PWM0[1]	I/O	0	<b>P1[2]</b> — General purpose digital input/output pin.
	O	1	<b>ENET_TXD2</b> — Ethernet transmit data 2 (MII interface).
	O	2	<b>SD_CLK</b> — Clock output line for SD card interface.
	O	3	<b>PWM0[1]</b> — Pulse Width Modulator 0, output 1.
P1[3]/ ENET_TXD3/ SD_CMD/ PWM0[2]	I/O	0	<b>P1[3]</b> — General purpose digital input/output pin.
	O	1	<b>ENET_TXD3</b> — Ethernet transmit data 3 (MII interface).
	I/O	2	<b>SD_CMD</b> — Command line for SD card interface.
	O	3	<b>PWM0[2]</b> — Pulse Width Modulator 0, output 2.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P1[4]/	I/O	0	<b>P1[4]</b> — General purpose digital input/output pin.
ENET_TX_EN/	O	1	<b>ENET_TX_EN</b> — Ethernet transmit data enable (RMII/MII interface).
T3_MAT2/	O	3	<b>T3_MAT2</b> — Match output for Timer 3, channel 2.
SSP2_MISO	I/O	4	<b>SSP2_MISO</b> — Master In Slave Out for SSP2.
P1[5]/	I/O	0	<b>P1[5]</b> — General purpose digital input/output pin.
ENET_TX_ER/	O	1	<b>ENET_TX_ER</b> — Ethernet Transmit Error (MII interface).
SD_PWR/ PWM0[3]	O	2	<b>SD_PWR</b> — Power Supply Enable for external SD card power supply.
	O	3	<b>PWM0[3]</b> — Pulse Width Modulator 0, output 3.
P1[6]/	I/O	0	<b>P1[6]</b> — General purpose digital input/output pin.
ENET_TX_CLK/	I	1	<b>ENET_TX_CLK</b> — Ethernet Transmit Clock (MII interface).
SD_DAT[0]/	I/O	2	<b>SD_DAT[0]</b> — Data line 0 for SD card interface.
PWM0[4]	O	3	<b>PWM0[4]</b> — Pulse Width Modulator 0, output 4.
P1[7]/ ENET_COL/	I/O	0	<b>P1[7]</b> — General purpose digital input/output pin.
SD_DAT[1]/	I	1	<b>ENET_COL</b> — Ethernet Collision detect (MII interface).
PWM0[5]	I/O	2	<b>SD_DAT[1]</b> — Data line 1 for SD card interface.
	O	3	<b>PWM0[5]</b> — Pulse Width Modulator 0, output 5.
P1[8]/ ENET_CRS	I/O	0	<b>P1[8]</b> — General purpose digital input/output pin.
(ENET_CRS_DV)/	I	1	<b>ENET_CRS (ENET_CRS_DV)</b> — Ethernet Carrier Sense (MII interface) or Ethernet Carrier Sense/Data Valid (RMII interface).
T3_MAT1/	O	3	<b>T3_MAT1</b> — Match output for Timer 3, channel 1.
SSP2_SSEL	I/O	4	<b>SSP2_SSEL</b> — Slave Select for SSP2.
P1[9]/ ENET_RXD0/	I/O	0	<b>P1[9]</b> — General purpose digital input/output pin.
T3_MAT0	I	1	<b>ENET_RXD0</b> — Ethernet receive data 0 (RMII/MII interface).
	O	3	<b>T3_MAT0</b> — Match output for Timer 3, channel 0.
P1[10]/	I/O	0	<b>P1[10]</b> — General purpose digital input/output pin.
ENET_RXD1/	I	1	<b>ENET_RXD1</b> — Ethernet receive data 1 (RMII/MII interface).
T3_CAP0	I	3	<b>T3_CAP0</b> — Capture input for Timer 3, channel 0.
P1[11]/	I/O	0	<b>P1[11]</b> — General purpose digital input/output pin.
ENET_RXD2/	I	1	<b>ENET_RXD2</b> — Ethernet Receive Data 2 (MII interface).
SD_DAT[2]/	I/O	2	<b>SD_DAT[2]</b> — Data line 2 for SD card interface.
PWM0[6]	O	3	<b>PWM0[6]</b> — Pulse Width Modulator 0, output 6.
P1[12]/	I/O	0	<b>P1[12]</b> — General purpose digital input/output pin.
ENET_RXD3/	I	1	<b>ENET_RXD3</b> — Ethernet Receive Data (MII interface).
SD_DAT[3]/	I/O	2	<b>SD_DAT[3]</b> — Data line 3 for SD card interface.
PWM0_CAP0	I	3	<b>PWM0_CAP0</b> — Capture input for PWM0, channel 0.
P1[13]/	I/O	0	<b>P1[13]</b> — General purpose digital input/output pin.
ENET_RX_DV	I	1	<b>ENET_RX_DV</b> — Ethernet Receive Data Valid (MII interface).
P1[14]/	I/O	0	<b>P1[14]</b> — General purpose digital input/output pin.
ENET_RX_ER/	I	1	<b>ENET_RX_ER</b> — Ethernet receive error (RMII/MII interface).
T2_CAP0	I	3	<b>T2_CAP0</b> — Capture input for Timer 2, channel 0.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>[1]</sup>	Description
P1[15]/ ENET_RX_CLK (ENET_REF_CLK)/ I2C2_SDA	I/O	0	<b>P1[15]</b> — General purpose digital input/output pin.
	I	1	<b>ENET_RX_CLK (ENET_REF_CLK)</b> — Ethernet Receive Clock (MII interface) or Ethernet Reference Clock (RMII interface).
	I/O	3	<b>I2C2_SDA</b> — I <sup>2</sup> C2 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
P1[16]/ ENET_MDC/ I2S_TX_MCLK	I/O	0	<b>P1[16]</b> — General purpose digital input/output pin.
	O	1	<b>ENET_MDC</b> — Ethernet MIIM clock.
	O	2	<b>I2S_TX_MCLK</b> — I <sup>2</sup> S transmitter master clock output.
P1[17]/ ENET_MDIO/ I2S_RX_MCLK	I/O	0	<b>P1[17]</b> — General purpose digital input/output pin.
	I/O	1	<b>ENET_MDIO</b> — Ethernet MIIM data input and output.
	O	2	<b>I2S_RX_MCLK</b> — I <sup>2</sup> S receiver master clock output.
P1[18]/ USB_UP_LED1/ PWM1[1]/ T1_CAP0/ SSP1_MISO	I/O	0	<b>P1[18]</b> — General purpose digital input/output pin.
	O	1	<b>USB_UP_LED1</b> — USB port 1 GoodLink LED indicator. It is LOW when device is configured (non-control endpoints enabled). It is HIGH when the device is not configured or during global suspend.
	O	2	<b>PWM1[1]</b> — Pulse Width Modulator 1, channel 1 output.
	I	3	<b>T1_CAP0</b> — Capture input for Timer 1, channel 0.
	I/O	5	<b>SSP1_MISO</b> — Master In Slave Out for SSP1.
P1[19]/USB_TX_E1/ USB_PPWR1/ T1_CAP1/ MC_0A/ SSP1_SCK/ U2_OE	I/O	0	<b>P1[19]</b> — General purpose digital input/output pin.
	O	1	<b>USB_TX_E1</b> — Transmit Enable signal for USB port 1 (OTG transceiver).
	O	2	<b>USB_PPWR1</b> — Port Power enable signal for USB port 1.
	I	3	<b>T1_CAP1</b> — Capture input for Timer 1, channel 1.
	O	4	<b>MC_0A</b> — Motor control PWM channel 0, output A.
	I/O	5	<b>SSP1_SCK</b> — Serial clock for SSP1.
	O	6	<b>U2_OE</b> — RS-485/EIA-485 output enable signal for UART 2.
	I/O	7	<b>P1[20]</b> — General purpose digital input/output pin.
P1[20]/ USB_TX_DP1/ PWM1[2]/ QEI_PHA/ MC_FB0/ SSP0_SCK/ LCD_VD[6]/ LCD_VD[10]	I/O	0	<b>P1[20]</b> — General purpose digital input/output pin.
	O	1	<b>USB_TX_DP1</b> — D+ transmit data for USB port 1 (OTG transceiver).
	O	2	<b>PWM1[2]</b> — Pulse Width Modulator 1, channel 2 output.
	I	3	<b>QEI_PHA</b> — Quadrature Encoder Interface PHA input.
	I	4	<b>MC_FB0</b> — Motor control PWM channel 0 feedback input.
	I/O	5	<b>SSP0_SCK0</b> — Serial clock for SSP0.
	O	6	<b>LCD_VD[6]</b> — LCD data.
	O	7	<b>LCD_VD[10]</b> — LCD data.
P1[21]/ USB_TX_DM1/ PWM1[3]/ SSP0_SSEL/ MC_ABORT/ LCD_VD[7]/ LCD_VD[11]	I/O	0	<b>P1[21]</b> — General purpose digital input/output pin.
	O	1	<b>USB_TX_DM1</b> — D– transmit data for USB port 1 (OTG transceiver).
	O	2	<b>PWM1[3]</b> — Pulse Width Modulator 1, channel 3 output.
	I/O	3	<b>SSP0_SSEL</b> — Slave Select for SSP0.
	I	4	<b>MC_ABORT</b> — Motor control PWM, active low fast abort.
	O	6	<b>LCD_VD[7]</b> — LCD data.
	O	7	<b>LCD_VD[11]</b> — LCD data.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P1[22]/ USB_RCV1/	I/O	0	<b>P1[22]</b> — General purpose digital input/output pin.
USB_PWRD1/	I	1	<b>USB_RCV1</b> — Differential receive data for USB port 1 (OTG transceiver).
T1_MAT0/ MC_0B/	I	2	<b>USB_PWRD1</b> — Power Status for USB port 1 (host power switch).
SSP1_MOSI/	O	3	<b>T1_MAT0</b> — Match output for Timer 1, channel 0.
LCD_VD[8]/	O	4	<b>MC_0B</b> — Motor control PWM channel 0, output B.
LCD_VD[12]	I/O	5	<b>SSP1_MOSI</b> — Master Out Slave In for SSP1.
	O	6	<b>LCD_VD[8]</b> — LCD data.
	O	7	<b>LCD_VD[12]</b> — LCD data.
P1[23]/	I/O	0	<b>P1[23]</b> — General purpose digital input/output pin.
USB_RX_DP1/	I	1	<b>USB_RX_DP1</b> — D+ receive data for USB port 1 (OTG transceiver).
PWM1[4]/ QEI_PHB/	O	2	<b>PWM1[4]</b> — Pulse Width Modulator 1, channel 4 output.
MC_FB1/	I	3	<b>QEI_PHB</b> — Quadrature Encoder Interface PHB input.
SSP0_MISO/	I	4	<b>MC_FB1</b> — Motor control PWM channel 1 feedback input.
LCD_VD[9]/	I/O	5	<b>SSP0_MISO</b> — Master In Slave Out for SSP0.
LCD_VD[13]	O	6	<b>LCD_VD[9]</b> — LCD data.
	O	7	<b>LCD_VD[13]</b> — LCD data.
P1[24]/	I/O	0	<b>P1[24]</b> — General purpose digital input/output pin.
USB_RX_DM1/	I	1	<b>USB_RX_DM1</b> — D- receive data for USB port 1 (OTG transceiver).
PWM1[5]/ QEI_IDX/	O	2	<b>PWM1[5]</b> — Pulse Width Modulator 1, channel 5 output.
MC_FB2/	I	3	<b>QEI_IDX</b> — Quadrature Encoder Interface INDEX input.
SSP0_MOSI/	I	4	<b>MC_FB2</b> — Motor control PWM channel 2 feedback input.
LCD_VD[10]/	I/O	5	<b>SSP0_MOSI</b> — Master Out Slave in for SSP0.
LCD_VD[14]	O	6	<b>LCD_VD[10]/LCD_VD[14]</b> — LCD data.
	O	7	<b>LCD_VD[10]/LCD_VD[14]</b> — LCD data.
P1[25]/ USB_LS1/	I/O	0	<b>P1[25]</b> — General purpose digital input/output pin.
USB_HSTEN1/	O	1	<b>USB_LS1</b> — Low Speed status for USB port 1 (OTG transceiver).
T1_MAT1/ MC_1A/	O	2	<b>USB_HSTEN1</b> — Host Enabled status for USB port 1.
CLKOUT/	O	3	<b>T1_MAT1</b> — Match output for Timer 1, channel 1.
LCD_VD[11]/	O	4	<b>MC_1A</b> — Motor control PWM channel 1, output A.
LCD_VD[15]	O	5	<b>CLKOUT</b> — Selectable clock output.
	O	6	<b>LCD_VD[11]</b> — LCD data.
	O	7	<b>LCD_VD[15]</b> — LCD data.
P1[26]/	I/O	0	<b>P1[26]</b> — General purpose digital input/output pin.
USB_SSPND1/	O	1	<b>USB_SSPND1</b> — USB port 1 Bus Suspend status (OTG transceiver).
PWM1[6]/ T0_CAP0/	O	2	<b>PWM1[6]</b> — Pulse Width Modulator 1, channel 6 output.
MC_1B/	I	3	<b>T0_CAP0</b> — Capture input for Timer 0, channel 0.
SSP1_SSEL/	O	4	<b>MC_1B</b> — Motor control PWM channel 1, output B.
LCD_VD[12]/	I/O	5	<b>SSP1_SSEL</b> — Slave Select for SSP1.
LCD_VD[20]	O	6	<b>LCD_VD[12]</b> — LCD data.
	O	7	<b>LCD_VD[20]</b> — LCD data.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P1[27]/ USB_INT1/ USB_OVRCR1/ T0_CAP1/ CLKOUT/ LCD_VD[13]/ LCD_VD[21]	I/O I I I O O O	0 1 2 3 4 6 7	<b>P1[27]</b> — General purpose digital input/output pin. <b>USB_INT1</b> — USB port 1 OTG transceiver interrupt (OTG transceiver). <b>USB_OVRCR1</b> — USB port 1 Over-Current status. <b>T0_CAP1</b> — Capture input for Timer 0, channel 1. <b>CLKOUT</b> — Selectable clock output. <b>LCD_VD[13]</b> — LCD data. <b>LCD_VD[21]</b> — LCD data.
P1[28]/ USB_SCL1/ PWM1_CAP0/ T0_MAT0/ MC_2A/ SSP0_SSEL/ LCD_VD[14]/ LCD_VD[22]	I/O I/O I O O I/O O O	0 1 2 3 4 5 6 7	<b>P1[28]</b> — General purpose digital input/output pin. <b>USB_SCL1</b> — USB port 1 I <sup>2</sup> C serial clock (OTG transceiver). <b>PWM1_CAP0</b> — Capture input for PWM1, channel 0. <b>T0_MAT0</b> — Match output for Timer 0, channel 0. <b>MC_2A</b> — Motor control PWM channel 2, output A. <b>SSP0_SSEL</b> — Slave Select for SSP0. <b>LCD_VD[14]</b> — LCD data. <b>LCD_VD[22]</b> — LCD data.
P1[29]/ USB_SDA1/ PWM1_CAP1/ T0_MAT1/ MC_2B/ U4_TXD/ LCD_VD[15]/ LCD_VD[23]	I/O I/O I O O O O O	0 1 2 3 4 5 6 7	<b>P1[29]</b> — General purpose digital input/output pin. <b>USB_SDA1</b> — USB port 1 I <sup>2</sup> C serial data (OTG transceiver). <b>PWM1_CAP1</b> — Capture input for PWM1, channel 1. <b>T0_MAT1</b> — Match output for Timer 0, channel 1. <b>MC_2B</b> — Motor control PWM channel 2, output B. <b>U4_TXD</b> — Transmitter output for UART 4 (input/output in smart card mode). <b>LCD_VD[15]</b> — LCD data. <b>LCD_VD[23]</b> — LCD data.
P1[30]/ USB_PWRD2/ USB_VBUS/ AD0[4]/ I2C0_SDA/ U3_OE	I/O I I I I/O O	0 1 2 3 4 5	<b>P1[30]</b> — General purpose digital input/output pin. <b>USB_PWRD2</b> — Power Status for USB port 2. <b>USB_VBUS</b> — Monitors the presence of USB bus power. <b>Note:</b> This signal must be HIGH for USB reset to occur. <b>AD0[4]</b> — A/D converter 0, input 4. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ). <b>I2C0_SDA</b> — I <sup>2</sup> C0 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details). <b>U3_OE</b> — RS-485/EIA-485 output enable signal for UART 3.
P1[31]/ USB_OVRCR2/ SSP1_SCK/ AD0[5]/ I2C0_SCL	I/O I I/O I I/O	0 1 2 3 4	<b>P1[31]</b> — General purpose digital input/output pin. <b>USB_OVRCR2</b> — Over-Current status for USB port 2. <b>SSP1_SCK</b> — Serial Clock for SSP1. <b>AD0[5]</b> — A/D converter 0, input 5. When configured as an ADC input, the digital function of the pin must be disabled (see <a href="#">Section 8.4.1</a> ). <b>I2C0_SCL</b> — I <sup>2</sup> C0 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
<b>P2[0] to P2[31]</b>	I/O		<b>Port 2:</b> Port 2 provides up to 32 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P2[0]/ PWM1[1]/ U1_TXD/ LCD_PWR	I/O	0	<b>P2[0]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[1]</b> — Pulse Width Modulator 1, channel 1 output.
	O	2	<b>U1_TXD</b> — Transmitter output for UART 1.
	O	7	<b>LCD_PWR</b> — LCD panel power enable.
P2[1]/ PWM1[2]/ U1_RXD/ LCD_LE	I/O	0	<b>P2[1]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[2]</b> — Pulse Width Modulator 1, channel 2 output.
	I	2	<b>U1_RXD</b> — Receiver input for UART 1.
	O	7	<b>LCD_LE</b> — Line end signal.
P2[2]/ PWM1[3]/ U1_CTS/ T2_MAT3/ TRACEDATA[3]/ LCD_DCLK	I/O	0	<b>P2[2]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[3]</b> — Pulse Width Modulator 1, channel 3 output.
	I	2	<b>U1_CTS</b> — Clear to Send input for UART 1.
	O	3	<b>T2_MAT3</b> — Match output for Timer 2, channel 3.
	O	5	<b>TRACEDATA[3]</b> — Trace data, bit 3.
	O	7	<b>LCD_DCLK</b> — LCD panel clock.
P2[3]/ PWM1[4]/ U1_DCD/ T2_MAT2/ TRACEDATA[2]/ LCD_FP	I/O	0	<b>P2[3]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[4]</b> — Pulse Width Modulator 1, channel 4 output.
	I	2	<b>U1_DCD</b> — Data Carrier Detect input for UART 1.
	O	3	<b>T2_MAT2</b> — Match output for Timer 2, channel 2.
	O	5	<b>TRACEDATA[2]</b> — Trace data, bit 2.
	O	7	<b>LCD_FP</b> — Frame pulse (STN). Vertical synchronization pulse (TFT).
P2[4]/ PWM1[5]/ U1_DSR/ T2_MAT1/ TRACEDATA[1]/ LCD_ENAB_M	I/O	0	<b>P2[4]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[5]</b> — Pulse Width Modulator 1, channel 5 output.
	I	2	<b>U1_DSR</b> — Data Set Ready input for UART 1.
	O	3	<b>T2_MAT1</b> — Match output for Timer 2, channel 1.
	O	5	<b>TRACEDATA[1]</b> — Trace data, bit 1.
	O	7	<b>LCD_ENAB_M</b> — STN AC bias drive or TFT data enable output.
P2[5]/ PWM1[6]/ U1_DTR/ T2_MAT0/ TRACEDATA[0]/ LCD_LP	I/O	0	<b>P2[5]</b> — General purpose digital input/output pin.
	O	1	<b>PWM1[6]</b> — Pulse Width Modulator 1, channel 6 output.
	O	2	<b>U1_DTR</b> — Data Terminal Ready output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
	O	3	<b>T2_MAT0</b> — Match output for Timer 2, channel 0.
	O	5	<b>TRACEDATA[0]</b> — Trace data, bit 0.
	O	7	<b>LCD_LP</b> — Line synchronization pulse (STN). Horizontal synchronization pulse (TFT).

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P2[6]/ PWM1_CAP0/	I/O	0	<b>P2[6]</b> — General purpose digital input/output pin.
U1_RI/ T2_CAP0/	I	1	<b>PWM1_CAP0</b> — Capture input for PWM1, channel 0.
U2_OE/	I	2	<b>U1_RI</b> — Ring Indicator input for UART 1.
TRACECLK/	I	3	<b>T2_CAP0</b> — Capture input for Timer 2, channel 0.
LCD_VD[0]/	O	4	<b>U2_OE</b> — RS-485/EIA-485 output enable signal for UART 2.
LCD_VD[4]	O	5	<b>TRACECLK</b> — Trace clock.
	O	6	<b>LCD_VD[0]</b> — LCD data.
	O	7	<b>LCD_VD[4]</b> — LCD data.
P2[7]/ CAN_RD2/	I/O	0	<b>P2[7]</b> — General purpose digital input/output pin.
U1_RTS/	I	1	<b>CAN_RD2</b> — CAN2 receiver input.
LCD_VD[1]/	O	2	<b>U1_RTS</b> — Request to Send output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
LCD_VD[5]	O	6	<b>LCD_VD[1]</b> — LCD data.
	O	7	<b>LCD_VD[5]</b> — LCD data.
P2[8]/ CAN_TD2/	I/O	0	<b>P2[8]</b> — General purpose digital input/output pin.
U2_TXD/ U1_CTS/	O	1	<b>CAN_TD2</b> — CAN2 transmitter output.
ENET_MDC/	O	2	<b>U2_TXD</b> — Transmitter output for UART 2.
LCD_VD[2]/	I	3	<b>U1_CTS</b> — Clear to Send input for UART 1.
LCD_VD[6]	O	4	<b>ENET_MDC</b> — Ethernet MIIM clock.
	O	6	<b>LCD_VD[2]</b> — LCD data.
	O	7	<b>LCD_VD[6]</b> — LCD data.
P2[9]/	I/O	0	<b>P2[9]</b> — General purpose digital input/output pin.
USB_CONNECT1/	O	1	<b>USB_CONNECT1</b> — USB1 SoftConnect control. Signal used to switch an external 1.5 kΩ resistor under the software control. Used with the SoftConnect USB feature.
U2_RXD/ U4_RXD/	I	2	<b>U2_RXD</b> — Receiver input for UART 2.
ENET_MDIO/	I	3	<b>U4_RXD</b> — Receiver input for UART 4.
LCD_VD[3]/	I/O	4	<b>ENET_MDIO</b> — Ethernet MIIM data input and output.
LCD_VD[7]	I	6	<b>LCD_VD[3]</b> — LCD data.
	I	7	<b>LCD_VD[7]</b> — LCD data.
P2[10]/ EINT0/ NMI	I/O	0	<b>P2[10]</b> — General purpose digital input/output pin. This pin includes a 5 ns input glitch filter. <b>Note:</b> A LOW on this pin while $\overline{\text{RESET}}$ is LOW forces the on-chip boot loader to take over control of the part after a reset and go into ISP mode. See <a href="#">Section 36.3</a> .
	I	1	<b>EINT0</b> — External interrupt 0 input.
	I	2	<b>NMI</b> — Non-maskable interrupt input.
P2[11]/ EINT1/	I/O	0	<b>P2[11]</b> — General purpose digital input/output pin. This pin includes a 5 ns input glitch filter.
SD_DAT[1]/	I	1	<b>EINT1</b> — External interrupt 1 input.
I2S_TX_SCK/	I/O	2	<b>SD_DAT[1]</b> — Data line 1 for SD card interface.
LCD_CLKIN	I/O	3	<b>I2S_TX_SCK</b> — Transmit Clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the <i>I<sup>2</sup>S-bus specification</i> .
	O	7	<b>LCD_CLKIN</b> — LCD clock.



Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P2[12]/ EINT2/ SD_DAT[2]/ I2S_TX_WS/ LCD_VD[4]/ LCD_VD[3]/ LCD_VD[8]/ LCD_VD[18]	I/O	0	<b>P2[12]</b> — General purpose digital input/output pin. This pin includes a 5 ns input glitch filter.
	I	1	<b>EINT2</b> — External interrupt 2 input.
	I/O	2	<b>SD_DAT[2]</b> — Data line 2 for SD card interface.
	I/O	3	<b>I2S_TX_WS</b> — Transmit Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in the <i>I<sup>2</sup>S-bus specification</i> .
	O	4	<b>LCD_VD[4]</b> — LCD data.
	O	5	<b>LCD_VD[3]</b> — LCD data.
	O	6	<b>LCD_VD[8]</b> — LCD data.
	O	7	<b>LCD_VD[18]</b> — LCD data.
P2[13]/ EINT3/ SD_DAT[3]/ I2S_TX_SDA/ LCD_VD[5]/ LCD_VD[9]/ LCD_VD[19]	I/O	0	<b>P2[13]</b> — General purpose digital input/output pin. This pin includes a 5 ns input glitch filter.
	I	1	<b>EINT3</b> — External interrupt 3 input.
	I/O	2	<b>SD_DAT[3]</b> — Data line 3 for SD card interface.
	I/O	3	<b>I2S_TX_SDA</b> — Transmit data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the <i>I<sup>2</sup>S-bus specification</i> .
	O	5	<b>LCD_VD[5]</b> — LCD data.
	O	6	<b>LCD_VD[9]</b> — LCD data.
	O	7	<b>LCD_VD[19]</b> — LCD data.
P2[14]/ <u>EMC_CS2</u> / I2C1_SDA/ T2_CAP0	I/O	0	<b>P2[14]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CS2</b> — LOW active Chip Select 2 signal.
	I/O	2	<b>I2C1_SDA</b> — I <sup>2</sup> C1 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I	3	<b>T2_CAP0</b> — Capture input for Timer 2, channel 0.
P2[15]/ <u>EMC_CS3</u> / I2C1_SCL/ T2_CAP1	I/O	0	<b>P2[15]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CS3</b> — LOW active Chip Select 3 signal.
	I/O	2	<b>I2C1_SCL</b> — I <sup>2</sup> C1 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I	3	<b>T2_CAP1</b> — Capture input for Timer 2, channel 1.
P2[16]/ <u>EMC_CAS</u>	I/O	0	<b>P2[16]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CAS</b> — LOW active SDRAM Column Address Strobe.
P2[17]/ <u>EMC_RAS</u>	I/O	0	<b>P2[17]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_RAS</b> — LOW active SDRAM Row Address Strobe.
P2[18]/ EMC_CLK[0]	I/O	0	<b>P2[18]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CLK[0]</b> — SDRAM clock 0.
P2[19]/ EMC_CLK[1]	I/O	0	<b>P2[19]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CLK[1]</b> — SDRAM clock 1.
P2[20]/ <u>EMC_DYCS0</u>	I/O	0	<b>P2[20]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_DYCS0</b> — SDRAM chip select 0.
P2[21]/ <u>EMC_DYCS1</u>	I/O	0	<b>P2[21]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_DYCS1</b> — SDRAM chip select 1.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P2[22]/	I/O	0	<b>P2[22]</b> — General purpose digital input/output pin.
EMC_DYCS2/	O	1	<b>EMC_DYCS2</b> — SDRAM chip select 2.
SSP0_SCK/	I/O	2	<b>SSP0_SCK</b> — Serial clock for SSP0.
T3_CAP0	I	3	<b>T3_CAP0</b> — Capture input for Timer 3, channel 0.
P2[23]/	I/O	0	<b>P2[23]</b> — General purpose digital input/output pin.
EMC_DYCS3/	O	1	<b>EMC_DYCS3</b> — SDRAM chip select 3.
SSP0_SSEL/	I/O	2	<b>SSP0_SSEL</b> — Slave Select for SSP0.
T3_CAP1	I	3	<b>T3_CAP1</b> — Capture input for Timer 3, channel 1.
P2[24]/ EMC_CKE0	I/O	0	<b>P2[24]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CKE0</b> — SDRAM clock enable 0.
P2[25]/ EMC_CKE1	I/O	0	<b>P2[25]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CKE1</b> — SDRAM clock enable 1.
P2[26]/ EMC_CKE2/	I/O	0	<b>P2[26]</b> — General purpose digital input/output pin.
SSP0_MISO/	O	1	<b>EMC_CKE2</b> — SDRAM clock enable 2.
T3_MAT0	I/O	2	<b>SSP0_MISO</b> — Master In Slave Out for SSP0.
	O	3	<b>T3_MAT0</b> — Match output for Timer 3, channel 0.
P2[27]/ EMC_CKE3/	I/O	0	<b>P2[27]</b> — General purpose digital input/output pin.
SSP0_MOSI/	O	1	<b>EMC_CKE3</b> — SDRAM clock enable 3.
T3_MAT1	I/O	2	<b>SSP0_MOSI</b> — Master Out Slave In for SSP0.
	O	3	<b>T3_MAT1</b> — Match output for Timer 3, channel 1.
P2[28]/ EMC_DQM0	I/O	0	<b>P2[28]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_DQM0</b> — Data mask 0 used with SDRAM and static devices.
P2[29]/ EMC_DQM1	I/O	0	<b>P2[29]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_DQM1</b> — Data mask 1 used with SDRAM and static devices.
P2[30]/ EMC_DQM2/	I/O	0	<b>P2[30]</b> — General purpose digital input/output pin.
I2C2_SDA/	O	1	<b>EMC_DQM2</b> — Data mask 2 used with SDRAM and static devices.
T3_MAT2	I/O	2	<b>I2C2_SDA</b> — I <sup>2</sup> C2 data input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	O	3	<b>T3_MAT2</b> — Match output for Timer 3, channel 2.
P2[31]/ EMC_DQM3/	I/O	0	<b>P2[31]</b> — General purpose digital input/output pin.
I2C2_SCL/	O	1	<b>EMC_DQM3</b> — Data mask 3 used with SDRAM and static devices.
T3_MAT3	I/O	2	<b>I2C2_SCL</b> — I <sup>2</sup> C2 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	O	3	<b>T3_MAT3</b> — Match output for Timer 3, channel 3.
<b>P3[0] to P3[31]</b>	I/O		<b>Port 3:</b> Port 3 provides up to 32 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.
P3[0]/ EMC_D[0]	I/O	0	<b>P3[0]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[0]</b> — External memory data line 0.
P3[1]/ EMC_D[1]	I/O	0	<b>P3[1]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[1]</b> — External memory data line 1.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P3[2]/ EMC_D[2]	I/O	0	<b>P3[2]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[2]</b> — External memory data line 2.
P3[3]/ EMC_D[3]	I/O	0	<b>P3[3]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[3]</b> — External memory data line 3.
P3[4]/ EMC_D[4]	I/O	0	<b>P3[4]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[4]</b> — External memory data line 4.
P3[5]/ EMC_D[5]	I/O	0	<b>P3[5]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[5]</b> — External memory data line 5.
P3[6]/ EMC_D[6]	I/O	0	<b>P3[6]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[6]</b> — External memory data line 6.
P3[7]/ EMC_D[7]	I/O	0	<b>P3[7]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[7]</b> — External memory data line 7.
P3[8]/ EMC_D[8]	I/O	0	<b>P3[8]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[8]</b> — External memory data line 8.
P3[9]/ EMC_D[9]	I/O	0	<b>P3[9]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[9]</b> — External memory data line 9.
P3[10]/ EMC_D[10]	I/O	0	<b>P3[10]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[10]</b> — External memory data line 10.
P3[11]/ EMC_D[11]	I/O	0	<b>P3[11]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[11]</b> — External memory data line 11.
P3[12]/ EMC_D[12]	I/O	0	<b>P3[12]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[12]</b> — External memory data line 12.
P3[13]/ EMC_D[13]	I/O	0	<b>P3[13]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[13]</b> — External memory data line 13.
P3[14]/ EMC_D[14]	I/O	0	<b>P3[14]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[14]</b> — External memory data line 14. On POR, this pin serves as the BOOT0 pin (see P3[15] description below).
P3[15]/ EMC_D[15]	I/O	0	<b>P3[15]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[15]</b> — External memory data line 15. On POR, this pin serves as the BOOT1 pin (flashless parts only). BOOT[1:0] = 00 selects 8-bit external memory on $\overline{\text{EMC\_CS1}}$ . BOOT[1:0] = 01 is reserved. Do not use. BOOT[1:0] = 10 selects 32-bit external memory on $\overline{\text{EMC\_CS1}}$ . BOOT[1:0] = 11 selects 16-bit external memory on $\overline{\text{EMC\_CS1}}$ .
P3[16]/ EMC_D[16]/ PWM0[1]/ U1_TXD	I/O	0	<b>P3[16]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[16]</b> — External memory data line 16.
	O	2	<b>PWM0[1]</b> — Pulse Width Modulator 0, output 1.
	O	3	<b>U1_TXD</b> — Transmitter output for UART 1.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P3[17]/ EMC_D[17]/ PWM0[2]/ U1_RXD	I/O	0	<b>P3[17]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[17]</b> — External memory data line 17.
	O	2	<b>PWM0[2]</b> — Pulse Width Modulator 0, output 2.
	I	3	<b>U1_RXD</b> — Receiver input for UART 1.
P3[18]/ EMC_D[18]/ PWM0[3]/ U1_CTS	I/O	0	<b>P3[18]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[18]</b> — External memory data line 18.
	O	2	<b>PWM0[3]</b> — Pulse Width Modulator 0, output 3.
	I	3	<b>U1_CTS</b> — Clear to Send input for UART 1.
P3[19]/ EMC_D[19]/ PWM0[4]/ U1_DCD	I/O	0	<b>P3[19]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[19]</b> — External memory data line 19.
	O	2	<b>PWM0[4]</b> — Pulse Width Modulator 0, output 4.
	I	3	<b>U1_DCD</b> — Data Carrier Detect input for UART 1.
P3[20]/ EMC_D[20]/ PWM0[5]/ U1_DSR	I/O	0	<b>P3[20]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[20]</b> — External memory data line 20.
	O	2	<b>PWM0[5]</b> — Pulse Width Modulator 0, output 5.
	I	3	<b>U1_DSR</b> — Data Set Ready input for UART 1.
P3[21]/ EMC_D[21]/ PWM0[6]/ U1_DTR	I/O	0	<b>P3[21]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[21]</b> — External memory data line 21.
	O	2	<b>PWM0[6]</b> — Pulse Width Modulator 0, output 6.
	O	3	<b>U1_DTR</b> — Data Terminal Ready output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
P3[22]/ EMC_D[22]/ PWM0_CAP0/ U1_RI	I/O	0	<b>P3[22]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[22]</b> — External memory data line 22.
	I	2	<b>PWM0_CAP0</b> — Capture input for PWM0, channel 0.
	I	3	<b>U1_RI</b> — Ring Indicator input for UART 1.
P3[23]/ EMC_D[23]/ PWM1_CAP0/ T0_CAP0	I/O	0	<b>P3[23]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[23]</b> — External memory data line 23.
	I	2	<b>PWM1_CAP0</b> — Capture input for PWM1, channel 0.
	I	3	<b>T0_CAP0</b> — Capture input for Timer 0, channel 0.
P3[24]/ EMC_D[24]/ PWM1[1]/ T0_CAP1	I/O	0	<b>P3[24]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[24]</b> — External memory data line 24.
	O	2	<b>PWM1[1]</b> — Pulse Width Modulator 1, output 1.
	I	3	<b>T0_CAP1</b> — Capture input for Timer 0, channel 1.
P3[25]/ EMC_D[25]/ PWM1[2]/ T0_MAT0	I/O	0	<b>P3[25]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[25]</b> — External memory data line 25.
	O	2	<b>PWM1[2]</b> — Pulse Width Modulator 1, output 2.
	O	3	<b>T0_MAT0</b> — Match output for Timer 0, channel 0.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P3[26]/ EMC_D[26]/ PWM1[3]/ T0_MAT1/ STCLK	I/O	0	<b>P3[26]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[26]</b> — External memory data line 26.
	O	2	<b>PWM1[3]</b> — Pulse Width Modulator 1, output 3.
	O	3	<b>T0_MAT1</b> — Match output for Timer 0, channel 1.
	I	4	<b>STCLK</b> — System tick timer clock input.
P3[27]/ EMC_D[27]/ PWM1[4]/ T1_CAP0	I/O	0	<b>P3[27]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[27]</b> — External memory data line 27.
	O	2	<b>PWM1[4]</b> — Pulse Width Modulator 1, output 4.
	I	3	<b>T1_CAP0</b> — Capture input for Timer 1, channel 0.
P3[28]/ EMC_D[28]/ PWM1[5]/ T1_CAP1	I/O	0	<b>P3[28]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[28]</b> — External memory data line 28.
	O	2	<b>PWM1[5]</b> — Pulse Width Modulator 1, output 5.
	I	3	<b>T1_CAP1</b> — Capture input for Timer 1, channel 1.
P3[29]/ EMC_D[29]/ PWM1[6]/ T1_MAT0	I/O	0	<b>P3[29]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[29]</b> — External memory data line 29.
	O	2	<b>PWM1[6]</b> — Pulse Width Modulator 1, output 6.
	O	3	<b>T1_MAT0</b> — Match output for Timer 1, channel 0.
P3[30]/ EMC_D[30]/ U1_RTS/ T1_MAT1	I/O	0	<b>P3[30]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[30]</b> — External memory data line 30.
	O	2	<b>U1_RTS</b> — Request to Send output for UART 1. Can also be configured to be an RS-485/EIA-485 output enable signal for UART 1.
	O	3	<b>T1_MAT1</b> — Match output for Timer 1, channel 1.
P3[31]/ EMC_D[31]/ T1_MAT2	I/O	0	<b>P3[31]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_D[31]</b> — External memory data line 31.
	O	3	<b>T1_MAT2</b> — Match output for Timer 1, channel 2.
<b>P4[0] to P4[31]</b>	I/O		<b>Port 4:</b> Port 4 provides up to 32 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.
P4[0]/ EMC_A[0]	I/O	0	<b>P4[0]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[0]</b> — External memory address line 0.
P4[1]/ EMC_A[1]	I/O	0	<b>P4[1]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[1]</b> — External memory address line 1.
P4[2]/ EMC_A[2]	I/O	0	<b>P4[2]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[2]</b> — External memory address line 2.
P4[3]/ EMC_A[3]	I/O	0	<b>P4[3]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[3]</b> — External memory address line 3.
P4[4]/ EMC_A[4]	I/O	0	<b>P4[4]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[4]</b> — External memory address line 4.
P4[5]/ EMC_A[5]	I/O	0	<b>P4[5]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[5]</b> — External memory address line 5.
P4[6]/ EMC_A[6]	I/O	0	<b>P4[6]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[6]</b> — External memory address line 6.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P4[7]/ EMC_A[7]	I/O	0	<b>P4[7]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[7]</b> — External memory address line 7.
P4[8]/ EMC_A[8]	I/O	0	<b>P4[8]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[8]</b> — External memory address line 8.
P4[9]/ EMC_A[9]	I/O	0	<b>P4[9]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[9]</b> — External memory address line 9.
P4[10]/ EMC_A[10]	I/O	0	<b>P4[10]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[10]</b> — External memory address line 10.
P4[11]/ EMC_A[11]	I/O	0	<b>P4[11]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[11]</b> — External memory address line 11.
P4[12]/ EMC_A[12]	I/O	0	<b>P4[12]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[12]</b> — External memory address line 12.
P4[13]/ EMC_A[13]	I/O	0	<b>P4[13]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[13]</b> — External memory address line 13.
P4[14]/ EMC_A[14]	I/O	0	<b>P4[14]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[14]</b> — External memory address line 14.
P4[15]/ EMC_A[15]	I/O	0	<b>P4[15]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[15]</b> — External memory address line 15.
P4[16]/ EMC_A[16]	I/O	0	<b>P4[16]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[16]</b> — External memory address line 16.
P4[17]/ EMC_A[17]	I/O	0	<b>P4[17]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[17]</b> — External memory address line 17.
P4[18]/ EMC_A[18]	I/O	0	<b>P4[18]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[18]</b> — External memory address line 18.
P4[19]/ EMC_A[19]	I/O	0	<b>P4[19]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[19]</b> — External memory address line 19.
P4[20]/ EMC_A[20]/ I2C2_SDA/ SSP1_SCK	I/O	0	<b>P4[20]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[20]</b> — External memory address line 20.
	I/O	2	<b>I2C2_SDA</b> — I <sup>2</sup> C2 data input/output ((this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I/O	3	<b>SSP1_SCK</b> — Serial Clock for SSP1.
P4[21]/ EMC_A[21]/ I2C2_SCL/ SSP1_SSEL	I/O	0	<b>P4[21]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[21]</b> — External memory address line 21.
	I/O	2	<b>I2C2_SCL</b> — I <sup>2</sup> C2 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	I/O	3	<b>SSP1_SSEL</b> — Slave Select for SSP1.
P4[22]/ EMC_A[22]/ U2_TXD/ SSP1_MISO	I/O	0	<b>P4[22]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[22]</b> — External memory address line 22.
	O	2	<b>U2_TXD</b> — Transmitter output for UART 2.
	I/O	3	<b>SSP1_MISO</b> — Master In Slave Out for SSP1.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>1</sup>	Description
P4[23]/ EMC_A[23]/ U2_RXD/ SSP1_MOSI	I/O	0	<b>P4[23]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[23]</b> — External memory address line 23.
	I	2	<b>U2_RXD</b> — Receiver input for UART 2.
	I/O	3	<b>SSP1_MOSI</b> — Master Out Slave In for SSP1.
P4[24]/ EMC_OE	I/O	0	<b>P4[24]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_OE</b> — LOW active Output Enable signal.
P4[25]/ EMC_WE	I/O	0	<b>P4[25]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_WE</b> — LOW active Write Enable signal.
P4[26]/ EMC_BLS0	I/O	0	<b>P4[26]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_BLS0</b> — LOW active Byte Lane select signal 0.
P4[27]/ EMC_BLS1	I/O	0	<b>P4[27]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_BLS1</b> — LOW active Byte Lane select signal 1.
P4[28]/ EMC_BLS2/ U3_TXD/ T2_MAT0/ LCD_VD[6]/ LCD_VD[10]/ LCD_VD[2]	I/O	0	<b>P4 [28]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_BLS2</b> — LOW active Byte Lane select signal 2.
	O	2	<b>TXD3</b> — Transmitter output for UART 3.
	O	3	<b>T2_MAT0</b> — Match output for Timer 2, channel 0.
	O	5	<b>LCD_VD[6]</b> — LCD data.
	O	6	<b>LCD_VD[10]</b> — LCD data.
	O	7	<b>LCD_VD[2]</b> — LCD data.
P4[29]/ EMC_BLS3/ U3_RXD/ T2_MAT1/ I2C2_SCL/ LCD_VD[7]/ LCD_VD[11]/ LCD_VD[3]	I/O	0	<b>P4[29]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_BLS3</b> — LOW active Byte Lane select signal 3.
	I	2	<b>U3_RXD</b> — Receiver input for UART 3.
	O	3	<b>T2_MAT1</b> — Match output for Timer 2, channel 1.
	I/O	4	<b>I2C2_SCL</b> — I <sup>2</sup> C2 clock input/output (this pin does not use a specialized I <sup>2</sup> C pad, see <a href="#">Section 22.1</a> for details).
	O	5	<b>LCD_VD[7]</b> — LCD data.
	O	6	<b>LCD_VD[11]</b> — LCD data.
	O	7	<b>LCD_VD[3]</b> — LCD data.
P4[30]/ EMC_CS0	I/O	0	<b>P4[30]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CS0</b> — LOW active Chip Select 0 signal.
P4[31]/ EMC_CS1	I/O	0	<b>P4[31]</b> — General purpose digital input/output pin.
	O	1	<b>EMC_CS1</b> — LOW active Chip Select 1 signal.
<b>P5[0] to P5[4]</b>	I/O		<b>Port 5:</b> Port 5 provides up to 5 I/O pins, depending on the package. Each pin has individual direction control, pin mode configuration, and function selection.
P5[0]/ EMC_A[24]/ T2_MAT2	I/O	0	<b>P5[0]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[24]</b> — External memory address line 24.
	O	3	<b>T2_MAT2</b> — Match output for Timer 2, channel 2.
P5[1]/ EMC_A[25]/ T2_MAT3	I/O	0	<b>P5[1]</b> — General purpose digital input/output pin.
	I/O	1	<b>EMC_A[25]</b> — External memory address line 25.
	O	3	<b>T2_MAT3</b> — Match output for Timer 2, channel 3.

Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>[1]</sup>	Description
P5[2]/ T3_MAT2/ I2C0_SDA	I/O	0	<b>P5[2]</b> — General purpose digital input/output pin.
	O	3	<b>T3_MAT2</b> — Match output for Timer 3, channel 2.
	I/O	5	<b>I2C0_SDA</b> — I <sup>2</sup> C0 data input/output (this pin uses a specialized I <sup>2</sup> C pad that supports I <sup>2</sup> C Fast Mode Plus).
P5[3]/ U4_RXD/ I2C0_SCL	I/O	0	<b>P5[3]</b> — General purpose digital input/output pin.
	I	4	<b>U4_RXD</b> — Receiver input for UART 4.
	I/O	5	<b>I2C0_SCL0</b> — I <sup>2</sup> C0 clock input/output (this pin uses a specialized I <sup>2</sup> C pad that supports I <sup>2</sup> C Fast Mode Plus).
P5[4]/ U0_OE/ T3_MAT3/ U4_TXD	I/O	0	<b>P5[4]</b> — General purpose digital input/output pin.
	O	1	<b>U0_OE</b> — RS-485/EIA-485 output enable signal for UART 0.
	O	3	<b>T3_MAT3</b> — Match output for Timer 3, channel 3.
	O	4	<b>U4_TXD</b> — Transmitter output for UART 4 (input/output in smart card mode).
RTC_ALARM	O		<b>RTC_ALARM</b> — RTC controlled output. This pin has a low drive strength and is powered by V <sub>BAT</sub> (see data sheet for details). It is driven LOW when a RTC alarm is generated.
USB_D-2	I/O		<b>USB_D-2</b> — USB port 2 bidirectional D- line.
JTAG_TDO (SWO)	O		<b>JTAG_TDO</b> — Test Data Out for JTAG interface. <b>SWO</b> — Serial wire trace output.
JTAG_TDI	I		<b>TDI</b> — Test Data In for JTAG interface. This pin includes an internal pull-up, see <a href="#">Section 37.1</a> .
JTAG_TMS (SWDIO)	I		<b>TMS</b> — Test Mode Select for JTAG interface. This pin includes an internal pull-up, see <a href="#">Section 37.1</a> . <b>SWDIO</b> — Serial wire debug data input/output.
JTAG_TRST	I		<b>TRST</b> — Test Reset for JTAG interface. This pin includes an internal pull-up, see <a href="#">Section 37.1</a> .
JTAG_TCK (SWDCLK)	I		<b>TCK</b> — Test Clock for JTAG interface. This clock must be slower than 1/6 of the CPU clock (CCLK) for the JTAG interface to operate. <b>SWDCLK</b> — Serial wire clock.
RSTOUT	O		Reset status output. A LOW output on this pin indicates that the device is in the reset state, for any reason. This reflects the <b>RESET</b> input pin and all internal reset sources.
RESET	I		External reset input. A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. This pin includes a 20 ns input glitch filter.
XTAL1 <sup>[2]</sup>	I		Input to the oscillator circuit and internal clock generator circuits.
XTAL2 <sup>[2]</sup>	O		Output from the oscillator amplifier.
RTCX1 <sup>[2]</sup>	I		Input to the RTC 32 kHz ultra-low power oscillator circuit.
RTCX2 <sup>[2]</sup>	O		Output from the RTC 32 kHz ultra-low power oscillator circuit.
V <sub>SS</sub> <sup>[2]</sup>	I		<b>ground:</b> 0 V reference for digital IO pins.
V <sub>SSREG</sub> <sup>[2]</sup>	I		<b>ground:</b> 0 V reference for internal logic.
V <sub>SSA</sub> <sup>[2]</sup>	I		<b>analog ground:</b> 0 V power supply and reference for the ADC and DAC. This should be the same voltage as V <sub>SS</sub> , but should be isolated to minimize noise and error.
V <sub>DD(3V3)</sub> <sup>[2]</sup>	I		<b>3.3 V supply voltage:</b> This is the power supply voltage for I/O other than pins in the Vbat domain.



Table 65. LPC178x/177x pin description ...continued

Symbol	Type	IOCON select <sup>[1]</sup>	Description
V <sub>DD(REG)(3V3)</sub> <sup>[2]</sup>	I		<b>3.3 V regulator supply voltage:</b> This is the power supply for the on-chip voltage regulator that supplies internal logic.
V <sub>DDA</sub> <sup>[2]</sup>	I		<b>analog 3.3 V pad supply voltage:</b> This can be connected to the same supply as V <sub>DD(3V3)</sub> but should be isolated to minimize noise and error. This voltage is used to power the ADC and DAC. <b>Note: this pin should be tied to 3.3V if the ADC and DAC are not used.</b>
VREFP <sup>[2]</sup>	I		<b>ADC positive reference voltage:</b> This should be the same voltage as V <sub>DDA</sub> , but should be isolated to minimize noise and error. The voltage level on this pin is used as a reference for ADC and DAC. <b>Note: this pin should be tied to 3.3V if the ADC and DAC are not used.</b>
VBAT <sup>[2]</sup>	I		<b>RTC power supply:</b> 3.3 V on this pin supplies power to the RTC.

[1] These values are used in the FUNC field of the IOCON registers, described in [Section 8.4.1](#).

[2] These pins provide special analog functionality.

## 7.2 Boot control

Flashless LPC178x/177x devices use pins P3[15]/D15 and P3[14]/D14 for configuring the external memory bus during the boot process. These pins are sampled during Power-on Reset (POR). See [Table 66](#) for possible settings of the boot control pins.

**Table 66. Boot control on pins P3[15]/D15 and P3[14]/D14**

P3[15]/D15 BOOT1	P3[14]/D14 BOOT0	Description
0	0	Boot from 8-bit external memory on $\overline{\text{EMC\_CS1}}$ . Sampled on POR signal.
0	1	Reserved. Do not use.
1	0	Boot from 32-bit external memory on $\overline{\text{EMC\_CS1}}$ . Sampled on POR signal.
1	1	Boot from 16-bit external memory on $\overline{\text{EMC\_CS1}}$ . Sampled on POR signal.

During the boot process, external memory banks 0 and 1 are configured with the same data bus width determined by the setting of the two boot pins P3[15]/D15 and P3[14]/D14. Unused address pins (A0 when booting from 16-bit wide external memory, A1 and A0 when booting from 32-bit wide memory) are configured as GPIO.

The boot loader will branch to external user code at address 0x8000 0000. This code must set up an interrupt vector table (see [Section 2.4](#)). The external boot memory must be connected to static chip select 1 ( $\overline{\text{EMC\_CS1}}$ ). Note that  $\overline{\text{EMC\_CS1}}$  is mirrored onto  $\overline{\text{EMC\_CS0}}$  because the address mirror bit is set in the EMCControl register during POR, see [Table 112](#). Therefore, the user code residing in the external boot memory must be linked to execute from address location 0x8000 0000 (EMC bank 0 address).

**Remark:** The external boot option is supported only for flashless devices.

# LPC178x/7x

## Chapter 8: LPC178x/7x I/O configuration

Rev. 01.03 — 7 March 2011

User manual

### 8.1 How to read this chapter

The LPC178x/177x provides a separate register to configure each GPIO pin. This configuration includes which internal function is connected to the pin, the output mode (plain, pull-up, pull-down, or repeater), open drain mode control, hysteresis enable, slew rate control, and buffer setup for analog functions. Some pins include additional special controls, such as for I<sup>2</sup>C buffer modes. These registers are summarized in [Table 67](#).

**Table 67. Summary of I/O pin configuration registers**

Port	Registers	Detail Table
Port 0 pins	IOCON_P0_nn, where nn is the port pin number, from 0 to 31 <sup>[1]</sup>	<a href="#">Table 68</a>
Port 1 pins	IOCON_P1_nn, where nn is the port pin number, from 0 to 31 <sup>[1]</sup>	<a href="#">Table 69</a>
Port 2 pins	IOCON_P2_nn, where nn is the port pin number, from 0 to 31 <sup>[1]</sup>	<a href="#">Table 70</a>
Port 3 pins	IOCON_P3_nn, where nn is the port pin number, from 0 to 31 <sup>[1]</sup>	<a href="#">Table 71</a>
Port 4 pins	IOCON_P4_nn, where nn is the port pin number, from 0 to 31 <sup>[1]</sup>	<a href="#">Table 72</a>
Port 5 pins	IOCON_P5_nn, where nn is the port pin number, from 0 to 4 <sup>[1]</sup>	<a href="#">Table 73</a>

[1] Which pins are available depends on the part number and package combination.

### 8.2 Description

The pin connect block allows most pins of the microcontroller to have more than one potential function. Configuration registers control the multiplexers to allow connection between the pin and the on-chip peripherals.

Peripherals should be connected to the appropriate pins prior to being activated and prior to any related interrupt(s) being enabled. Activity of any enabled peripheral function that is not mapped to a related pin should be considered undefined.

Selection of a single function on a port pin excludes other peripheral functions available on the same pin. However, the GPIO input stays connected and may be read by software or used to contribute to the GPIO interrupt feature.

### 8.3 IOCON registers

The IOCON registers control the functions of device pins. Each GPIO pin has a dedicated control register to select its function and characteristics. Each pin has a unique set of functional capabilities. Not all pin characteristics are selectable on all pins. For instance, pins that have an I<sup>2</sup>C function can be configured for different I<sup>2</sup>C-bus modes, while pins that have an analog alternate function have an analog mode can be selected. Details of the IOCON registers are in [Section 8.4.1](#). The following sections describe specific characteristics of pins.

Multiple connections

Since a particular peripheral function may be allowed on more than one pin, it is possible to configure more than one pin to perform the same function. If a peripheral output function is configured to appear on more than one pin, it will in fact be routed to those pins. If a peripheral input function is defined as coming from more than one source, the values will be logically combined, possibly resulting in incorrect peripheral operation. Therefore care should be taken to avoid this situation.

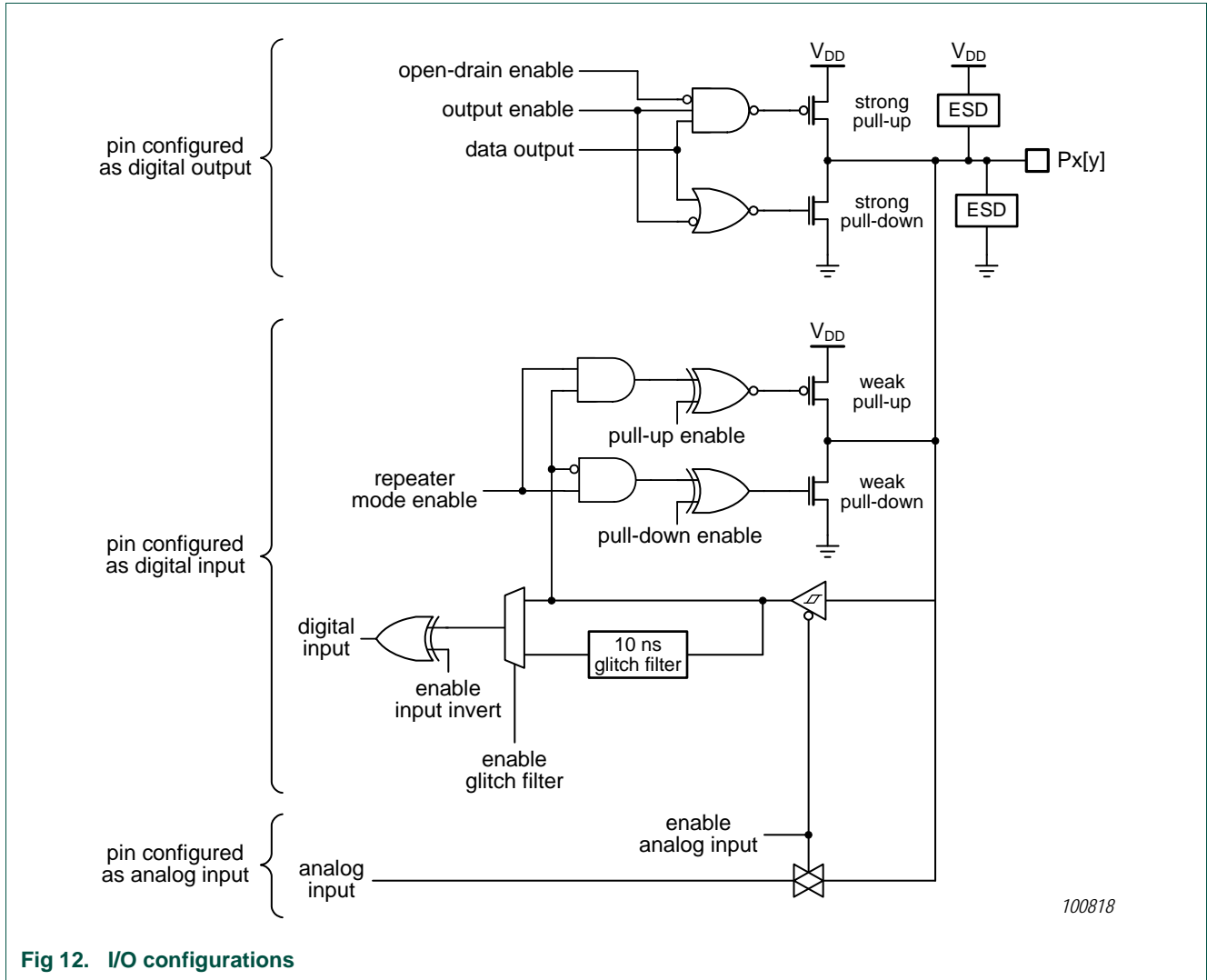


Fig 12. I/O configurations

8.3.1 Pin function

The FUNC bits in the IOCON registers can be set to GPIO (typically value 000) or to a special function. For pins set to GPIO, the FIONDIR registers determine whether the pin is configured as an input or output (see Section 9.5.1). For any special function, the pin direction is controlled automatically depending on the function. The FIONDIR registers have no effect for special functions.

### 8.3.2 Pin mode

The MODE bits in the IOCON register allow the selection of on-chip pull-up or pull-down resistors for each pin or select the repeater mode.

The possible on-chip resistor configurations are pull-up enabled, pull-down enabled, or no pull-up/pull-down. The default value is pull-up enabled.

The repeater mode enables the pull-up resistor if the pin is high and enables the pull-down resistor if the pin is low. This causes the pin to retain its last known state if it is configured as an input and is not driven externally. Such state retention is not applicable to the Deep Power-down mode. Repeater mode may typically be used to prevent a pin from floating (and potentially using significant power if it floats to an indeterminate state) if it is temporarily not driven.

### 8.3.3 Hysteresis

The input buffer for digital functions can be configured with or without hysteresis. See the appropriate specific device data sheet for quantitative details.

### 8.3.4 Input Inversion

This option is included to save users from having to include an external inverter on an input that is only available in the opposite polarity from an external source. Do not set this option on a GPIO output. Doing so can result in inadvertent toggling of an output with input inversion selected, as a result of operations on other pins in the same port. For example, if software reads a GPIO Port register, modifies other bits/outputs in the value, and writes the result back to the Port register, any output in the port that has input inversion selected will change state.

### 8.3.5 Analog/digital mode

In Analog mode, the analog input connection is enabled. In digital mode, the analog input connection is disabled. This protects the analog input from voltages outside the range of the analog power supply and reference that may sometimes be present on digital pins, since they are typically 5V tolerant.

If Analog mode is selected, the MODE field should be “Inactive” (00); the HYS, INV, FILTR, SLEW, and OD settings have no effect. For an unconnected pin that has an analog function, keep the ADMODE bit set to 1 (digital mode), and pull-up or pull-down mode selected in the MODE field.

### 8.3.6 Input filter

Type A and W pins include a filter that can be selectively enabled. The filter suppresses input pulses smaller than about 10 ns.

### 8.3.7 Output slew rate

The SLEW bits of digital outputs that do not need to switch state very quickly should be set to “standard”. This setting allows multiple outputs to switch simultaneously without noticeably degrading the power/ground distribution of the device, and has only a small effect on signal transition time. This is particularly important if analog accuracy is significant to the application. See the relevant specific device data sheet for more details.

### 8.3.8 I<sup>2</sup>C modes

Pins that support I<sup>2</sup>C with specialized pad electronics (P0[27], P0[28], P5[2], and P5[3]) have additional configuration bits. These are not hardwired so that the pins can be more easily used for non-I<sup>2</sup>C functions.

The HS bit applies to standard, Fast-mode, and Fast-mode Plus I<sup>2</sup>C, and is available for all the pins noted above.

The HIDRIVE bit applies only to pins P5[2] and P5[3], and is used to select between Standard mode and Fast-mode I<sup>2</sup>C or Fast-mode Plus I<sup>2</sup>C.

- For any I<sup>2</sup>C mode, clear the HS bit so that the input glitch filter is enabled. Clear the HIDRIVE bit if it exists for that pin to select the correct drive strength for Standard mode or Fast-mode I<sup>2</sup>C
- For Fast-mode Plus I<sup>2</sup>C operation, set the HIDRIVE bit to select the correct drive strength for Fast-mode Plus I<sup>2</sup>C.
- For non-I<sup>2</sup>C operation, these pins remain open-drain and can only drive low, regardless of how HS and HIDRIVE are set. They would typically be used with an external pull-up resistor if they are used as outputs unless they are used only to sink current. Leave HS = 1 and HIDRIVE = 0 (if applicable) to maximize compatibility with other GPIO pins.

### 8.3.9 Open-Drain Mode

When output is selected, either by selecting a special function in the FUNC field, or by selecting the GPIO function for a pin having a 1 in its FIONDIR register, a 1 in the OD bit selects open-drain operation, that is, a 1 disables the high-drive transistor. This option has no effect on the primary I<sup>2</sup>C pins. Note that the properties of a pin in this simulated open-drain mode are somewhat different than those of a true open drain output.

### 8.3.10 DAC enable

The pin that includes the DAC output as a potential function includes an enable for the function that must be set if the DAC output is used.

## 8.4 Register description

The pin connect block contains an I/O Control register (IOCON) for each pin that has programmable attributes, and selects peripheral functions associated with that pin. These registers are shown by GPIO port number in Tables 8–68 through 8–73.

**Table 68. I/O Control registers for port 0**

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type <sup>[2]</sup>	Available on packages
P0[0]	IOCON_P0_00	R/W	0x030	0x4002 C000	D (tables 74, 75)	All
P0[1]	IOCON_P0_01	R/W	0x030	0x4002 C004	D (tables 74, 75)	All
P0[2]	IOCON_P0_02	R/W	0x030	0x4002 C008	D (tables 74, 75)	All
P0[3]	IOCON_P0_03	R/W	0x030	0x4002 C00C	D (tables 74, 75)	All
P0[4]	IOCON_P0_04	R/W	0x030	0x4002 C010	D (tables 74, 75)	All
P0[5]	IOCON_P0_05	R/W	0x030	0x4002 C014	D (tables 74, 75)	All
P0[6]	IOCON_P0_06	R/W	0x030	0x4002 C018	D (tables 74, 75)	All
P0[7]	IOCON_P0_07	R/W	0x0A0	0x4002 C01C	W (tables 82, 83)	All
P0[8]	IOCON_P0_08	R/W	0x0A0	0x4002 C020	W (tables 82, 83)	All
P0[9]	IOCON_P0_09	R/W	0x0A0	0x4002 C024	W (tables 82, 83)	All
P0[10]	IOCON_P0_10	R/W	0x030	0x4002 C028	D (tables 74, 75)	All
P0[11]	IOCON_P0_11	R/W	0x030	0x4002 C02C	D (tables 74, 75)	All
P0[12]	IOCON_P0_12	R/W	0x1B0	0x4002 C030	A (tables 76, 77)	144, 180, 208
P0[13]	IOCON_P0_13	R/W	0x1B0	0x4002 C034	A (tables 76, 77)	144, 180, 208
P0[14]	IOCON_P0_14	R/W	0x030	0x4002 C038	D (tables 74, 75)	144, 180, 208
P0[15]	IOCON_P0_15	R/W	0x030	0x4002 C03C	D (tables 74, 75)	All
P0[16]	IOCON_P0_16	R/W	0x030	0x4002 C040	D (tables 74, 75)	All
P0[17]	IOCON_P0_17	R/W	0x030	0x4002 C044	D (tables 74, 75)	All
P0[18]	IOCON_P0_18	R/W	0x030	0x4002 C048	D (tables 74, 75)	All
P0[19]	IOCON_P0_19	R/W	0x030	0x4002 C04C	D (tables 74, 75)	All
P0[20]	IOCON_P0_20	R/W	0x030	0x4002 C050	D (tables 74, 75)	All
P0[21]	IOCON_P0_21	R/W	0x030	0x4002 C054	D (tables 74, 75)	All
P0[22]	IOCON_P0_22	R/W	0x030	0x4002 C058	D (tables 74, 75)	All
P0[23]	IOCON_P0_23	R/W	0x1B0	0x4002 C05C	A (tables 76, 77)	All
P0[24]	IOCON_P0_24	R/W	0x1B0	0x4002 C060	A (tables 76, 77)	All
P0[25]	IOCON_P0_25	R/W	0x1B0	0x4002 C064	A (tables 76, 77)	All
P0[26]	IOCON_P0_26	R/W	0x1B0	0x4002 C068	A (tables 76, 77)	All
P0[27]	IOCON_P0_27	R/W	0	0x4002 C06C	I (tables 80, 81)	All
P0[28]	IOCON_P0_28	R/W	0	0x4002 C070	I (tables 80, 81)	All
P0[29]	IOCON_P0_29	R/W	0	0x4002 C074	U (tables 78, 79)	All
P0[30]	IOCON_P0_30	R/W	0	0x4002 C078	U (tables 78, 79)	All
P0[31]	IOCON_P0_31	R/W	0	0x4002 C07C	U (tables 78, 79)	144, 180, 208

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

[2] IOCON types are D (standard digital pin), and other pins with a specialized function: A (analog), U (USB), I (I2C), and W.

Table 69. I/O Control registers for port 1

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type	Available on packages
P1[0]	IOCON_P1_00	R/W	0x030	0x4002 C080	D (tables 74, 75)	All
P1[1]	IOCON_P1_01	R/W	0x030	0x4002 C084	D (tables 74, 75)	All
P1[2]	IOCON_P1_02	R/W	0x030	0x4002 C088	D (tables 74, 75)	180, 208
P1[3]	IOCON_P1_03	R/W	0x030	0x4002 C08C	D (tables 74, 75)	180, 208
P1[4]	IOCON_P1_04	R/W	0x030	0x4002 C090	D (tables 74, 75)	All
P1[5]	IOCON_P1_05	R/W	0x030	0x4002 C094	D (tables 74, 75)	180, 208
P1[6]	IOCON_P1_06	R/W	0x030	0x4002 C098	D (tables 74, 75)	180, 208
P1[7]	IOCON_P1_07	R/W	0x030	0x4002 C09C	D (tables 74, 75)	180, 208
P1[8]	IOCON_P1_08	R/W	0x030	0x4002 C0A0	D (tables 74, 75)	All
P1[9]	IOCON_P1_09	R/W	0x030	0x4002 C0A4	D (tables 74, 75)	All
P1[10]	IOCON_P1_10	R/W	0x030	0x4002 C0A8	D (tables 74, 75)	All
P1[11]	IOCON_P1_11	R/W	0x030	0x4002 C0AC	D (tables 74, 75)	180, 208
P1[12]	IOCON_P1_12	R/W	0x030	0x4002 C0B0	D (tables 74, 75)	180, 208
P1[13]	IOCON_P1_13	R/W	0x030	0x4002 C0B4	D (tables 74, 75)	180, 208
P1[14]	IOCON_P1_14	R/W	0x030	0x4002 C0B8	D (tables 74, 75)	All
P1[15]	IOCON_P1_15	R/W	0x030	0x4002 C0BC	D (tables 74, 75)	All
P1[16]	IOCON_P1_16	R/W	0x030	0x4002 C0C0	D (tables 74, 75)	All
P1[17]	IOCON_P1_17	R/W	0x030	0x4002 C0C4	D (tables 74, 75)	All
P1[18]	IOCON_P1_18	R/W	0x030	0x4002 C0C8	D (tables 74, 75)	All
P1[19]	IOCON_P1_19	R/W	0x030	0x4002 C0CC	D (tables 74, 75)	All
P1[20]	IOCON_P1_20	R/W	0x030	0x4002 C0D0	D (tables 74, 75)	All
P1[21]	IOCON_P1_21	R/W	0x030	0x4002 C0D4	D (tables 74, 75)	All
P1[22]	IOCON_P1_22	R/W	0x030	0x4002 C0D8	D (tables 74, 75)	All
P1[23]	IOCON_P1_23	R/W	0x030	0x4002 C0DC	D (tables 74, 75)	All
P1[24]	IOCON_P1_24	R/W	0x030	0x4002 C0E0	D (tables 74, 75)	All
P1[25]	IOCON_P1_25	R/W	0x030	0x4002 C0E4	D (tables 74, 75)	All
P1[26]	IOCON_P1_26	R/W	0x030	0x4002 C0E8	D (tables 74, 75)	All
P1[27]	IOCON_P1_27	R/W	0x030	0x4002 C0EC	D (tables 74, 75)	All
P1[28]	IOCON_P1_28	R/W	0x030	0x4002 C0F0	D (tables 74, 75)	All
P1[29]	IOCON_P1_29	R/W	0x030	0x4002 C0F4	D (tables 74, 75)	All
P1[30]	IOCON_P1_30	R/W	0x1B0	0x4002 C0F8	A (tables 76, 77)	All
P1[31]	IOCON_P1_31	R/W	0x1B0	0x4002 C0FC	A (tables 76, 77)	All

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.



Table 70. I/O Control registers for port 2

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type	Available on packages
P2[0]	IOCON_P2_00	R/W	0x030	0x4002 C100	D (tables 74, 75)	All
P2[1]	IOCON_P2_01	R/W	0x030	0x4002 C104	D (tables 74, 75)	All
P2[2]	IOCON_P2_02	R/W	0x030	0x4002 C108	D (tables 74, 75)	All
P2[3]	IOCON_P2_03	R/W	0x030	0x4002 C10C	D (tables 74, 75)	All
P2[4]	IOCON_P2_04	R/W	0x030	0x4002 C110	D (tables 74, 75)	All
P2[5]	IOCON_P2_05	R/W	0x030	0x4002 C114	D (tables 74, 75)	All
P2[6]	IOCON_P2_06	R/W	0x030	0x4002 C118	D (tables 74, 75)	All
P2[7]	IOCON_P2_07	R/W	0x030	0x4002 C11C	D (tables 74, 75)	All
P2[8]	IOCON_P2_08	R/W	0x030	0x4002 C120	D (tables 74, 75)	All
P2[9]	IOCON_P2_09	R/W	0x030	0x4002 C124	D (tables 74, 75)	All
P2[10]	IOCON_P2_10	R/W	0x030	0x4002 C128	D (tables 74, 75)	All
P2[11]	IOCON_P2_11	R/W	0x030	0x4002 C12C	D (tables 74, 75)	All
P2[12]	IOCON_P2_12	R/W	0x030	0x4002 C130	D (tables 74, 75)	All
P2[13]	IOCON_P2_13	R/W	0x030	0x4002 C134	D (tables 74, 75)	All
P2[14]	IOCON_P2_14	R/W	0x030	0x4002 C138	D (tables 74, 75)	208
P2[15]	IOCON_P2_15	R/W	0x030	0x4002 C13C	D (tables 74, 75)	208
P2[16]	IOCON_P2_16	R/W	0x030	0x4002 C140	D (tables 74, 75)	180, 208
P2[17]	IOCON_P2_17	R/W	0x030	0x4002 C144	D (tables 74, 75)	180, 208
P2[18]	IOCON_P2_18	R/W	0x030	0x4002 C148	D (tables 74, 75)	180, 208
P2[19]	IOCON_P2_19	R/W	0x030	0x4002 C14C	D (tables 74, 75)	180, 208
P2[20]	IOCON_P2_20	R/W	0x030	0x4002 C150	D (tables 74, 75)	180, 208
P2[21]	IOCON_P2_21	R/W	0x030	0x4002 C154	D (tables 74, 75)	180, 208
P2[22]	IOCON_P2_22	R/W	0x030	0x4002 C158	D (tables 74, 75)	208
P2[23]	IOCON_P2_23	R/W	0x030	0x4002 C15C	D (tables 74, 75)	208
P2[24]	IOCON_P2_24	R/W	0x030	0x4002 C160	D (tables 74, 75)	180, 208
P2[25]	IOCON_P2_25	R/W	0x030	0x4002 C164	D (tables 74, 75)	180, 208
P2[26]	IOCON_P2_26	R/W	0x030	0x4002 C168	D (tables 74, 75)	208
P2[27]	IOCON_P2_27	R/W	0x030	0x4002 C16C	D (tables 74, 75)	208
P2[28]	IOCON_P2_28	R/W	0x030	0x4002 C170	D (tables 74, 75)	180, 208
P2[29]	IOCON_P2_29	R/W	0x030	0x4002 C174	D (tables 74, 75)	180, 208
P2[30]	IOCON_P2_30	R/W	0x030	0x4002 C178	D (tables 74, 75)	208
P2[31]	IOCON_P2_31	R/W	0x030	0x4002 C17C	D (tables 74, 75)	208

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

Table 71. I/O Control registers for port 3

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type	Available on packages
P3[0]	IOCON_P3_00	R/W	0x030	0x4002 C180	D (tables 74, 75)	144, 180, 208
P3[1]	IOCON_P3_01	R/W	0x030	0x4002 C184	D (tables 74, 75)	144, 180, 208
P3[2]	IOCON_P3_02	R/W	0x030	0x4002 C188	D (tables 74, 75)	144, 180, 208
P3[3]	IOCON_P3_03	R/W	0x030	0x4002 C18C	D (tables 74, 75)	144, 180, 208
P3[4]	IOCON_P3_04	R/W	0x030	0x4002 C190	D (tables 74, 75)	144, 180, 208
P3[5]	IOCON_P3_05	R/W	0x030	0x4002 C194	D (tables 74, 75)	144, 180, 208
P3[6]	IOCON_P3_06	R/W	0x030	0x4002 C198	D (tables 74, 75)	144, 180, 208
P3[7]	IOCON_P3_07	R/W	0x030	0x4002 C19C	D (tables 74, 75)	144, 180, 208
P3[8]	IOCON_P3_08	R/W	0x030	0x4002 C1A0	D (tables 74, 75)	180, 208
P3[9]	IOCON_P3_09	R/W	0x030	0x4002 C1A4	D (tables 74, 75)	180, 208
P3[10]	IOCON_P3_10	R/W	0x030	0x4002 C1A8	D (tables 74, 75)	180, 208
P3[11]	IOCON_P3_11	R/W	0x030	0x4002 C1AC	D (tables 74, 75)	180, 208
P3[12]	IOCON_P3_12	R/W	0x030	0x4002 C1B0	D (tables 74, 75)	180, 208
P3[13]	IOCON_P3_13	R/W	0x030	0x4002 C1B4	D (tables 74, 75)	180, 208
P3[14]	IOCON_P3_14	R/W	0x030	0x4002 C1B8	D (tables 74, 75)	180, 208
P3[15]	IOCON_P3_15	R/W	0x030	0x4002 C1BC	D (tables 74, 75)	180, 208
P3[16]	IOCON_P3_16	R/W	0x030	0x4002 C1C0	D (tables 74, 75)	208
P3[17]	IOCON_P3_17	R/W	0x030	0x4002 C1C4	D (tables 74, 75)	208
P3[18]	IOCON_P3_18	R/W	0x030	0x4002 C1C8	D (tables 74, 75)	208
P3[19]	IOCON_P3_19	R/W	0x030	0x4002 C1CC	D (tables 74, 75)	208
P3[20]	IOCON_P3_20	R/W	0x030	0x4002 C1D0	D (tables 74, 75)	208
P3[21]	IOCON_P3_21	R/W	0x030	0x4002 C1D4	D (tables 74, 75)	208
P3[22]	IOCON_P3_22	R/W	0x030	0x4002 C1D8	D (tables 74, 75)	208
P3[23]	IOCON_P3_23	R/W	0x030	0x4002 C1DC	D (tables 74, 75)	144, 180, 208
P3[24]	IOCON_P3_24	R/W	0x030	0x4002 C1E0	D (tables 74, 75)	144, 180, 208
P3[25]	IOCON_P3_25	R/W	0x030	0x4002 C1E4	D (tables 74, 75)	All
P3[26]	IOCON_P3_26	R/W	0x030	0x4002 C1E8	D (tables 74, 75)	All
P3[27]	IOCON_P3_27	R/W	0x030	0x4002 C1EC	D (tables 74, 75)	208
P3[28]	IOCON_P3_28	R/W	0x030	0x4002 C1F0	D (tables 74, 75)	208
P3[29]	IOCON_P3_29	R/W	0x030	0x4002 C1F4	D (tables 74, 75)	208
P3[30]	IOCON_P3_30	R/W	0x030	0x4002 C1F8	D (tables 74, 75)	208
P3[31]	IOCON_P3_31	R/W	0x030	0x4002 C1FC	D (tables 74, 75)	208

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

Table 72. I/O Control registers for port 4

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type	Available on packages
P4[0]	IOCON_P4_00	R/W	0x030	0x4002 C200	D (tables 74, 75)	144, 180, 208
P4[1]	IOCON_P4_01	R/W	0x030	0x4002 C204	D (tables 74, 75)	144, 180, 208
P4[2]	IOCON_P4_02	R/W	0x030	0x4002 C208	D (tables 74, 75)	144, 180, 208
P4[3]	IOCON_P4_03	R/W	0x030	0x4002 C20C	D (tables 74, 75)	144, 180, 208
P4[4]	IOCON_P4_04	R/W	0x030	0x4002 C210	D (tables 74, 75)	144, 180, 208
P4[5]	IOCON_P4_05	R/W	0x030	0x4002 C214	D (tables 74, 75)	144, 180, 208
P4[6]	IOCON_P4_06	R/W	0x030	0x4002 C218	D (tables 74, 75)	144, 180, 208
P4[7]	IOCON_P4_07	R/W	0x030	0x4002 C21C	D (tables 74, 75)	144, 180, 208
P4[8]	IOCON_P4_08	R/W	0x030	0x4002 C220	D (tables 74, 75)	144, 180, 208
P4[9]	IOCON_P4_09	R/W	0x030	0x4002 C224	D (tables 74, 75)	144, 180, 208
P4[10]	IOCON_P4_10	R/W	0x030	0x4002 C228	D (tables 74, 75)	144, 180, 208
P4[11]	IOCON_P4_11	R/W	0x030	0x4002 C22C	D (tables 74, 75)	144, 180, 208
P4[12]	IOCON_P4_12	R/W	0x030	0x4002 C230	D (tables 74, 75)	144, 180, 208
P4[13]	IOCON_P4_13	R/W	0x030	0x4002 C234	D (tables 74, 75)	144, 180, 208
P4[14]	IOCON_P4_14	R/W	0x030	0x4002 C238	D (tables 74, 75)	144, 180, 208
P4[15]	IOCON_P4_15	R/W	0x030	0x4002 C23C	D (tables 74, 75)	144, 180, 208
P4[16]	IOCON_P4_16	R/W	0x030	0x4002 C240	D (tables 74, 75)	180, 208
P4[17]	IOCON_P4_17	R/W	0x030	0x4002 C244	D (tables 74, 75)	180, 208
P4[18]	IOCON_P4_18	R/W	0x030	0x4002 C248	D (tables 74, 75)	180, 208
P4[19]	IOCON_P4_19	R/W	0x030	0x4002 C24C	D (tables 74, 75)	180, 208
P4[20]	IOCON_P4_20	R/W	0x030	0x4002 C250	D (tables 74, 75)	208
P4[21]	IOCON_P4_21	R/W	0x030	0x4002 C254	D (tables 74, 75)	208
P4[22]	IOCON_P4_22	R/W	0x030	0x4002 C258	D (tables 74, 75)	208
P4[23]	IOCON_P4_23	R/W	0x030	0x4002 C25C	D (tables 74, 75)	208
P4[24]	IOCON_P4_24	R/W	0x030	0x4002 C260	D (tables 74, 75)	144, 180, 208
P4[25]	IOCON_P4_25	R/W	0x030	0x4002 C264	D (tables 74, 75)	144, 180, 208
P4[26]	IOCON_P4_26	R/W	0x030	0x4002 C268	D (tables 74, 75)	180, 208
P4[27]	IOCON_P4_27	R/W	0x030	0x4002 C26C	D (tables 74, 75)	180, 208
P4[28]	IOCON_P4_28	R/W	0x030	0x4002 C270	D (tables 74, 75)	All
P4[29]	IOCON_P4_29	R/W	0x030	0x4002 C274	D (tables 74, 75)	All
P4[30]	IOCON_P4_30	R/W	0x030	0x4002 C278	D (tables 74, 75)	144, 180, 208
P4[31]	IOCON_P4_31	R/W	0x030	0x4002 C27C	D (tables 74, 75)	144, 180, 208

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

**Table 73. I/O Control registers for port 5**

Port pin	Register	Access	Reset Value <sup>[1]</sup>	Address	IOCON type	Available on packages
P5[0]	IOCON_P5_00	R/W	0x030	0x4002 C280	D (tables <a href="#">74</a> , <a href="#">75</a> )	All except TFBGA100
P5[1]	IOCON_P5_01	R/W	0x030	0x4002 C284	D (tables <a href="#">74</a> , <a href="#">75</a> )	144, 180, 208
P5[2]	IOCON_P5_02	R/W	0	0x4002 C288	I (tables <a href="#">80</a> , <a href="#">81</a> )	144, 180, 208
P5[3]	IOCON_P5_03	R/W	0	0x4002 C28C	I (tables <a href="#">80</a> , <a href="#">81</a> )	144, 180, 208
P5[4]	IOCON_P5_04	R/W	0x030	0x4002 C290	D (tables <a href="#">74</a> , <a href="#">75</a> )	All

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 8.4.1 I/O configuration register contents (IOCON)

The functions of bits in the IOCON register for each GPIO port pin is described in the following sections. There are some differences in IOCON for special port pins compared to most other port pins. These include pins that support analog functions (such as ADC inputs and the DAC output), the USB D+/D- pins, and specialized I<sup>2</sup>C pins:

- ["Type D IOCON registers \(applies to most GPIO port pins\)"](#)
- ["Type A IOCON registers \(applies to pins that include an analog function\)"](#)
- ["Type U IOCON registers \(applies to pins that include a USB D+ or D- function\)"](#)
- ["Type I IOCON registers \(applies to pins that include a specialized I<sup>2</sup>C function\)"](#)
- ["Type W IOCON registers \(these pins are otherwise the same as Type D, but include a selectable input glitch filter, and default to pull-down/pull-up disabled\)."](#)

8.4.1.1 Type D IOCON registers (applies to most GPIO port pins)

This IOCON table applies to all port pins except P0[7 to 9], P0[12 to 13], P0[23 to 31], P1[30 to 31], and P5[2 to 3]. Those pins include DAC, ADC, USB, I<sup>2</sup>C, or input glitch filter functions that alter the contents of the related IOCON registers.

Table 74. Type D IOCON registers bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See <a href="#">Table 75</a> for specific values.	000
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control). See <a href="#">Section 8.3.2 "Pin mode"</a> .	10
		00	Inactive (no pull-down/pull-up resistor enabled).	
		01	Pull-down resistor enabled.	
		10	Pull-up resistor enabled.	
		11	Repeater mode.	
5	HYS		Hysteresis. See <a href="#">Section 8.3.3 "Hysteresis"</a> .	1
		0	Disable.	
		1	Enable.	
6	INV		Input polarity. See <a href="#">Section 8.3.4 "Input Inversion"</a> .	0
		0	Input is not inverted (a HIGH on the pin reads as 1)	
		1	Input is inverted (a HIGH on the pin reads as 0)	
8:7	-		Reserved. Read value is undefined, only zero should be written.	NA
9	SLEW		Driver slew rate. See <a href="#">Section 8.3.7 "Output slew rate"</a> .	0
		0	Standard mode, output slew rate control is enabled. More outputs can be switched simultaneously.	
		1	Fast mode, slew rate control is disabled. Refer to the appropriate specific device data sheet for details.	
10	OD		Controls open-drain mode. See <a href="#">Section 8.3.9 "Open-Drain Mode"</a> .	0
		0	Normal push-pull output	
		1	Simulated open-drain output (high drive disabled)	
31:11	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 75. Type D I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P0_0	P0[0]	CAN_RD_1	U3_TXD	I2C1_SDA	U0_TXD			
IOCON_P0_1	P0[1]	CAN_TD_1	U3_RXD	I2C1_SCL	U0_RXD			
IOCON_P0_2	P0[2]	U0_TXD	U3_TXD					
IOCON_P0_3	P0[3]	U0_RXD	U3_RXD					
IOCON_P0_4	P0[4]	I2S_RX_SCK	CAN_RD_2	T2_CAP0				LCD_VD_0
IOCON_P0_5	P0[5]	I2S_RX_WS	CAN_TD_2	T2_CAP1				LCD_VD_1
IOCON_P0_6	P0[6]	I2S_RX_SDA	SSP1_SSEL	T2_MAT0	U1_RTS			LCD_VD_8
IOCON_P0_10	P0[10]	U2_TXD	I2C2_SDA	T3_MAT0				
IOCON_P0_11	P0[11]	U2_RXD	I2C2_SCL	T3_MAT1				
IOCON_P0_14	P0[14]	USB_HSTEN2	SSP1_SSEL	USB_CONNECT2				
IOCON_P0_15	P0[15]	U1_TXD	SSP0_SCK					
IOCON_P0_16	P0[16]	U1_RXD	SSP0_SSEL					
IOCON_P0_17	P0[17]	U1_CTS	SSP0_MISO					
IOCON_P0_18	P0[18]	U1_DCD	SSP0_MOSI					
IOCON_P0_19	P0[19]	U1_DSR	SD_CLK	I2C1_SDA				
IOCON_P0_20	P0[20]	U1_DTR	SD_CMD	I2C1_SCL				
IOCON_P0_21	P0[21]	U1_RI	SD_PWR	U4_OE	CAN_RD_1	U4_CLK		
IOCON_P0_22	P0[22]	U1_RTS	SD_DAT_0	U4_TXD	CAN_TD_1			
IOCON_P1_0	P1[0]	ENET_TXD[0]		T3_CAP1	SSP2_SCK			
IOCON_P1_1	P1[1]	ENET_TXD[1]		T3_MAT3	SSP2_MOSI			
IOCON_P1_2	P1[2]	ENET_TXD[2]	SD_CLK	PWM0_1				
IOCON_P1_3	P1[3]	ENET_TXD[3]	SD_CMD	PWM0_2				
IOCON_P1_4	P1[4]	ENET_TX_EN		T3_MAT2	SSP2_MISO			
IOCON_P1_5	P1[5]	ENET_TX_ER	SD_PWR	PWM0_3				
IOCON_P1_6	P1[6]	ENET_TX_CLK	SD_DAT_0	PWM0_4				
IOCON_P1_7	P1[7]	ENET_COL	SD_DAT_1	PWM0_5				
IOCON_P1_8	P1[8]	ENET_CRCS		T3_MAT1	SSP2_SSEL			
IOCON_P1_9	P1[9]	ENET_RXD_0		T3_MAT0				
IOCON_P1_10	P1[10]	ENET_RXD_1		T3_CAP0				
IOCON_P1_11	P1[11]	ENET_RXD_2	SD_DAT_2	PWM0_6				



Table 75. Type D I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P1_12	P1[12]	ENET_RXD_3	SD_DAT_3	PWM0_CAP0				
IOCON_P1_13	P1[13]	ENET_RX_DV						
IOCON_P1_14	P1[14]	ENET_RX_ER		T2_CAP0				
IOCON_P1_15	P1[15]	ENET_RX_CLK		I2C2_SDA				
IOCON_P1_16	P1[16]	ENET_MDC	I2S_TX_MCLK					
IOCON_P1_17	P1[17]	ENET_MDIO	I2S_RX_MCLK					
IOCON_P1_18	P1[18]	USB_UP_LED1	PWM1_1	T1_CAP0		SSP1_MISO		
IOCON_P1_19	P1[19]	USB_TX_E1	USB_PPWR1	T1_CAP1	MC_0A	SSP1_SCK	U2_OE	
IOCON_P1_20	P1[20]	USB_TX_DP1	PWM1_2	QEI_PHA	MC_FB0	SSP0_SCK	LCD_VD_6	LCD_VD_10
IOCON_P1_21	P1[21]	USB_TX_DM1	PWM1_3	SSP0_SSEL	MC_ABORT		LCD_VD_7	LCD_VD_11
IOCON_P1_22	P1[22]	USB_RCV1	USB_PWRD1	T1_MAT0	MC_0B	SSP1_MOSI	LCD_VD_8	LCD_VD_12
IOCON_P1_23	P1[23]	USB_RX_DP1	PWM1_4	QEI_PHB	MC_FB1	SSP0_MISO	LCD_VD_9	LCD_VD_13
IOCON_P1_24	P1[24]	USB_RX_DM1	PWM1_5	QEI_IDX	MC_FB2	SSP0_MOSI	LCD_VD_10	LCD_VD_14
IOCON_P1_25	P1[25]	USB_LS1	USB_HSTEN1	T1_MAT1	MC_1A	CLKOUT	LCD_VD_11	LCD_VD_15
IOCON_P1_26	P1[26]	USB_SSPND1	PWM1_6	T0_CAP0	MC_1B	SSP1_SSEL	LCD_VD_12	LCD_VD_20
IOCON_P1_27	P1[27]	USB_INT1	USB_OVRCR1	T0_CAP1	CLKOUT		LCD_VD_13	LCD_VD_21
IOCON_P1_28	P1[28]	USB_SCL1	PWM1_CAP0	T0_MAT0	MC_2A	SSP0_SSEL	LCD_VD_14	LCD_VD_22
IOCON_P1_29	P1[29]	USB_SDA1	PWM1_CAP1	T0_MAT1	MC_2B	U4_TXD	LCD_VD_15	LCD_VD_23
IOCON_P2_0	P2[0]	PWM1_1	U1_TXD					LCD_PWR
IOCON_P2_1	P2[1]	PWM1_2	U1_RXD					LCD_LE
IOCON_P2_2	P2[2]	PWM1_3	U1_CTS	T2_MAT3		TRACEDATA[3]		LCD_DCLK
IOCON_P2_3	P2[3]	PWM1_4	U1_DCD	T2_MAT2		TRACEDATA[2]		LCD_FP
IOCON_P2_4	P2[4]	PWM1_5	U1_DSR	T2_MAT1		TRACEDATA[1]		LCD_ENAB_M
IOCON_P2_5	P2[5]	PWM1_6	U1_DTR	T2_MAT0		TRACEDATA[0]		LCD_LP
IOCON_P2_6	P2[6]	PWM1_CAP0	U1_RI	T2_CAP0	U2_OE	TRACECLK	LCD_VD_0	LCD_VD_4
IOCON_P2_7	P2[7]	CAN_RD_2	U1_RTS				LCD_VD_1	LCD_VD_5
IOCON_P2_8	P2[8]	CAN_TD_2	U2_TXD	U1_CTS	ENET_MDC		LCD_VD_2	LCD_VD_6
IOCON_P2_9	P2[9]	USB_CONNECT1	U2_RXD	U4_RXD	ENET_MDIO		LCD_VD_3	LCD_VD_7
IOCON_P2_10	P2[10]	EINT_0	NMI					
IOCON_P2_11	P2[11]	EINT_1	SD_DAT_1	I2S_TX_SCK				LCD_CLKIN



Table 75. Type D I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register								
	000	001	010	011	100	101	110	111	
IOCON_P2_12	P2[12]	EINT_2	SD_DAT_2	I2S_TX_WS	LCD_VD_4	LCD_VD_3	LCD_VD_8	LCD_VD_18	
IOCON_P2_13	P2[13]	EINT_3	SD_DAT_3	I2S_TX_SDA		LCD_VD_5	LCD_VD_9	LCD_VD_19	
IOCON_P2_14	P2[14]	EMC_CS[2]	I2C1_SDA	T2_CAP0					
IOCON_P2_15	P2[15]	EMC_CS[3]	I2C1_SCL	T2_CAP1					
IOCON_P2_16	P2[16]	EMC_CAS							
IOCON_P2_17	P2[17]	EMC_RAS							
IOCON_P2_18	P2[18]	EMC_CLK[0]							
IOCON_P2_19	P2[19]	EMC_CLK[1]							
IOCON_P2_20	P2[20]	EMC_DYCS[0]							
IOCON_P2_21	P2[21]	EMC_DYCS[1]							
IOCON_P2_22	P2[22]	EMC_DYCS[2]	SSP0_SCK	T3_CAP0					
IOCON_P2_23	P2[23]	EMC_DYCS[3]	SSP0_SSEL	T3_CAP1					
IOCON_P2_24	P2[24]	EMC_CKE[0]							
IOCON_P2_25	P2[25]	EMC_CKE[1]							
IOCON_P2_26	P2[26]	EMC_CKE[2]	SSP0_MISO	T3_MAT0					
IOCON_P2_27	P2[27]	EMC_CKE[3]	SSP0_MOSI	T3_MAT1					
IOCON_P2_28	P2[28]	EMC_DQM[0]							
IOCON_P2_29	P2[29]	EMC_DQM[1]							
IOCON_P2_30	P2[30]	EMC_DQM[2]	I2C2_SDA	T3_MAT2					
IOCON_P2_31	P2[31]	EMC_DQM[3]	I2C2_SCL	T3_MAT3					
IOCON_P3_0	P3[0]	EMC_D[0]							
IOCON_P3_1	P3[1]	EMC_D[1]							
IOCON_P3_2	P3[2]	EMC_D[2]							
IOCON_P3_3	P3[3]	EMC_D[3]							
IOCON_P3_4	P3[4]	EMC_D[4]							
IOCON_P3_5	P3[5]	EMC_D[5]							
IOCON_P3_6	P3[6]	EMC_D[6]							
IOCON_P3_7	P3[7]	EMC_D[7]							
IOCON_P3_8	P3[8]	EMC_D[8]							
IOCON_P3_9	P3[9]	EMC_D[9]							

Table 75. Type D I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P3_10	P3[10]	EMC_D[10]						
IOCON_P3_11	P3[11]	EMC_D[11]						
IOCON_P3_12	P3[12]	EMC_D[12]						
IOCON_P3_13	P3[13]	EMC_D[13]						
IOCON_P3_14	P3[14]	EMC_D[14]						
IOCON_P3_15	P3[15]	EMC_D[15]						
IOCON_P3_16	P3[16]	EMC_D[16]	PWM0_1	U1_TXD				
IOCON_P3_17	P3[17]	EMC_D[17]	PWM0_2	U1_RXD				
IOCON_P3_18	P3[18]	EMC_D[18]	PWM0_3	U1_CTS				
IOCON_P3_19	P3[19]	EMC_D[19]	PWM0_4	U1_DCD				
IOCON_P3_20	P3[20]	EMC_D[20]	PWM0_5	U1_DSR				
IOCON_P3_21	P3[21]	EMC_D[21]	PWM0_6	U1_DTR				
IOCON_P3_22	P3[22]	EMC_D[22]	PWM0_CAP0	U1_RI				
IOCON_P3_23	P3[23]	EMC_D[23]	PWM1_CAP0	T0_CAP0				
IOCON_P3_24	P3[24]	EMC_D[24]	PWM1_1	T0_CAP1				
IOCON_P3_25	P3[25]	EMC_D[25]	PWM1_2	T0_MAT0				
IOCON_P3_26	P3[26]	EMC_D[26]	PWM1_3	T0_MAT1	STCLK			
IOCON_P3_27	P3[27]	EMC_D[27]	PWM1_4	T1_CAP0				
IOCON_P3_28	P3[28]	EMC_D[28]	PWM1_5	T1_CAP1				
IOCON_P3_29	P3[29]	EMC_D[29]	PWM1_6	T1_MAT0				
IOCON_P3_30	P3[30]	EMC_D[30]	U1_RTS	T1_MAT1				
IOCON_P3_31	P3[31]	EMC_D[31]		T1_MAT2				
IOCON_P4_0	P4[0]	EMC_A[0]						
IOCON_P4_1	P4[1]	EMC_A[1]						
IOCON_P4_2	P4[2]	EMC_A[2]						
IOCON_P4_3	P4[3]	EMC_A[3]						
IOCON_P4_4	P4[4]	EMC_A[4]						
IOCON_P4_5	P4[5]	EMC_A[5]						
IOCON_P4_6	P4[6]	EMC_A[6]						
IOCON_P4_7	P4[7]	EMC_A[7]						

Table 75. Type D I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P4_8	P4[8]	EMC_A[8]						
IOCON_P4_9	P4[9]	EMC_A[9]						
IOCON_P4_10	P4[10]	EMC_A[10]						
IOCON_P4_11	P4[11]	EMC_A[11]						
IOCON_P4_12	P4[12]	EMC_A[12]						
IOCON_P4_13	P4[13]	EMC_A[13]						
IOCON_P4_14	P4[14]	EMC_A[14]						
IOCON_P4_15	P4[15]	EMC_A[15]						
IOCON_P4_16	P4[16]	EMC_A[16]						
IOCON_P4_17	P4[17]	EMC_A[17]						
IOCON_P4_18	P4[18]	EMC_A[18]						
IOCON_P4_19	P4[19]	EMC_A[19]						
IOCON_P4_20	P4[20]	EMC_A[20]	I2C2_SDA	SSP1_SCK				
IOCON_P4_21	P4[21]	EMC_A[21]	I2C2_SCL	SSP1_SSEL				
IOCON_P4_22	P4[22]	EMC_A[22]	U2_TXD	SSP1_MISO				
IOCON_P4_23	P4[23]	EMC_A[23]	U2_RXD	SSP1_MOSI				
IOCON_P4_24	P4[24]	$\overline{\text{EMC\_OE}}$						
IOCON_P4_25	P4[25]	$\overline{\text{EMC\_WE}}$						
IOCON_P4_26	P4[26]	$\overline{\text{EMC\_BLS[0]}}$						
IOCON_P4_27	P4[27]	$\overline{\text{EMC\_BLS[1]}}$						
IOCON_P4_28	P4[28]	$\overline{\text{EMC\_BLS[2]}}$	U3_TXD	T2_MAT0		LCD_VD_6	LCD_VD_10	LCD_VD_2
IOCON_P4_29	P4[29]	$\overline{\text{EMC\_BLS[3]}}$	U3_RXD	T2_MAT1	I2C2_SCL	LCD_VD_7	LCD_VD_11	LCD_VD_3
IOCON_P4_30	P4[30]	$\overline{\text{EMC\_CS[0]}}$						
IOCON_P4_31	P4[31]	$\overline{\text{EMC\_CS[1]}}$						
IOCON_P5_0	P5[0]	EMC_A[24]		T2_MAT2				
IOCON_P5_1	P5[1]	EMC_A[25]		T2_MAT3				
IOCON_P5_4	P5[4]	U0_OE		T3_MAT3	U4_TXD			

**8.4.1.2 Type A IOCON registers (applies to pins that include an analog function)**

This IOCON table applies to pins P0[12 to 13], P0[23 to 26], and P1[30 to 31]. The presence of the DAC output on P0[26] makes that pin slightly different, see the description of bit 16 below.

**Table 76. Type A IOCON registers bit description**

Bit	Symbol	Value	Description	Reset value <sup>[1]</sup>
2:0	FUNC		Selects pin function. See <a href="#">Table 77</a> for specific values.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control). See <a href="#">Section 8.3.2 “Pin mode”</a> .	10
		00	Inactive (no pull-down/pull-up resistor enabled).	
		01	Pull-down resistor enabled.	
		10	Pull-up resistor enabled.	
		11	Repeater mode.	
5	HYS		Hysteresis. See <a href="#">Section 8.3.3 “Hysteresis”</a> .	1
		0	Disable.	
		1	Enable.	
6	INVERT		Input polarity. See <a href="#">Section 8.3.4 “Input Inversion”</a> .	0
		0	Input is not inverted (a HIGH on the pin reads as 1)	
		1	Input is inverted (a HIGH on the pin reads as 0)	
7	ADMODE		Select Analog/Digital mode. See <a href="#">Section 8.3.5 “Analog/digital mode”</a> .	1
		0	Analog mode.	
		1	Digital mode.	
8	FILTER		Controls glitch filter. See <a href="#">Section 8.3.6 “Input filter”</a> .	1
		0	Noise pulses below approximately 10 ns are filtered out	
		1	No input filtering is done	
9	-		Reserved. Read value is undefined, only zero should be written.	NA
10	OD		Controls open-drain mode. See <a href="#">Section 8.3.9 “Open-Drain Mode”</a> .	0
		0	Normal push-pull output	
		1	Simulated open-drain output (high drive disabled)	
14:11	-		Reserved. Read value is undefined, only zero should be written.	NA
16	DACEN		DAC enable control. This bit applies only to P0[26], which includes the DAC output function DAC_OUT. See <a href="#">Section 8.3.10 “DAC enable”</a> .	0
		0	DAC is disabled	
		1	DAC is enabled	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

Table 77. Type A I/O Control registers: FUNC values and pin functions

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P0_12	P0[12]	USB_PPWR2	SSP1_MISO	ADC0_IN[6]				
IOCON_P0_13	P0[13]	USB_UP_LED2	SSP1_MOSI	ADC0_IN[7]				
IOCON_P0_23	P0[23]	ADC0_IN[0]	I2S_RX_SCK	T3_CAP0				
IOCON_P0_24	P0[24]	ADC0_IN[1]	I2S_RX_WS	T3_CAP1				
IOCON_P0_25	P0[25]	ADC0_IN[2]	I2S_RX_SDA	U3_TXD				
IOCON_P0_26	P0[26]	ADC0_IN[3]	DAC_OUT	U3_RXD				
IOCON_P1_30	P1[30]	USB_PWRD2	USB_VBUS	ADC0_IN[4]	I2C0_SDA	U3_OE		
IOCON_P1_31	P1[31]	USB_OVRCR2	SSP1_SCK	ADC0_IN[5]	I2C0_SCL			

**8.4.1.3 Type U IOCON registers (applies to pins that include a USB D+ or D-function)**

This IOCON table applies to pins P0[29], P0[30], and P0[31]. These special function pins do not include the selectable modes and options of other pins.

**Table 78. Type U IOCON registers bit description**

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See <a href="#">Table 79</a> for specific values.	000
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

**Table 79. Type U I/O Control registers: FUNC values and pin functions**

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P0_29	P0[29]	USB_D+1	EINT_0					
IOCON_P0_30	P0[30]	USB_D-1	EINT_1					
IOCON_P0_31	P0[31]	USB_D+2						

**8.4.1.4 Type I IOCON registers (applies to pins that include a specialized I<sup>2</sup>C function)**

This IOCON table applies to pins P0[27 to 28] and P5[2 to 3].

**Table 80. Type I IOCON registers bit description**

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See <a href="#">Table 81</a> for specific values.	0
5:3	-		Reserved. Read value is undefined, only zero should be written.	NA
6	INVERT		Input polarity. See <a href="#">Section 8.3.4 "Input Inversion"</a> .	0
		0	Input is not inverted (a HIGH on the pin reads as 1)	
		1	Input is inverted (a HIGH on the pin reads as 0)	
7	-		Reserved. Read value is undefined, only zero should be written.	NA
8	HS		Configures I <sup>2</sup> C features for standard mode, fast mode, and Fast Mode Plus operation. See <a href="#">Section 8.3.8 "I<sup>2</sup>C modes"</a> .	0
		0	I <sup>2</sup> C 50ns glitch filter and slew rate control enabled.	
		1	I <sup>2</sup> C 50ns glitch filter and slew rate control disabled.	
9	HIDRIVE		Controls sink current capability of the pin, only for P5[2] and P5[3]. See <a href="#">Section 8.3.8 "I<sup>2</sup>C modes"</a> .	0
		0	Output drive sink is 4 mA. This is sufficient for standard and fast mode I <sup>2</sup> C.	
		1	Output drive sink is 20 mA. This is needed for Fast Mode Plus I <sup>2</sup> C. Refer to the appropriate specific device data sheet for details.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

**Table 81. Type I I/O Control registers: FUNC values and pin functions**

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P0_27	P0[27]	I2C0_SDA	USB_SDA1					
IOCON_P0_28	P0[28]	I2C0_SCL	USB_SCL1					
IOCON_P5_2	P5[2]			T3_MAT2		I2C0_SDA		
IOCON_P5_3	P5[3]				U4_RXD	I2C0_SCL		

**8.4.1.5 Type W IOCON registers (these pins are otherwise the same as Type D, but include a selectable input glitch filter, and default to pull-down/pull-up disabled).**

This IOCON table applies to pins P0[7], P0[8], and P0[9].

**Table 82. Type W IOCON registers bit description**

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See <a href="#">Table 83</a> for specific values.	000
4:3	MODE		Selects the output functional mode for the pin (on-chip pull-up/pull-down resistor control). See <a href="#">Section 8.3.2 “Pin mode”</a> .	00
		00	Inactive (no pull-down/pull-up resistor enabled).	
		01	Pull-down resistor enabled.	
		10	Pull-up resistor enabled.	
		11	Repeater mode.	
5	HYS		Hysteresis. See <a href="#">Section 8.3.3 “Hysteresis”</a> .	1
		0	Disable.	
		1	Enable.	
6	INV		Input polarity. See <a href="#">Section 8.3.4 “Input Inversion”</a> .	0
		0	Input is not inverted (a HIGH on the pin reads as 1)	
		1	Input is inverted (a HIGH on the pin reads as 0)	
7	-		<b>Note:</b> this bit must be set to 1 for normal operation.	1
8	FILTER		Controls glitch filter. See <a href="#">Section 8.3.6 “Input filter”</a> .	0
		0	Noise pulses below approximately 10 ns are filtered out	
		1	No input filtering is done	
9	SLEW		Driver slew rate. See <a href="#">Section 8.3.7 “Output slew rate”</a> .	0
		0	Standard mode, output slew rate control is enabled. More outputs can be switched simultaneously.	
		1	Fast mode, slew rate control is disabled. Refer to the appropriate specific device data sheet for details.	
10	OD		Controls open-drain mode. See <a href="#">Section 8.3.9 “Open-Drain Mode”</a> .	0
		0	Normal push-pull output	
		1	Simulated open-drain output (high drive disabled)	
31:11	-		Reserved. Read value is undefined, only zero should be written.	NA

**Table 83. Type W I/O Control registers: FUNC values and pin functions**

Register	Value of FUNC field in IOCON register							
	000	001	010	011	100	101	110	111
IOCON_P0_7	P0[7]	I2S_TX_SCK	SSP1_SCK	T2_MAT1	RTC_EV0			LCD_VD_9
IOCON_P0_8	P0[8]	I2S_TX_WS	SSP1_MISO	T2_MAT2	RTC_EV1			LCD_VD_16
IOCON_P0_9	P0[9]	I2S_TX_SDA	SSP1_MOSI	T2_MAT3	RTC_EV2			LCD_VD_17



## 9.1 Basic configuration

GPIOs are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCGPIO. This enables the GPIOs themselves, GPIO interrupts, and the IOCON block.
2. Pins: See [Section 8.4.1](#) for GPIO pins and their modes.
3. Wake-up: GPIO ports 0 and 2 can be used for wake-up if needed, see ([Section 4.7.8](#)).
4. Interrupts: Enable GPIO interrupts in IO0/2IntEnR ([Table 98](#)) or IO0/2IntEnF ([Table 100](#)). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

## 9.2 Features

### 9.2.1 Digital I/O ports

- Accelerated GPIO functions:
  - GPIO registers are located on a peripheral AHB bus for fast I/O timing.
  - Mask registers allow treating sets of port bits as a group, leaving other bits unchanged.
  - All GPIO registers are byte, half-word, and word addressable.
  - Entire port value can be written in one instruction.
  - GPIO registers are accessible by the GPDMA.
- Bit-level set and clear registers allow a single instruction set or clear of any number of bits in one port.
- All GPIO registers support Cortex-M3 bit-banding.
- GPIO registers are accessible by the GPDMA controller to allow DMA of data to or from GPIOs, synchronized to any DMA request.
- Direction control of individual port bits.
- All I/Os default to input with pull-up after reset.

### 9.2.2 Interrupt generating digital ports

- Port 0 and Port 2 can provide a single interrupt for any combination of port pins.
- Each port pin can be programmed to generate an interrupt on a rising edge, a falling edge, or both.
- Edge detection is asynchronous, so it may operate when clocks are not present, such as during Power-down mode. With this feature, level triggered interrupts are not needed.
- Each enabled interrupt contributes to a wake-up signal that can be used to bring the part out of Power-down mode.

- Registers provide a software view of pending rising edge interrupts, pending falling edge interrupts, and overall pending GPIO interrupts.
- The GPIO interrupt function does not require that the pin be configured for GPIO. This allows interrupting on a change to a pin that is part of an operating peripheral interface.

### 9.3 Applications

- General purpose I/O
- Driving LEDs or other indicators
- Controlling off-chip devices
- Sensing digital inputs, detecting edges
- Bringing the part out of Power-down mode

### 9.4 Pin description

**Table 84. GPIO pin description**

Pin Name	Type	Description
P0[31:0]; P1[31:0]; P2[31:0]; P3[31:0]; P4[31:0]; P5[4:0]	Input/ Output	General purpose input/output. These are typically shared with other peripherals functions and will therefore not all be available in an application. Packaging options may affect the number of GPIOs available in a particular device.  Some pins may be limited by requirements of the alternate functions of the pin. For example, some pins that can be used for I <sup>2</sup> C are special pins, and some of that behavior is inherited by any other function selected on that pin. Details may be found in <a href="#">Section 7.1</a> .

## 9.5 Register description

The registers in [Table 85](#) represent the enhanced GPIO features available on all of the GPIO ports. These registers are located on an AHB bus for fast read and write timing. They can all be accessed in byte, half-word, and word sizes. A mask register allows access to a group of bits in a single GPIO port independently from other bits in the same port.

**Table 85. GPIO register map (local bus accessible registers - enhanced GPIO features)**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	PORTn Register Name & Address	Table
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0	FIO0DIR - 0x2009 8000 FIO1DIR - 0x2009 8020 FIO2DIR - 0x2009 8040 FIO3DIR - 0x2009 8060 FIO4DIR - 0x2009 8080 FIO5DIR - 0x2009 80A0	<a href="#">87</a>
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	R/W	0	FIO0MASK - 0x2009 8010 FIO1MASK - 0x2009 8030 FIO2MASK - 0x2009 8050 FIO3MASK - 0x2009 8070 FIO4MASK - 0x2009 8090 FIO5MASK - 0x2009 80B0	<a href="#">95</a>
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK.  <b>Important:</b> if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state.	R/W	0	FIO0PIN - 0x2009 8014 FIO1PIN - 0x2009 8034 FIO2PIN - 0x2009 8054 FIO3PIN - 0x2009 8074 FIO4PIN - 0x2009 8094 FIO5PIN - 0x2009 80B4	<a href="#">93</a>
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	R/W	0	FIO0SET - 0x2009 8018 FIO1SET - 0x2009 8038 FIO2SET - 0x2009 8058 FIO3SET - 0x2009 8078 FIO4SET - 0x2009 8098 FIO5SET - 0x2009 80B8	<a href="#">89</a>
FIOCLR	Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.	WO	0	FIO0CLR - 0x2009 801C FIO1CLR - 0x2009 803C FIO2CLR - 0x2009 805C FIO3CLR - 0x2009 807C FIO4CLR - 0x2009 809C FIO5CLR - 0x2009 80BC	<a href="#">91</a>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

**Table 86. GPIO interrupt register map**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	PORTn Register Name & Address	Table
IntEnR	GPIO Interrupt Enable for Rising edge.	R/W	0	IO0IntEnR - 0x4002 8090	<a href="#">98</a>
				IO2IntEnR - 0x4002 80B0	<a href="#">99</a>
IntEnF	GPIO Interrupt Enable for Falling edge.	R/W	0	IO0IntEnR - 0x4002 8094	<a href="#">100</a>
				IO2IntEnR - 0x4002 80B4	<a href="#">101</a>
IntStatR	GPIO Interrupt Status for Rising edge.	RO	0	IO0IntStatR - 0x4002 8084	<a href="#">102</a>
				IO2IntStatR - 0x4002 80A4	<a href="#">103</a>
IntStatF	GPIO Interrupt Status for Falling edge.	RO	0	IO0IntStatF - 0x4002 8088	<a href="#">104</a>
				IO2IntStatF - 0x4002 80A8	<a href="#">105</a>
IntClr	GPIO Interrupt Clear.	WO	0	IO0IntClr - 0x4002 808C	<a href="#">106</a>
				IO2IntClr - 0x4002 80AC	<a href="#">107</a>
IntStatus	GPIO overall Interrupt Status.	RO	0	IOIntStatus - 0x4002 8080	<a href="#">97</a>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

### 9.5.1 GPIO port Direction register FIOxDIR (FIO0DIR to FIO5DIR- 0x2009 8000 to 0x2009 80A0)

This word accessible register is used to control the direction of the pins when they are configured as GPIO port pins. Direction bit for any pin must be set according to the pin functionality.

Note that GPIO pins P0[29] and P0[30] are shared with the USB\_D+ and USB\_D- pins and must have the same direction. If either FIO0DIR bit 29 or 30 are configured as zero, both P0[29] and P0[30] will be inputs. If both FIO0DIR bits 29 and 30 are ones, both P0[29] and P0[30] will be outputs.

**Table 87. Fast GPIO port Direction register FIO0DIR to FIO5DIR - addresses 0x2009 8000 to 0x2009 80A0) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	FIO0DIR FIO1DIR FIO2DIR FIO3DIR FIO4DIR FIO5DIR		Fast GPIO Direction PORTx control bits. Bit 0 in FIOxDIR controls pin Px[0], bit 31 in FIOxDIR controls pin Px[31].	0x0
		0	Controlled pin is input.	
		1	Controlled pin is output.	

Aside from the 32-bit long and word only accessible FIODIR register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in [Table 88](#), too. Next to providing the same functions as the FIODIR register, these additional registers allow easier and faster access to the physical port pins.

**Table 88. Fast GPIO port Direction control byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxDIR0	Fast GPIO Port x Direction control register 0. Bit 0 in FIOxDIR0 register corresponds to pin Px[0] ... bit 7 to pin Px[7].	8 (byte) R/W	0x00	FIO0DIR0 - 0x2009 8000 FIO1DIR0 - 0x2009 8020 FIO2DIR0 - 0x2009 8040 FIO3DIR0 - 0x2009 8060 FIO4DIR0 - 0x2009 8080 FIO5DIR0 - 0x2009 80A0
FIOxDIR1	Fast GPIO Port x Direction control register 1. Bit 0 in FIOxDIR1 register corresponds to pin Px[8] ... bit 7 to pin Px[15].	8 (byte) R/W	0x00	FIO0DIR1 - 0x2009 8001 FIO1DIR1 - 0x2009 8021 FIO2DIR1 - 0x2009 8041 FIO3DIR1 - 0x2009 8061 FIO4DIR1 - 0x2009 8081 FIO5DIR1 - 0x2009 80A1
FIOxDIR2	Fast GPIO Port x Direction control register 2. Bit 0 in FIOxDIR2 register corresponds to pin Px[16] ... bit 7 to pin Px[23].	8 (byte) R/W	0x00	FIO0DIR2 - 0x2009 8002 FIO1DIR2 - 0x2009 8022 FIO2DIR2 - 0x2009 8042 FIO3DIR2 - 0x2009 8062 FIO4DIR2 - 0x2009 8082 FIO5DIR2 - 0x2009 80A2
FIOxDIR3	Fast GPIO Port x Direction control register 3. Bit 0 in FIOxDIR3 register corresponds to pin Px[24] ... bit 7 to pin Px[31].	8 (byte) R/W	0x00	FIO0DIR3 - 0x2009 8003 FIO1DIR3 - 0x2009 8023 FIO2DIR3 - 0x2009 8043 FIO3DIR3 - 0x2009 8063 FIO4DIR3 - 0x2009 8083 FIO5DIR3 - 0x2009 80A3
FIOxDIRL	Fast GPIO Port x Direction control Lower half-word register. Bit 0 in FIOxDIRL register corresponds to pin Px[0] ... bit 15 to pin Px[15].	16 (half-word) R/W	0x0000	FIO0DIRL - 0x2009 8000 FIO1DIRL - 0x2009 8020 FIO2DIRL - 0x2009 8040 FIO3DIRL - 0x2009 8060 FIO4DIRL - 0x2009 8080 FIO5DIRL - 0x2009 80A0
FIOxDIRU	Fast GPIO Port x Direction control Upper half-word register. Bit 0 in FIOxDIRU register corresponds to Px[16] ... bit 15 to Px[31].	16 (half-word) R/W	0x0000	FIO0DIRU - 0x2009 8002 FIO1DIRU - 0x2009 8022 FIO2DIRU - 0x2009 8042 FIO3DIRU - 0x2009 8062 FIO4DIRU - 0x2009 8082 FIO5DIRU - 0x2009 80A2

### 9.5.2 GPIO port output Set register FIOxSET (FIO0SET to FIO5SET - 0x2009 8018 to 0x2009 80B8)

This register is used to produce a HIGH level output at the port pins configured as GPIO in an OUTPUT mode. Writing 1 produces a HIGH level at the corresponding port pins. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing 1 to the corresponding bit in the FIOxSET has no effect.

Reading the FIOxSET register returns the value of this register, as determined by previous writes to FIOxSET and FIOxCLR (or FIOxPIN as noted above). This value does not reflect the effect of any outside world influence on the I/O pins.

Access to a port pin via the FIOxSET register is conditioned by the corresponding bit of the FIOxMASK register (see [Section 9.5.5](#)).

**Table 89. Fast GPIO port output Set register (FIO0SET to FIO4SET - addresses 0x2009 8018 to 0x2009 8098) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	FIO0SET FIO1SET FIO2SET FIO3SET FIO4SET FIO5SET	0	Fast GPIO output value Set bits. Bit 0 in FIOxSET controls pin Px[0], bit 31 in FIOxSET controls pin Px[31]. Controlled pin output is unchanged.	0x0
		1	Controlled pin output is set to HIGH.	

Aside from the 32-bit long and word only accessible FIOxSET register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in [Table 90](#), too. Next to providing the same functions as the FIOxSET register, these additional registers allow easier and faster access to the physical port pins.

**Table 90. Fast GPIO port output Set byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxSET0	Fast GPIO Port x output Set register 0. Bit 0 in FIOxSET0 register corresponds to pin Px[0] ... bit 7 to pin Px[7].	8 (byte) R/W	0x00	FIO0SET0 - 0x2009 8018 FIO1SET0 - 0x2009 8038 FIO2SET0 - 0x2009 8058 FIO3SET0 - 0x2009 8078 FIO4SET0 - 0x2009 8098 FIO5SET0 - 0x2009 80B8
FIOxSET1	Fast GPIO Port x output Set register 1. Bit 0 in FIOxSET1 register corresponds to pin Px[8] ... bit 7 to pin Px[15].	8 (byte) R/W	0x00	FIO0SET1 - 0x2009 8019 FIO1SET1 - 0x2009 8039 FIO2SET1 - 0x2009 8059 FIO3SET1 - 0x2009 8079 FIO4SET1 - 0x2009 8099 FIO5SET1 - 0x2009 80B9
FIOxSET2	Fast GPIO Port x output Set register 2. Bit 0 in FIOxSET2 register corresponds to pin Px[16] ... bit 7 to pin Px[23].	8 (byte) R/W	0x00	FIO0SET2 - 0x2009 801A FIO1SET2 - 0x2009 803A FIO2SET2 - 0x2009 805A FIO3SET2 - 0x2009 807A FIO4SET2 - 0x2009 809A FIO5SET2 - 0x2009 80BA

**Table 90. Fast GPIO port output Set byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxSET3	Fast GPIO Port x output Set register 3. Bit 0 in FIOxSET3 register corresponds to pin Px[24] ... bit 7 to pin Px[31].	8 (byte) R/W	0x00	FIO0SET3 - 0x2009 801B FIO1SET3 - 0x2009 803B FIO2SET3 - 0x2009 805B FIO3SET3 - 0x2009 807B FIO4SET3 - 0x2009 809B FIO5SET3 - 0x2009 80BB
FIOxSETL	Fast GPIO Port x output Set Lower half-word register. Bit 0 in FIOxSETL register corresponds to pin Px[0] ... bit 15 to pin Px[15].	16 (half-word) R/W	0x0000	FIO0SETL - 0x2009 8018 FIO1SETL - 0x2009 8038 FIO2SETL - 0x2009 8058 FIO3SETL - 0x2009 8078 FIO4SETL - 0x2009 8098 FIO5SETL - 0x2009 80B8
FIOxSETU	Fast GPIO Port x output Set Upper half-word register. Bit 0 in FIOxSETU register corresponds to Px[16] ... bit 15 to Px[31].	16 (half-word) R/W	0x0000	FIO0SETU - 0x2009 801A FIO1SETU - 0x2009 803A FIO2SETU - 0x2009 805A FIO3SETU - 0x2009 807A FIO4SETU - 0x2009 809A FIO5SETU - 0x2009 80BA

### 9.5.3 GPIO port output Clear register FIOxCLR (FIO0CLR to FIO5CLR- 0x2009 801C to 0x2009 80BC)

This register is used to produce a LOW level output at port pins configured as GPIO in an OUTPUT mode. Writing 1 produces a LOW level at the corresponding port pin and clears the corresponding bit in the FIOxSET register. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing to FIOxCLR has no effect.

Access to a port pin via the FIOxCLR register is conditioned by the corresponding bit of the FIOxMASK register (see [Section 9.5.5](#)).

**Table 91. Fast GPIO port output Clear register (FIO0CLR to FIO4CLR- addresses 0x2009 801C to 0x2009 809C) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	FIO0CLR FIO1CLR FIO2CLR FIO3CLR FIO4CLR FIO5CLR		Fast GPIO output value Clear bits. Bit 0 in FIOxCLR controls pin Px[0], bit 31 controls pin Px[31].	0x0
		0	Controlled pin output is unchanged.	
		1	Controlled pin output is set to LOW.	

Aside from the 32-bit long and word only accessible FIOxCLR register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in [Table 92](#), too. Next to providing the same functions as the FIOxCLR register, these additional registers allow easier and faster access to the physical port pins.

**Table 92. Fast GPIO port output Clear byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxCLR0	Fast GPIO Port x output Clear register 0. Bit 0 in FIOxCLR0 register corresponds to pin Px[0] ... bit 7 to pin Px[7].	8 (byte) WO	0x00	FIO0CLR0 - 0x2009 801C FIO1CLR0 - 0x2009 803C FIO2CLR0 - 0x2009 805C FIO3CLR0 - 0x2009 807C FIO4CLR0 - 0x2009 809C FIO5CLR0 - 0x2009 80BC
FIOxCLR1	Fast GPIO Port x output Clear register 1. Bit 0 in FIOxCLR1 register corresponds to pin Px[8] ... bit 7 to pin Px[15].	8 (byte) WO	0x00	FIO0CLR1 - 0x2009 801D FIO1CLR1 - 0x2009 803D FIO2CLR1 - 0x2009 805D FIO3CLR1 - 0x2009 807D FIO4CLR1 - 0x2009 809D FIO5CLR1 - 0x2009 80BD
FIOxCLR2	Fast GPIO Port x output Clear register 2. Bit 0 in FIOxCLR2 register corresponds to pin Px[16] ... bit 7 to pin Px[23].	8 (byte) WO	0x00	FIO0CLR2 - 0x2009 801E FIO1CLR2 - 0x2009 803E FIO2CLR2 - 0x2009 805E FIO3CLR2 - 0x2009 807E FIO4CLR2 - 0x2009 809E FIO5CLR2 - 0x2009 80BE
FIOxCLR3	Fast GPIO Port x output Clear register 3. Bit 0 in FIOxCLR3 register corresponds to pin Px[24] ... bit 7 to pin Px[31].	8 (byte) WO	0x00	FIO0CLR3 - 0x2009 801F FIO1CLR3 - 0x2009 803F FIO2CLR3 - 0x2009 805F FIO3CLR3 - 0x2009 807F FIO4CLR3 - 0x2009 809F FIO5CLR3 - 0x2009 80BF
FIOxCLRL	Fast GPIO Port x output Clear Lower half-word register. Bit 0 in FIOxCLRL register corresponds to pin Px[0] ... bit 15 to pin Px[15].	16 (half-word) WO	0x0000	FIO0CLRL - 0x2009 801C FIO1CLRL - 0x2009 803C FIO2CLRL - 0x2009 805C FIO3CLRL - 0x2009 807C FIO4CLRL - 0x2009 809C FIO5CLRL - 0x2009 80BC
FIOxCLRU	Fast GPIO Port x output Clear Upper half-word register. Bit 0 in FIOxCLRU register corresponds to pin Px[16] ... bit 15 to Px[31].	16 (half-word) WO	0x0000	FIO0CLRU - 0x2009 801E FIO1CLRU - 0x2009 803E FIO2CLRU - 0x2009 805E FIO3CLRU - 0x2009 807E FIO4CLRU - 0x2009 809E FIO5CLRU - 0x2009 80BE

### 9.5.4 GPIO port Pin value register FIOxPIN (FIO0PIN to FIO5PIN-0x2009 8014 to 0x2009 80B4)

This register provides the value of port pins that are configured to perform only digital functions. The register will give the logic value of the pin regardless of whether the pin is configured for input or output, or as GPIO or an alternate digital function. As an example, a particular port pin may have GPIO input, GPIO output, UART receive, and PWM output as selectable functions. Any configuration of that pin will allow its current logic state to be read from the corresponding FIOxPIN register.



If a pin has an analog function as one of its options, the pin state cannot be read if the analog configuration is selected. Selecting the pin as an A/D input disconnects the digital features of the pin. In that case, the pin value read in the FIOxPIN register is not valid.

Writing to the FIOxPIN register stores the value in the port output register, bypassing the need to use both the FIOxSET and FIOxCLR registers to obtain the entire written value. This feature should be used carefully in an application since it affects the entire port.

Access to a port pin via the FIOxPIN register is conditioned by the corresponding bit of the FIOxMASK register (see [Section 9.5.5](#)).

Only pins masked with zeros in the Mask register (see [Section 9.5.5](#)) will be correlated to the current content of the Fast GPIO port value register.

**Table 93. Fast GPIO port Pin value register (FIO0PIN to FIO4PIN- addresses 0x2009 8014 to 0x2009 8094) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	FIO0VAL FIO1VAL FIO2VAL FIO3VAL FIO4VAL FIO5VAL		Fast GPIO output value Set bits. Bit 0 in FIOxCLR corresponds to pin Px[0], bit 31 in FIOxCLR corresponds to pin Px[31].	0x0
		0	Controlled pin output is set to LOW.	
		1	Controlled pin output is set to HIGH.	

Aside from the 32-bit long and word only accessible FIOxPIN register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in [Table 94](#), too. Next to providing the same functions as the FIOxPIN register, these additional registers allow easier and faster access to the physical port pins.

**Table 94. Fast GPIO port Pin value byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxPIN0	Fast GPIO Port x Pin value register 0. Bit 0 in FIOxPIN0 register corresponds to pin Px[0] ... bit 7 to pin Px[7].	8 (byte) R/W	0x00	FIO0PIN0 - 0x2009 8014 FIO1PIN0 - 0x2009 8034 FIO2PIN0 - 0x2009 8054 FIO3PIN0 - 0x2009 8074 FIO4PIN0 - 0x2009 8094 FIO5PIN0 - 0x2009 80B4
FIOxPIN1	Fast GPIO Port x Pin value register 1. Bit 0 in FIOxPIN1 register corresponds to pin Px[8] ... bit 7 to pin Px[15].	8 (byte) R/W	0x00	FIO0PIN1 - 0x2009 8015 FIO1PIN1 - 0x2009 8035 FIO2PIN1 - 0x2009 8055 FIO3PIN1 - 0x2009 8075 FIO4PIN1 - 0x2009 8095 FIO5PIN1 - 0x2009 80B5
FIOxPIN2	Fast GPIO Port x Pin value register 2. Bit 0 in FIOxPIN2 register corresponds to pin Px[16] ... bit 7 to pin Px[23].	8 (byte) R/W	0x00	FIO0PIN2 - 0x2009 8016 FIO1PIN2 - 0x2009 8036 FIO2PIN2 - 0x2009 8056 FIO3PIN2 - 0x2009 8076 FIO4PIN2 - 0x2009 8096 FIO5PIN2 - 0x2009 80B6

**Table 94. Fast GPIO port Pin value byte and half-word accessible register description**

Generic name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxPIN3	Fast GPIO Port x Pin value register 3. Bit 0 in FIOxPIN3 register corresponds to pin Px[24] ... bit 7 to pin Px[31].	8 (byte) R/W	0x00	FIO0PIN3 - 0x2009 8017 FIO1PIN3 - 0x2009 8037 FIO2PIN3 - 0x2009 8057 FIO3PIN3 - 0x2009 8077 FIO4PIN3 - 0x2009 8097 FIO5PIN3 - 0x2009 80B7
FIOxPINL	Fast GPIO Port x Pin value Lower half-word register. Bit 0 in FIOxPINL register corresponds to pin Px[0] ... bit 15 to pin Px[15].	16 (half-word) R/W	0x0000	FIO0PINL - 0x2009 8014 FIO1PINL - 0x2009 8034 FIO2PINL - 0x2009 8054 FIO3PINL - 0x2009 8074 FIO4PINL - 0x2009 8094 FIO5PINL - 0x2009 80B4
FIOxPINU	Fast GPIO Port x Pin value Upper half-word register. Bit 0 in FIOxPINU register corresponds to pin Px[16] ... bit 15 to Px[31].	16 (half-word) R/W	0x0000	FIO0PINU - 0x2009 8016 FIO1PINU - 0x2009 8036 FIO2PINU - 0x2009 8056 FIO3PINU - 0x2009 8076 FIO4PINU - 0x2009 8096 FIO5PINU - 0x2009 80B6

### 9.5.5 Fast GPIO port Mask register FIOxMASK (FIO0MASK to FIO5MASK - 0x2009 8010 to 0x2009 80B0)

This register is used to select port pins that will and will not be affected by write accesses to the FIOxPIN, FIOxSET or FIOxCLR register. Mask register also filters out port's content when the FIOxPIN register is read.

A zero in this register's bit enables an access to the corresponding physical pin via a read or write access. If a bit in this register is one, corresponding pin will not be changed with write access and if read, will not be reflected in the updated FIOxPIN register. For software examples, see [Section 9.6](#).

**Table 95. Fast GPIO port Mask register (FIO0MASK to FIO4MASK - addresses 0x2009 8010 to 0x2009 8090) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	FIO0MASK FIO1MASK FIO2MASK FIO3MASK FIO4MASK FIO5MASK	0	Fast GPIO physical pin access control. Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.	0x0
		1	Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.	

Aside from the 32-bit long and word only accessible FIOxMASK register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in [Table 96](#), too. Next to providing the same functions as the FIOxMASK register, these additional registers allow easier and faster access to the physical port pins.

**Table 96. Fast GPIO port Mask byte and half-word accessible register description**

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxMASK0	Fast GPIO Port x Mask register 0. Bit 0 in FIOxMASK0 register corresponds to pin Px[0] ... bit 7 to pin Px[7].	8 (byte) R/W	0x0	FIO0MASK0 - 0x2009 8010 FIO1MASK0 - 0x2009 8030 FIO2MASK0 - 0x2009 8050 FIO3MASK0 - 0x2009 8070 FIO4MASK0 - 0x2009 8090 FIO5MASK0 - 0x2009 80B0
FIOxMASK1	Fast GPIO Port x Mask register 1. Bit 0 in FIOxMASK1 register corresponds to pin Px[8] ... bit 7 to pin Px[15].	8 (byte) R/W	0x0	FIO0MASK1 - 0x2009 8011 FIO1MASK1 - 0x2009 8031 FIO2MASK1 - 0x2009 8051 FIO3MASK1 - 0x2009 8071 FIO4MASK1 - 0x2009 8091 FIO5MASK1 - 0x2009 80B1
FIOxMASK2	Fast GPIO Port x Mask register 2. Bit 0 in FIOxMASK2 register corresponds to pin Px[16] ... bit 7 to pin Px[23].	8 (byte) R/W	0x0	FIO0MASK2 - 0x2009 8012 FIO1MASK2 - 0x2009 8032 FIO2MASK2 - 0x2009 8052 FIO3MASK2 - 0x2009 8072 FIO4MASK2 - 0x2009 8092 FIO5MASK2 - 0x2009 80B2
FIOxMASK3	Fast GPIO Port x Mask register 3. Bit 0 in FIOxMASK3 register corresponds to pin Px[24] ... bit 7 to pin Px[31].	8 (byte) R/W	0x0	FIO0MASK3 - 0x2009 8013 FIO1MASK3 - 0x2009 8033 FIO2MASK3 - 0x2009 8053 FIO3MASK3 - 0x2009 8073 FIO4MASK3 - 0x2009 8093 FIO5MASK3 - 0x2009 80B3
FIOxMASKL	Fast GPIO Port x Mask Lower half-word register. Bit 0 in FIOxMASKL register corresponds to pin Px[0] ... bit 15 to pin Px[15].	16 (half-word) R/W	0x0	FIO0MASKL - 0x2009 8010 FIO1MASKL - 0x2009 8030 FIO2MASKL - 0x2009 8050 FIO3MASKL - 0x2009 8070 FIO4MASKL - 0x2009 8090 FIO5MASKL - 0x2009 80B0
FIOxMASKU	Fast GPIO Port x Mask Upper half-word register. Bit 0 in FIOxMASKU register corresponds to pin Px[16] ... bit 15 to Px[31].	16 (half-word) R/W	0x0	FIO0MASKU - 0x2009 8012 FIO1MASKU - 0x2009 8032 FIO2MASKU - 0x2009 8052 FIO3MASKU - 0x2009 8072 FIO4MASKU - 0x2009 8092 FIO5MASKU - 0x2009 80B2

**9.5.6 GPIO interrupt registers**

The following registers configure the pins of Port 0 and Port 2 to generate interrupts.

**9.5.6.1 GPIO overall Interrupt Status register (IOIntStatus - 0x4002 8080)**

This read-only register indicates the presence of interrupt pending on all of the GPIO ports that support GPIO interrupts. Only status one bit per port is required.

**Table 97. GPIO overall Interrupt Status register (IOIntStatus - address 0x4002 8080) bit description**

Bit	Symbol	Value	Description	Reset value
0	P0Int		Port 0 GPIO interrupt pending.	0
		0	There are no pending interrupts on Port 0.	
		1	There is at least one pending interrupt on Port 0.	
1	-		Reserved. The value read from a reserved bit is not defined.	NA
2	P2Int		Port 2 GPIO interrupt pending.	0
		0	There are no pending interrupts on Port 2.	
		1	There is at least one pending interrupt on Port 2.	
31:2	-		Reserved. The value read from a reserved bit is not defined.	NA

9.5.6.2 GPIO Interrupt Enable for port 0 Rising Edge (IO0IntEnR - 0x4002 8090)

Each bit in these read-write registers enables the rising edge interrupt for the corresponding port 0 pin.

Table 98. GPIO Interrupt Enable for port 0 Rising Edge (IO0IntEnR - 0x4002 8090) bit description

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P0.0ER		Enable rising edge interrupt for P0[0].	0
		0	Rising edge interrupt is disabled on P0[0].	
		1	Rising edge interrupt is enabled on P0[0].	
1	P0.1ER		Enable rising edge interrupt for P0[1].	0
2	P0.2ER		Enable rising edge interrupt for P0[2].	0
3	P0.3ER		Enable rising edge interrupt for P0[3].	0
4	P0.4ER		Enable rising edge interrupt for P0[4].	0
5	P0.5ER		Enable rising edge interrupt for P0[5].	0
6	P0.6ER		Enable rising edge interrupt for P0[6].	0
7	P0.7ER		Enable rising edge interrupt for P0[7].	0
8	P0.8ER		Enable rising edge interrupt for P0[8].	0
9	P0.9ER		Enable rising edge interrupt for P0[9].	0
10	P0.10ER		Enable rising edge interrupt for P0[10].	0
11	P0.11ER		Enable rising edge interrupt for P0[11].	0
12	P0.12ER		Enable rising edge interrupt for P0[12].	0
13	P0.13ER		Enable rising edge interrupt for P0[13].	0
14	P0.14ER		Enable rising edge interrupt for P0[14].	0
15	P0.15ER		Enable rising edge interrupt for P0[15].	0
16	P0.16ER		Enable rising edge interrupt for P0[16].	0
17	P0.17ER		Enable rising edge interrupt for P0[17].	0
18	P0.18ER		Enable rising edge interrupt for P0[18].	0
19	P0.19ER		Enable rising edge interrupt for P0[19].	0
20	P0.20ER		Enable rising edge interrupt for P0[20].	0
21	P0.21ER		Enable rising edge interrupt for P0[21].	0
22	P0.22ER		Enable rising edge interrupt for P0[22].	0
23	P0.23ER		Enable rising edge interrupt for P0[23].	0
24	P0.24ER		Enable rising edge interrupt for P0[24].	0
25	P0.25ER		Enable rising edge interrupt for P0[25].	0
26	P0.26ER		Enable rising edge interrupt for P0[26].	0
27	P0.27ER		Enable rising edge interrupt for P0[27].	0
28	P0.28ER		Enable rising edge interrupt for P0[28].	0
29	P0.29ER		Enable rising edge interrupt for P0[29].	0
30	P0.30ER		Enable rising edge interrupt for P0[30].	0
31	P0.31ER		Enable rising edge interrupt for P0[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

9.5.6.3 GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR - 0x4002 80B0)

Each bit in these read-write registers enables the rising edge interrupt for the corresponding port 2 pin.

Table 99. GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR - 0x4002 80B0) bit description

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P2.0ER		Enable rising edge interrupt for P2[0].	0
		0	Rising edge interrupt is disabled on P2[0].	
		1	Rising edge interrupt is enabled on P2[0].	
1	P2.1ER		Enable rising edge interrupt for P2[1].	0
2	P2.2ER		Enable rising edge interrupt for P2[2].	0
3	P2.3ER		Enable rising edge interrupt for P2[3].	0
4	P2.4ER		Enable rising edge interrupt for P2[4].	0
5	P2.5ER		Enable rising edge interrupt for P2[5].	0
6	P2.6ER		Enable rising edge interrupt for P2[6].	0
7	P2.7ER		Enable rising edge interrupt for P2[7].	0
8	P2.8ER		Enable rising edge interrupt for P2[8].	0
9	P2.9ER		Enable rising edge interrupt for P2[9].	0
10	P2.10ER		Enable rising edge interrupt for P2[10].	0
11	P2.11ER		Enable rising edge interrupt for P2[11].	0
12	P2.12ER		Enable rising edge interrupt for P2[12].	0
13	P2.13ER		Enable rising edge interrupt for P2[13].	0
14	P2.14ER		Enable rising edge interrupt for P2[14].	0
15	P2.15ER		Enable rising edge interrupt for P2[15].	0
16	P2.16ER		Enable rising edge interrupt for P2[16].	0
17	P2.17ER		Enable rising edge interrupt for P2[17].	0
18	P2.18ER		Enable rising edge interrupt for P2[18].	0
19	P2.19ER		Enable rising edge interrupt for P2[19].	0
20	P2.20ER		Enable rising edge interrupt for P2[20].	0
21	P2.21ER		Enable rising edge interrupt for P2[21].	0
22	P2.22ER		Enable rising edge interrupt for P2[22].	0
23	P2.23ER		Enable rising edge interrupt for P2[23].	0
24	P2.24ER		Enable rising edge interrupt for P2[24].	0
25	P2.25ER		Enable rising edge interrupt for P2[25].	0
26	P2.26ER		Enable rising edge interrupt for P2[26].	0
27	P2.27ER		Enable rising edge interrupt for P2[27].	0
28	P2.28ER		Enable rising edge interrupt for P2[28].	0
29	P2.29ER		Enable rising edge interrupt for P2[29].	0
30	P2.30ER		Enable rising edge interrupt for P2[30].	0
31	P2.31ER		Enable rising edge interrupt for P2[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

9.5.6.4 GPIO Interrupt Enable for port 0 Falling Edge (IO0IntEnF - 0x4002 8094)

Each bit in these read-write registers enables the falling edge interrupt for the corresponding GPIO port 0 pin.

Table 100. GPIO Interrupt Enable for port 0 Falling Edge (IO0IntEnF - address 0x4002 8094) bit description

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P0.0EF		Enable falling edge interrupt for P0[0].	0
		0	Falling edge interrupt is disabled on P0[0].	
		1	Falling edge interrupt is enabled on P0[0].	
1	P0.1EF		Enable falling edge interrupt for P0[1].	0
2	P0.2EF		Enable falling edge interrupt for P0[2].	0
3	P0.3EF		Enable falling edge interrupt for P0[3].	0
4	P0.4EF		Enable falling edge interrupt for P0[4].	0
5	P0.5EF		Enable falling edge interrupt for P0[5].	0
6	P0.6EF		Enable falling edge interrupt for P0[6].	0
7	P0.7EF		Enable falling edge interrupt for P0[7].	0
8	P0.8EF		Enable falling edge interrupt for P0[8].	0
9	P0.9EF		Enable falling edge interrupt for P0[9].	0
10	P0.10EF		Enable falling edge interrupt for P0[10].	0
11	P0.11EF		Enable falling edge interrupt for P0[11].	0
12	P0.12EF		Enable falling edge interrupt for P0[12].	0
13	P0.13EF		Enable falling edge interrupt for P0[13].	0
14	P0.14EF		Enable falling edge interrupt for P0[14].	0
15	P0.15EF		Enable falling edge interrupt for P0[15].	0
16	P0.16EF		Enable falling edge interrupt for P0[16].	0
17	P0.17EF		Enable falling edge interrupt for P0[17].	0
18	P0.18EF		Enable falling edge interrupt for P0[18].	0
19	P0.19EF		Enable falling edge interrupt for P0[19].	0
20	P0.20EF		Enable falling edge interrupt for P0[20].	0
21	P0.21EF		Enable falling edge interrupt for P0[21].	0
22	P0.22EF		Enable falling edge interrupt for P0[22].	0
23	P0.23EF		Enable falling edge interrupt for P0[23].	0
24	P0.24EF		Enable falling edge interrupt for P0[24].	0
25	P0.25EF		Enable falling edge interrupt for P0[25].	0
26	P0.26EF		Enable falling edge interrupt for P0[26].	0
27	P0.27EF		Enable falling edge interrupt for P0[27].	0
28	P0.28EF		Enable falling edge interrupt for P0[28].	0
29	P0.29EF		Enable falling edge interrupt for P0[29].	0
30	P0.30EF		Enable falling edge interrupt for P0[30].	0
31	P0.31EF		Enable falling edge interrupt for P0[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

9.5.6.5 GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF - 0x4002 80B4)

Each bit in these read-write registers enables the falling edge interrupt for the corresponding GPIO port 2 pin.

Table 101. GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF - 0x4002 80B4) bit description

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P2.0EF		Enable falling edge interrupt for P2[0].	0
		0	Falling edge interrupt is disabled on P2[0].	
		1	Falling edge interrupt is enabled on P2[0].	
1	P2.1EF		Enable falling edge interrupt for P2[1].	0
2	P2.2EF		Enable falling edge interrupt for P2[2].	0
3	P2.3EF		Enable falling edge interrupt for P2[3].	0
4	P2.4EF		Enable falling edge interrupt for P2[4].	0
5	P2.5EF		Enable falling edge interrupt for P2[5].	0
6	P2.6EF		Enable falling edge interrupt for P2[6].	0
7	P2.7EF		Enable falling edge interrupt for P2[7].	0
8	P2.8EF		Enable falling edge interrupt for P2[8].	0
9	P2.9EF		Enable falling edge interrupt for P2[9].	0
10	P2.10EF		Enable falling edge interrupt for P2[10].	0
11	P2.11EF		Enable falling edge interrupt for P2[11].	0
12	P2.12EF		Enable falling edge interrupt for P2[12].	0
13	P2.13EF		Enable falling edge interrupt for P2[13].	0
14	P2.14EF		Enable falling edge interrupt for P2[14].	0
15	P2.15EF		Enable falling edge interrupt for P2[15].	0
16	P2.16EF		Enable falling edge interrupt for P2[16].	0
17	P2.17EF		Enable falling edge interrupt for P2[17].	0
18	P2.18EF		Enable falling edge interrupt for P2[18].	0
19	P2.19EF		Enable falling edge interrupt for P2[19].	0
20	P2.20EF		Enable falling edge interrupt for P2[20].	0
21	P2.21EF		Enable falling edge interrupt for P2[21].	0
22	P2.22EF		Enable falling edge interrupt for P2[22].	0
23	P2.23EF		Enable falling edge interrupt for P2[23].	0
24	P2.24EF		Enable falling edge interrupt for P2[24].	0
25	P2.25EF		Enable falling edge interrupt for P2[25].	0
26	P2.26EF		Enable falling edge interrupt for P2[26].	0
27	P2.27EF		Enable falling edge interrupt for P2[27].	0
28	P2.28EF		Enable falling edge interrupt for P2[28].	0
29	P2.29EF		Enable falling edge interrupt for P2[29].	0
30	P2.30EF		Enable falling edge interrupt for P2[30].	0
31	P2.31EF		Enable falling edge interrupt for P2[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.



**9.5.6.6 GPIO Interrupt Status for port 0 Rising Edge Interrupt (IO0IntStatR - 0x4002 8084)**

Each bit in these read-only registers indicates the rising edge interrupt status for port 0.

**Table 102. GPIO Interrupt Status for port 0 Rising Edge Interrupt (IO0IntStatR - 0x4002 8084) bit description**

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P0.0REI		Status of Rising Edge Interrupt for P0[0].	0
		0	A rising edge has not been detected on P0[0].	
		1	Interrupt has been generated due to a rising edge on P0[0].	
1	P0.1REI		Status of Rising Edge Interrupt for P0[1].	0
2	P0.2REI		Status of Rising Edge Interrupt for P0[2].	0
3	P0.3REI		Status of Rising Edge Interrupt for P0[3].	0
4	P0.4REI		Status of Rising Edge Interrupt for P0[4].	0
5	P0.5REI		Status of Rising Edge Interrupt for P0[5].	0
6	P0.6REI		Status of Rising Edge Interrupt for P0[6].	0
7	P0.7REI		Status of Rising Edge Interrupt for P0[7].	0
8	P0.8REI		Status of Rising Edge Interrupt for P0[8].	0
9	P0.9REI		Status of Rising Edge Interrupt for P0[9].	0
10	P0.10REI		Status of Rising Edge Interrupt for P0[10].	0
11	P0.11REI		Status of Rising Edge Interrupt for P0[11].	0
12	P0.12REI		Status of Rising Edge Interrupt for P0[12].	0
13	P0.13REI		Status of Rising Edge Interrupt for P0[13].	0
14	P0.14REI		Status of Rising Edge Interrupt for P0[14].	0
15	P0.15REI		Status of Rising Edge Interrupt for P0[15].	0
16	P0.16REI		Status of Rising Edge Interrupt for P0[16].	0
17	P0.17REI		Status of Rising Edge Interrupt for P0[17].	0
18	P0.18REI		Status of Rising Edge Interrupt for P0[18].	0
19	P0.19REI		Status of Rising Edge Interrupt for P0[19].	0
20	P0.20REI		Status of Rising Edge Interrupt for P0[20].	0
21	P0.21REI		Status of Rising Edge Interrupt for P0[21].	0
22	P0.22REI		Status of Rising Edge Interrupt for P0[22].	0
23	P0.23REI		Status of Rising Edge Interrupt for P0[23].	0
24	P0.24REI		Status of Rising Edge Interrupt for P0[24].	0
25	P0.25REI		Status of Rising Edge Interrupt for P0[25].	0
26	P0.26REI		Status of Rising Edge Interrupt for P0[26].	0
27	P0.27REI		Status of Rising Edge Interrupt for P0[27].	0
28	P0.28REI		Status of Rising Edge Interrupt for P0[28].	0
29	P0.29REI		Status of Rising Edge Interrupt for P0[29].	0
30	P0.30REI		Status of Rising Edge Interrupt for P0[30].	0
31	P0.31REI		Status of Rising Edge Interrupt for P0[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

**9.5.6.7 GPIO Interrupt Status for port 2 Rising Edge Interrupt (IO2IntStatR - 0x4002 80A4)**

Each bit in these read-only registers indicates the rising edge interrupt status for port 2.

**Table 103. GPIO Interrupt Status for port 2 Rising Edge Interrupt (IO2IntStatR - 0x4002 80A4) bit description**

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P2.0REI		Status of Rising Edge Interrupt for P2[0].	0
		0	A rising edge has not been detected on P2[0].	
		1	Interrupt has been generated due to a rising edge on P2[0].	
1	P2.1REI		Status of Rising Edge Interrupt for P2[1].	0
2	P2.2REI		Status of Rising Edge Interrupt for P2[2].	0
3	P2.3REI		Status of Rising Edge Interrupt for P2[3].	0
4	P2.4REI		Status of Rising Edge Interrupt for P2[4].	0
5	P2.5REI		Status of Rising Edge Interrupt for P2[5].	0
6	P2.6REI		Status of Rising Edge Interrupt for P2[6].	0
7	P2.7REI		Status of Rising Edge Interrupt for P2[7].	0
8	P2.8REI		Status of Rising Edge Interrupt for P2[8].	0
9	P2.9REI		Status of Rising Edge Interrupt for P2[9].	0
10	P2.10REI		Status of Rising Edge Interrupt for P2[10].	0
11	P2.11REI		Status of Rising Edge Interrupt for P2[11].	0
12	P2.12REI		Status of Rising Edge Interrupt for P2[12].	0
13	P2.13REI		Status of Rising Edge Interrupt for P2[13].	0
14	P2.14REI		Status of Rising Edge Interrupt for P2[14].	0
15	P2.15REI		Status of Rising Edge Interrupt for P2[15].	0
16	P2.16REI		Status of Rising Edge Interrupt for P2[16].	0
17	P2.17REI		Status of Rising Edge Interrupt for P2[17].	0
18	P2.18REI		Status of Rising Edge Interrupt for P2[18].	0
19	P2.19REI		Status of Rising Edge Interrupt for P2[19].	0
20	P2.20REI		Status of Rising Edge Interrupt for P2[20].	0
21	P2.21REI		Status of Rising Edge Interrupt for P2[21].	0
22	P2.22REI		Status of Rising Edge Interrupt for P2[22].	0
23	P2.23REI		Status of Rising Edge Interrupt for P2[23].	0
24	P2.24REI		Status of Rising Edge Interrupt for P2[24].	0
25	P2.25REI		Status of Rising Edge Interrupt for P2[25].	0
26	P2.26REI		Status of Rising Edge Interrupt for P2[26].	0
27	P2.27REI		Status of Rising Edge Interrupt for P2[27].	0
28	P2.28REI		Status of Rising Edge Interrupt for P2[28].	0
29	P2.29REI		Status of Rising Edge Interrupt for P2[29].	0
30	P2.30REI		Status of Rising Edge Interrupt for P2[30].	0
31	P2.31REI		Status of Rising Edge Interrupt for P2[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

**9.5.6.8 GPIO Interrupt Status for port 0 Falling Edge Interrupt (IO0IntStatF - 0x4002 8088)**

Each bit in these read-only registers indicates the falling edge interrupt status for port 0.

**Table 104. GPIO Interrupt Status for port 0 Falling Edge Interrupt (IO0IntStatF - 0x4002 8088) bit description**

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P0.0FEI		Status of Falling Edge Interrupt for P0[0].	0
		0	A falling edge has not been detected on P0[0].	
		1	Interrupt has been generated due to a falling edge on P0[0].	
1	P0.1FEI		Status of Falling Edge Interrupt for P0[1].	0
2	P0.2FEI		Status of Falling Edge Interrupt for P0[2].	0
3	P0.3FEI		Status of Falling Edge Interrupt for P0[3].	0
4	P0.4FEI		Status of Falling Edge Interrupt for P0[4].	0
5	P0.5FEI		Status of Falling Edge Interrupt for P0[5].	0
6	P0.6FEI		Status of Falling Edge Interrupt for P0[6].	0
7	P0.7FEI		Status of Falling Edge Interrupt for P0[7].	0
8	P0.8FEI		Status of Falling Edge Interrupt for P0[8].	0
9	P0.9FEI		Status of Falling Edge Interrupt for P0[9].	0
10	P0.10FEI		Status of Falling Edge Interrupt for P0[10].	0
11	P0.11FEI		Status of Falling Edge Interrupt for P0[11].	0
12	P0.12FEI		Status of Falling Edge Interrupt for P0[12].	0
13	P0.13FEI		Status of Falling Edge Interrupt for P0[13].	0
14	P0.14FEI		Status of Falling Edge Interrupt for P0[14].	0
15	P0.15FEI		Status of Falling Edge Interrupt for P0[15].	0
16	P0.16FEI		Status of Falling Edge Interrupt for P0[16].	0
17	P0.17FEI		Status of Falling Edge Interrupt for P0[17].	0
18	P0.18FEI		Status of Falling Edge Interrupt for P0[18].	0
19	P0.19FEI		Status of Falling Edge Interrupt for P0[19].	0
20	P0.20FEI		Status of Falling Edge Interrupt for P0[20].	0
21	P0.21FEI		Status of Falling Edge Interrupt for P0[21].	0
22	P0.22FEI		Status of Falling Edge Interrupt for P0[22].	0
23	P0.23FEI		Status of Falling Edge Interrupt for P0[23].	0
24	P0.24FEI		Status of Falling Edge Interrupt for P0[24].	0
25	P0.25FEI		Status of Falling Edge Interrupt for P0[25].	0
26	P0.26FEI		Status of Falling Edge Interrupt for P0[26].	0
27	P0.27FEI		Status of Falling Edge Interrupt for P0[27].	0
28	P0.28FEI		Status of Falling Edge Interrupt for P0[28].	0
29	P0.29FEI		Status of Falling Edge Interrupt for P0[29].	0
30	P0.30FEI		Status of Falling Edge Interrupt for P0[30].	0
31	P0.31FEI		Status of Falling Edge Interrupt for P0[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

**9.5.6.9 GPIO Interrupt Status for port 2 Falling Edge Interrupt (IO2IntStatF - 0x4002 80A8)**

Each bit in these read-only registers indicates the falling edge interrupt status for port 2.

**Table 105. GPIO Interrupt Status for port 2 Falling Edge Interrupt (IO2IntStatF - 0x4002 80A8) bit description**

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P2.0FEI		Status of Falling Edge Interrupt for P2[0].	0
		0	A falling edge has not been detected on P2[0].	
		1	Interrupt has been generated due to a falling edge on P2[0].	
1	P2.1FEI		Status of Falling Edge Interrupt for P2[1].	0
2	P2.2FEI		Status of Falling Edge Interrupt for P2[2].	0
3	P2.3FEI		Status of Falling Edge Interrupt for P2[3].	0
4	P2.4FEI		Status of Falling Edge Interrupt for P2[4].	0
5	P2.5FEI		Status of Falling Edge Interrupt for P2[5].	0
6	P2.6FEI		Status of Falling Edge Interrupt for P2[6].	0
7	P2.7FEI		Status of Falling Edge Interrupt for P2[7].	0
8	P2.8FEI		Status of Falling Edge Interrupt for P2[8].	0
9	P2.9FEI		Status of Falling Edge Interrupt for P2[9].	0
10	P2.10FEI		Status of Falling Edge Interrupt for P2[10].	0
11	P2.11FEI		Status of Falling Edge Interrupt for P2[11].	0
12	P2.12FEI		Status of Falling Edge Interrupt for P2[12].	0
13	P2.13FEI		Status of Falling Edge Interrupt for P2[13].	0
14	P2.14FEI		Status of Falling Edge Interrupt for P2[14].	0
15	P2.15FEI		Status of Falling Edge Interrupt for P2[15].	0
16	P2.16FEI		Status of Falling Edge Interrupt for P2[16].	0
17	P2.17FEI		Status of Falling Edge Interrupt for P2[17].	0
18	P2.18FEI		Status of Falling Edge Interrupt for P2[18].	0
19	P2.19FEI		Status of Falling Edge Interrupt for P2[19].	0
20	P2.20FEI		Status of Falling Edge Interrupt for P2[20].	0
21	P2.21FEI		Status of Falling Edge Interrupt for P2[21].	0
22	P2.22FEI		Status of Falling Edge Interrupt for P2[22].	0
23	P2.23FEI		Status of Falling Edge Interrupt for P2[23].	0
24	P2.24FEI		Status of Falling Edge Interrupt for P2[24].	0
25	P2.25FEI		Status of Falling Edge Interrupt for P2[25].	0
26	P2.26FEI		Status of Falling Edge Interrupt for P2[26].	0
27	P2.27FEI		Status of Falling Edge Interrupt for P2[27].	0
28	P2.28FEI		Status of Falling Edge Interrupt for P2[28].	0
29	P2.29FEI		Status of Falling Edge Interrupt for P2[29].	0
30	P2.30FEI		Status of Falling Edge Interrupt for P2[30].	0
31	P2.31FEI		Status of Falling Edge Interrupt for P2[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

### 9.5.6.10 GPIO Interrupt Clear register for port 0 (IO0IntClr - 0x4002 808C)

Writing a 1 into a bit in this write-only register clears any interrupts for the corresponding port 0 pin.

**Table 106. GPIO Interrupt Clear register for port 0 (IO0IntClr - 0x4002 808C) bit description**

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P0.0CI		Clear GPIO port Interrupts for P0[0].	0
		0	Corresponding bits in IOxIntStatR and IOxIntStatF are unchanged.	
		1	Corresponding bits in IOxIntStatR and IOxStatF are cleared.	
1	P0.1CI		Clear GPIO port Interrupts for P0[1].	0
2	P0.2CI		Clear GPIO port Interrupts for P0[2].	0
3	P0.3CI		Clear GPIO port Interrupts for P0[3].	0
4	P0.4CI		Clear GPIO port Interrupts for P0[4].	0
5	P0.5CI		Clear GPIO port Interrupts for P0[5].	0
6	P0.6CI		Clear GPIO port Interrupts for P0[6].	0
7	P0.7CI		Clear GPIO port Interrupts for P0[7].	0
8	P0.8CI		Clear GPIO port Interrupts for P0[8].	0
9	P0.9CI		Clear GPIO port Interrupts for P0[9].	0
10	P0.10CI		Clear GPIO port Interrupts for P0[10].	0
11	P0.11CI		Clear GPIO port Interrupts for P0[11].	0
12	P0.12CI		Clear GPIO port Interrupts for P0[12].	0
13	P0.13CI		Clear GPIO port Interrupts for P0[13].	0
14	P0.14CI		Clear GPIO port Interrupts for P0[14].	0
15	P0.15CI		Clear GPIO port Interrupts for P0[15].	0
16	P0.16CI		Clear GPIO port Interrupts for P0[16].	0
17	P0.17CI		Clear GPIO port Interrupts for P0[17].	0
18	P0.18CI		Clear GPIO port Interrupts for P0[18].	0
19	P0.19CI		Clear GPIO port Interrupts for P0[19].	0
20	P0.20CI		Clear GPIO port Interrupts for P0[20].	0
21	P0.21CI		Clear GPIO port Interrupts for P0[21].	0
22	P0.22CI		Clear GPIO port Interrupts for P0[22].	0
23	P0.23CI		Clear GPIO port Interrupts for P0[23].	0
24	P0.24CI		Clear GPIO port Interrupts for P0[24].	0
25	P0.25CI		Clear GPIO port Interrupts for P0[25].	0
26	P0.26CI		Clear GPIO port Interrupts for P0[26].	0
27	P0.27CI		Clear GPIO port Interrupts for P0[27].	0
28	P0.28CI		Clear GPIO port Interrupts for P0[28].	0
29	P0.29CI		Clear GPIO port Interrupts for P0[29].	0
30	P0.30CI		Clear GPIO port Interrupts for P0[30].	0
31	P0.31CI		Clear GPIO port Interrupts for P0[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

9.5.6.11 GPIO Interrupt Clear register for port 0 (IO2IntClr - 0x4002 80AC)

Writing a 1 into a bit in this write-only register clears any interrupts for the corresponding port 2 pin.

Table 107. GPIO Interrupt Clear register for port 0 (IO2IntClr - 0x4002 80AC) bit description

Bit	Symbol	Value	Description <sup>[1]</sup>	Reset value
0	P2.0CI		Clear GPIO port Interrupts for P2[0].	0
		0	Corresponding bits in IOxIntStatR and IOxIntStatF are unchanged.	
		1	Corresponding bits in IOxIntStatR and IOxStatF are cleared.	
1	P2.1CI		Clear GPIO port Interrupts for P2[1].	0
2	P2.2CI		Clear GPIO port Interrupts for P2[2].	0
3	P2.3CI		Clear GPIO port Interrupts for P2[3].	0
4	P2.4CI		Clear GPIO port Interrupts for P2[4].	0
5	P2.5CI		Clear GPIO port Interrupts for P2[5].	0
6	P2.6CI		Clear GPIO port Interrupts for P2[6].	0
7	P2.7CI		Clear GPIO port Interrupts for P2[7].	0
8	P2.8CI		Clear GPIO port Interrupts for P2[8].	0
9	P2.9CI		Clear GPIO port Interrupts for P2[9].	0
10	P2.10CI		Clear GPIO port Interrupts for P2[10].	0
11	P2.11CI		Clear GPIO port Interrupts for P2[11].	0
12	P2.12CI		Clear GPIO port Interrupts for P2[12].	0
13	P2.13CI		Clear GPIO port Interrupts for P2[13].	0
14	P2.14CI		Clear GPIO port Interrupts for P2[14].	0
15	P2.15CI		Clear GPIO port Interrupts for P2[15].	0
16	P2.16CI		Clear GPIO port Interrupts for P2[16].	0
17	P2.17CI		Clear GPIO port Interrupts for P2[17].	0
18	P2.18CI		Clear GPIO port Interrupts for P2[18].	0
19	P2.19CI		Clear GPIO port Interrupts for P2[19].	0
20	P2.20CI		Clear GPIO port Interrupts for P2[20].	0
21	P2.21CI		Clear GPIO port Interrupts for P2[21].	0
22	P2.22CI		Clear GPIO port Interrupts for P2[22].	0
23	P2.23CI		Clear GPIO port Interrupts for P2[23].	0
24	P2.24CI		Clear GPIO port Interrupts for P2[24].	0
25	P2.25CI		Clear GPIO port Interrupts for P2[25].	0
26	P2.26CI		Clear GPIO port Interrupts for P2[26].	0
27	P2.27CI		Clear GPIO port Interrupts for P2[27].	0
28	P2.28CI		Clear GPIO port Interrupts for P2[28].	0
29	P2.29CI		Clear GPIO port Interrupts for P2[29].	0
30	P2.30CI		Clear GPIO port Interrupts for P2[30].	0
31	P2.31CI		Clear GPIO port Interrupts for P2[31].	0

[1] Which pins are available depends on the part number and package combination. See the specific device data sheet for details.

## 9.6 GPIO usage notes

### 9.6.1 Example: An instantaneous output of 0s and 1s on a GPIO port

**Solution 1:** using 32-bit (word) accessible fast GPIO registers

```
FIOOMASK = 0xFFFF00FF ;  
FIOOPIN  = 0x0000A500;
```

**Solution 2:** using 16-bit (half-word) accessible fast GPIO registers

```
FIOOMASKL = 0x00FF;  
FIOOPINL  = 0xA500;
```

**Solution 3:** using 8-bit (byte) accessible fast GPIO registers

```
FIOOPIN1  = 0xA5;
```

### 9.6.2 Writing to FIOSET/FIOCLR vs. FIOPIN

Writing to the FIOSET/FIOCLR registers allow a program to easily change a port's output pin(s) to both high and low levels at the same time. When FIOSET or FIOCLR are used, only pin/bit(s) written with 1 will be changed, while those written as 0 will remain unaffected.

Writing to the FIOPIN register enables instantaneous output of a desired value on the parallel GPIO. Data written to the FIOPIN register will affect all pins configured as outputs on that port: zeroes in the value will produce low level pin outputs and ones in the value will produce high level pin outputs.

A subset of a port's pins may be changed by using the FIOMASK register to define which pins are affected. FIOMASK is set up to contain zeroes in bits corresponding to pins that will be changed, and ones for all others. Solution 2 from [Section 9.6.1](#) above illustrates output of 0xA5 on PORT0 pins 15 to 8 while preserving all other PORT0 output pins as they were before.

### 10.1 How to read this chapter

This chapter describes the external memory controller for LP178x/7x devices that support external memory. EMC configurations vary with different packages for devices that support external memory, see [Table 108](#).

**Table 108. EMC configuration**

Device package	Data bus widths supported	Pins available	Dynamic memory configuration registers <sup>[1][2]</sup>	Static memory configuration registers <sup>[1][3]</sup>	External memory connections
144-pin	8-bit	EMC_A[15:0] EMC_D[7:0] $\overline{\text{EMC\_OE}}$ $\overline{\text{EMC\_WE}}$ $\overline{\text{EMC\_CS}}[1:0]$		EMCStaticConfig1/0 EMCStaticWaitWen1/0 EMCStaticWaitOen1/0 EMCStaticWaitRd1/0 EMCStaticWaitPage1/0 EMCStaticWaitWr1/0 EMCStaticWaitTurn1/0	<a href="#">Section 10.13.3</a>
180-pin	16-bit, 8-bit	EMC_A[19:0] EMC_D[15:0] $\overline{\text{EMC\_OE}}$ $\overline{\text{EMC\_WE}}$ $\overline{\text{EMC\_BLS}}[1:0]$ $\overline{\text{EMC\_CS}}[1:0]$ $\overline{\text{EMC\_DYCS}}[1:0]$ EMC_CAS $\overline{\text{EMC\_RAS}}$ EMC_CLK[1:0] EMC_CKE[1:0] EMC_DQM[1:0]	EMCDynamicConfig1/0 EMCDynamicRasCas1/0	EMCStaticConfig1/0 EMCStaticWaitWen1/0 EMCStaticWaitOen1/0 EMCStaticWaitRd1/0 EMCStaticWaitPage1/0 EMCStaticWaitWr1/0 EMCStaticWaitTurn1/0	<a href="#">Section 10.13.2</a> <a href="#">Section 10.13.3</a>
208-pin	32-bit, 16-bit, 8-bit	EMC_A[25:0] EMC_D[31:0] $\overline{\text{EMC\_OE}}$ $\overline{\text{EMC\_WE}}$ $\overline{\text{EMC\_BLS}}[3:0]$ $\overline{\text{EMC\_CS}}[3:0]$ $\overline{\text{EMC\_DYCS}}[3:0]$ $\overline{\text{EMC\_CAS}}$ $\overline{\text{EMC\_RAS}}$ EMC_CLK[1:0] EMC_CKE[3:0] EMC_DQM[3:0]	EMCDynamicConfig3/2/1/0 EMCDynamicRasCas3/2/1/0	EMCStaticConfig3/2/1/0 EMCStaticWaitWen3/2/1/0 EMCStaticWaitOen3/2/1/0 EMCStaticWaitRd3/2/1/0 EMCStaticWaitPage3/2/1/0 EMCStaticWaitWr3/2/1/0 EMCStaticWaitTurn3/2/1/0	<a href="#">Section 10.13.1</a> <a href="#">Section 10.13.2</a> <a href="#">Section 10.13.3</a>

[1] In addition to the registers that are common to all EMC operations: EMCControl and EMCConfig.



- [2] In addition to the registers that are common to all EMC dynamic chip selects: EMCDynamicControl, EMCDynamicRefresh, EMCDynamicReadConfig, EMCDynamicRP, EMCDynamicRAS, EMCDynamicSREX, EMCDynamicAPR, EMCDynamicDAL, EMCDynamicWR, EMCDynamicRC, EMCDynamicRFC, EMCDynamicXSR, EMCDynamicRRD, and EMCDynamicMRD
- [3] In addition to the EMCStaticExtendedWait register which applies to all static chip selects.

## 10.2 Basic configuration

The EMC is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCEMC.  
**Remark:** The EMC is enabled on reset (PCEMC = 1). On POR and warm reset, the EMC is enabled as well, see [Table 112](#) and [Table 115](#).
2. Clock: The EMC clock can be the same as the CPU clock (the default), or half that. The lower rate is intended to be used primarily when the CPU is running faster than the external bus can support. Clock selection for the EMC is described in [Section 4.6.3](#).
3. Pins: Select EMC pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).
4. Configuration: See [Table 112](#) to [Table 115](#). Also see additional EMC configurations in [Section 3.8.1 “System Controls and Status register \(SCS - 0x400F C1A0\)”](#). In particular make sure that the address shift mode is configured correctly for the application hardware.
5. MPU: Default memory space permissions for the Cortex-M3 do not allow program execution from the address range that includes the dynamic memory chip selects. These permissions can be changed by programming the MPU (see [Section 38.4.5 “Memory protection unit”](#)).

## 10.3 Introduction

The LPC178x/177x External Memory Controller (EMC) is an ARM PrimeCell™ MultiPort Memory Controller peripheral offering support for asynchronous static memory devices such as RAM, ROM and Flash, as well as dynamic memories such as Single Data Rate SDRAM. The EMC is an Advanced Microcontroller Bus Architecture (AMBA) compliant peripheral.

## 10.4 Features

- Static chip selects each support up to 64 MB of data. By enabling the address shift mode, static chip select 0 can support up to 256 MB, and static chip select 1 can support up to 128 MB (see SCS register bit 0 ([Section 3.8.1](#)))
- Dynamic chip selects each support up to 256 MB of data.
- Dynamic memory interface support including Single Data Rate SDRAM.
- Asynchronous static memory device support including RAM, ROM, and Flash, with or without asynchronous page mode.
- Low transaction latency.
- Read and write buffers to reduce latency and to improve performance.
- 8-bit, 16-bit, and 32-bit wide static memory support.
- 16-bit and 32-bit wide chip select SDRAM memory support.
- Static memory features include:
  - Asynchronous page mode read
  - Programmable wait states
  - Bus turnaround delay
  - Output enable and write enable delays
  - Extended wait
- Four chip selects for synchronous memory and four chip selects for static memory devices.
- Power-saving modes dynamically control CKE and CLKOUT to SDRAMs.
- Dynamic memory self-refresh mode controlled by software.
- Controller supports 2 kbit, 4 kbit, and 8 kbit row address synchronous memory parts. That is typical 512 Mb, 256 Mb, and 128 Mb parts, with 4, 8, 16, or 32 data bits per device.
- Separate reset domains allow the for auto-refresh through a chip reset if desired.
- Programmable delay elements allow fine-tuning EMC timing.

Note: Synchronous static memory devices (synchronous burst mode) are not supported.

### 10.5 EMC functional description

Figure 13 shows a block diagram of the EMC.

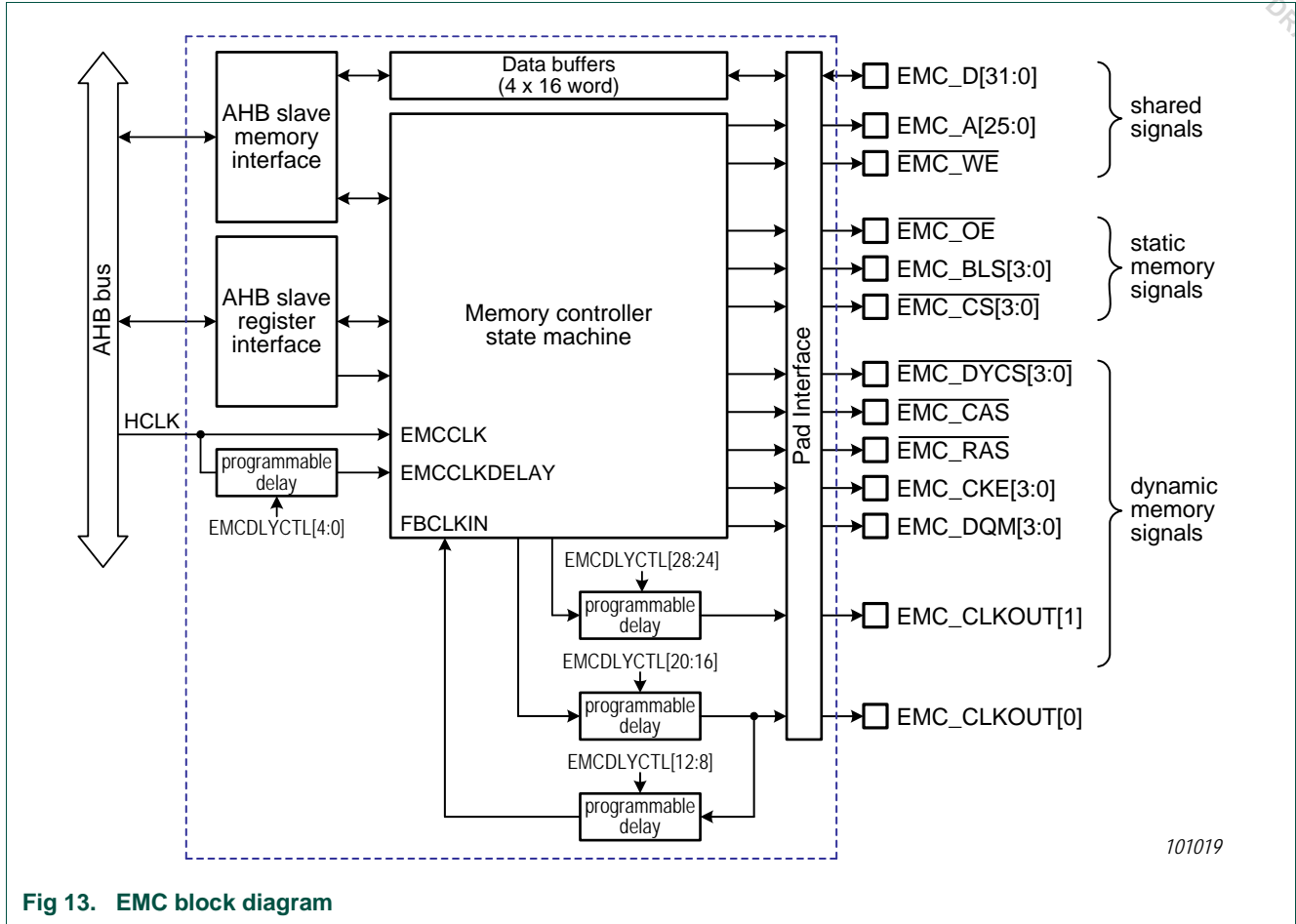


Fig 13. EMC block diagram

The functions of the EMC blocks are described in the following sections:

- AHB slave register interface.
- AHB slave memory interfaces.
- Data buffers.
- Memory controller state machine.
- Pad interface.

Note: For 32 bit wide chip selects data is transferred to and from dynamic memory in SDRAM bursts of four. For 16 bit wide chip selects SDRAM bursts of eight are used.

#### 10.5.1 AHB slave register interface

The AHB slave register interface block enables the registers of the EMC to be programmed. This module also contains most of the registers and performs the majority of the register address decoding.

To eliminate the possibility of endianness problems, all data transfers to and from the registers of the EMC must be 32 bits wide.

**Note:** If an access is attempted with a size other than a word (32 bits), it causes an ERROR response to the AHB bus and the transfer is terminated.

## 10.5.2 AHB slave memory interface

The AHB slave memory interface allows access to external memories.

### 10.5.2.1 Memory transaction endianness

The endianness of the data transfers to and from the external memories is determined by the Endian mode (N) bit in the EMCConfig register.

**Note:** The memory controller must be idle (see the busy field of the EMCStatus Register) before endianness is changed, so that the data is transferred correctly.

### 10.5.2.2 Memory transaction size

Memory transactions can be 8, 16, or 32 bits wide. Any access attempted with a size greater than a word (32 bits) causes an ERROR response to the AHB bus and the transfer is terminated.

### 10.5.2.3 Write protected memory areas

Write transactions to write-protected memory areas generate an ERROR response to the AHB bus and the transfer is terminated.

## 10.5.3 Pad interface

The pad interface block provides the interface to the pads. The pad interface uses a feedback clock, FBCLKIN, from the CLKOUT[0] output of the EMC to resynchronize SDRAM read data from the off-chip to on-chip domains.

## 10.5.4 Data buffers

The AHB interface reads and writes via buffers to improve memory bandwidth and reduce transaction latency. The EMC contains four 16-word buffers. The buffers can be used as read buffers, write buffers, or a combination of both. The buffers are allocated automatically.

The buffers must be disabled during SDRAM and SyncFlash initialization. They must also be disabled when performing SyncFlash commands. The buffers must be enabled during normal operation.

The buffers can be enabled or disabled for static memory using the EMCStaticConfig Registers.

### 10.5.4.1 Write buffers

Write buffers are used to:

- Merge write transactions so that the number of external transactions are minimized. Buffer data until the EMC can complete the write transaction, improving AHB write latency.

Convert all dynamic memory write transactions into quadword bursts on the external memory interface. This enhances transfer efficiency for dynamic memory.

- Reduce external memory traffic. This improves memory bandwidth and reduces power consumption.

Write buffer operation:

- If the buffers are enabled, an AHB write operation writes into the Least Recently Used (LRU) buffer, if empty.

If the LRU buffer is not empty, the contents of the buffer are flushed to memory to make space for the AHB write data.

- If a buffer contains write data it is marked as dirty, and its contents are written to memory before the buffer can be reallocated.

The write buffers are flushed whenever:

- The memory controller state machine is not busy performing accesses to external memory.

The memory controller state machine is not busy performing accesses to external memory, and an AHB interface is writing to a different buffer.

Note: For dynamic memory, the smallest buffer flush is a quadword of data. For static memory, the smallest buffer flush is a byte of data.

#### 10.5.4.2 Read buffers

Read buffers are used to:

- Buffer read requests from memory. Future read requests that hit the buffer read the data from the buffer rather than memory, reducing transaction latency.

Convert all read transactions into quadword bursts on the external memory interface. This enhances transfer efficiency for dynamic memory.

- Reduce external memory traffic. This improves memory bandwidth and reduces power consumption.

Read buffer operation:

- If the buffers are enabled and the read data is contained in one of the buffers, the read data is provided directly from the buffer.
- If the read data is not contained in a buffer, the LRU buffer is selected. If the buffer is dirty (contains write data), the write data is flushed to memory. When an empty buffer is available the read command is posted to the memory.

A buffer filled by performing a read from memory is marked as not-dirty (not containing write data) and its contents are not flushed back to the memory controller unless a subsequent AHB transfer performs a write that hits the buffer.

#### 10.5.5 Memory controller state machine

The memory controller state machine comprises a static memory controller and a dynamic memory controller.

### 10.5.6 Timing control with programmable delay elements

Programmable delay elements are provided to allow fine-tuning the timing of various aspects of EMC operation in connection with SDRAM memory.

- For the clock delayed operating mode, separate programmable delays are provided for each potential clock output, CLKOUT0 and CLKOUT1.
- For the command delayed operating mode, a programmable delay is provided to control delay of all command outputs.
- For both operating modes, a programmable delay is provided to control the time at which input data from SDRAM memory is sampled.

The locations of the programmable delays are shown in the EMC overall block diagram ([Figure 13](#)). See descriptions of the EMCDLYCTL and EMCCAL registers for more information.

## 10.6 Low-power operation

In many systems, the contents of the memory system have to be maintained during low-power sleep modes. The EMC provides a mechanism to place the dynamic memories into self-refresh mode.

Self-refresh mode can be entered by software by setting the SREFREQ bit in the EMCDynamicControl Register and polling the SREFACK bit in the EMCStatus Register.

Any transactions to memory that are generated while the memory controller is in self-refresh mode are rejected and an error response is generated to the AHB bus. Clearing the SREFREQ bit in the EMCDynamicControl Register returns the memory to normal operation. See the memory data sheet for refresh requirements.

Note: The static memory can be accessed as normal when the SDRAM memory is in self-refresh mode.

### 10.6.1 Low-power SDRAM Deep-sleep Mode

The EMC supports JEDEC low-power SDRAM deep-sleep mode. Deep-sleep mode can be entered by setting the deep-sleep mode (DP) bit, the dynamic memory clock enable bit (CE), and the dynamic clock control bit (CS) in the EMCDynamicControl register. The device is then put into a low-power mode where the device is powered down and no longer refreshed. All data in the memory is lost.

### 10.6.2 Low-power SDRAM partial array refresh

The EMC supports JEDEC low-power SDRAM partial array refresh. Partial array refresh can be programmed by initializing the SDRAM memory device appropriately. When the memory device is put into self-refresh mode only the memory banks specified are refreshed. The memory banks that are not refreshed lose their data contents.

## 10.7 Memory bank select

Eight independently-configurable memory chip selects are supported:

- Pins  $\overline{\text{EMC\_CS3}}$  to  $\overline{\text{EMC\_CS0}}$  are used to select static memory devices.
- Pins  $\overline{\text{EMC\_DYCS3}}$  to  $\overline{\text{EMC\_DYCS0}}$  are used to select dynamic memory devices.

Static memory chip select ranges are each 64 megabytes in size, while dynamic memory chip selects cover a range of 256 megabytes each. [Table 109](#) shows the address ranges of the chip selects.

**Table 109. Memory bank selection**

Chip select pin	Address range	Memory type	Size of range
$\overline{\text{EMC\_CS0}}$	0x8000 0000 - 0x83FF FFFF	Static	64 MB
$\overline{\text{EMC\_CS1}}$	0x9000 0000 - 0x93FF FFFF	Static	64 MB
$\overline{\text{EMC\_CS2}}$	0x9800 0000 - 0x9BFF FFFF	Static	64 MB
$\overline{\text{EMC\_CS3}}$	0x9C00 0000 - 0x9FFF FFFF	Static	64 MB
$\overline{\text{EMC\_DYCS0}}$	0xA000 0000 - 0xAFFF FFFF	Dynamic	256 MB
$\overline{\text{EMC\_DYCS1}}$	0xB000 0000 - 0xBFFF FFFF	Dynamic	256 MB
$\overline{\text{EMC\_DYCS2}}$	0xC000 0000 - 0xCFFF FFFF	Dynamic	256 MB
$\overline{\text{EMC\_DYCS3}}$	0xD000 0000 - 0xDFFF FFFF	Dynamic	256 MB

## 10.8 EMC Reset

The EMC receives two reset signals. One is Power-On Reset (POR), asserted when chip power is applied, and when a brown-out condition is detected (see the System Control Block chapter for details of Brown-Out Detect). The other reset is from the external Reset pin and the Watchdog Timer.

A configuration bit in the SCS register, called EMC\_Reset\_Disable, allows control of how the EMC is reset (see [Section 3.8.1 “System Controls and Status register \(SCS - 0x400F C1A0\)”](#)). The default configuration (EMC\_Reset\_Disable = 0) is that both EMC resets are asserted when any type of reset event occurs. In this mode, all registers and functions of the EMC are initialized upon any reset condition.

If EMC\_Reset\_Disable is set to 1, many portions of the EMC are only reset by a power-on or brown-out event, in order to allow the EMC to retain its state through a warm reset (external reset or watchdog reset). If the EMC is configured correctly, auto-refresh can be maintained through a warm reset.

## 10.9 Address shift mode

The EMC supports an optional address shift mode for static memories that can simplify board design and potentially increase external memory addressing range in some cases. The latter cases are described in footnotes of [Table 3 “LPC178x/177x memory usage and details”](#) in the Memory Map chapter of this manual.

Address shift mode is controlled by a configuration bit in the SCS register, called EMC Shift Control (see [Section 3.8.1 “System Controls and Status register \(SCS - 0x400F C1A0\)”](#)).

When the address shift mode is not activated (the EMC Shift Control bit in the SCS register = 1), static memory addresses are output as byte addresses. This means that for memories wider than a byte, one or two address lines are not used, and that address connections to memory devices must be shifted in the board design. For example, if a 32-bit wide memory system is connected, the lowest line address of the memory device(s) would be connected to EMC address line 2, skipping bits 0 and 1.

When the address shift mode is activated (the EMC Shift Control bit in the SCS register = 0), static memory addresses are shifted to match the lowest address bit needed for bus width. In this case, the lowest address line of the memory device(s) is always to EMC address line 0.

## 10.10 Memory mapped I/O and burst disable

By default, the EMC uses buffering to obtain better external memory access performance. However, in the case of memory mapped I/O devices, the read-ahead operations that occur due to the buffering can cause issues with some such devices. This could be from a change of status in one register caused by reading another register, or could simply cause an unplanned read of a data FIFO when another register in the device is read intentionally.

To prevent this issue, the use of buffering to read ahead of actual CPU memory read requests can be disabled. The configuration bit that controls this function is called EMC Burst Control, and is found in the SCS register (see [Section 3.8.1 “System Controls and Status register \(SCS - 0x400F C1A0\)”](#)).



## 10.11 Pin description

[Table 110](#) shows the interface and control signal pins for the EMC.

**Table 110. Pad interface and control signal descriptions**

Name	Type	Value on POR reset	Value during self-refresh	Description
EMC_A[23:0]	Output	0	Depends on static memory accesses	External memory address output. Used for both static and SDRAM devices. SDRAM memories use only bits [14:0].
EMC_D[31:0]	Input/Output	Data outputs = 0	Depends on static memory accesses	External memory data lines. These are inputs when data is read from external memory and outputs when data is written to external memory.
$\overline{\text{EMC\_OE}}$	Output	1	Depends on static memory accesses	Low active output enable for static memory devices.
EMC_BLS3:0	Output	0xF	Depends on static memory accesses	Low active byte lane selects. Used for static memory devices.
$\overline{\text{EMC\_WE}}$	Output	1	Depends on static memory accesses	Low active write enable. Used for SDRAM and static memories.
$\overline{\text{EMC\_CS}}3:0$	Output	0xF	Depends on static memory accesses	Static memory chip selects. Default active LOW. Used for static memory devices.
$\overline{\text{EMC\_DYCS}}3:0$	Output	0xF	0xF	SDRAM chip selects. Used for SDRAM devices.
$\overline{\text{EMC\_CAS}}$	Output	1	1	Column address strobe. Used for SDRAM devices.
$\overline{\text{EMC\_RAS}}$	Output	1	1	Row address strobe. Used for SDRAM devices.
EMC_CLK1:0	Output	Follows CCLK	Follows CCLK	SDRAM clocks. Used for SDRAM devices.
EMC_CKE3:0	Output	0xF	0x0	SDRAM clock enables. Used for SDRAM devices. One is allocated for each Chip Select.
EMC_DQM3:0	Output	0xF	0xF	Data mask output to SDRAMs. Used for SDRAM devices and static memories.

## 10.12 Register description

This chapter describes the EMC registers and provides details required when programming the microcontroller. The EMC registers are shown in [Table 111](#).

**Table 111. Summary of EMC registers**

Address	Register Name	Description	Warm Reset Value <a href="#">[1]</a>	POR Reset Value <a href="#">[1]</a>	Type	Table
0x2009 C000	EMCControl	Controls operation of the memory controller.	0x1	0x3	R/W	<a href="#">112</a>
0x2009 C004	EMCStatus	Provides EMC status information.	-	0x5	RO	<a href="#">113</a>
0x2009 C008	EMCConfig	Configures operation of the memory controller	-	0x0	R/W	<a href="#">114</a>
0x2009 C020	EMCDynamicControl	Controls dynamic memory operation.	-	0x006	R/W	<a href="#">115</a>
0x2009 C024	EMCDynamicRefresh	Configures dynamic memory refresh operation.	-	0x0	R/W	<a href="#">116</a>
0x2009 C028	EMCDynamicReadConfig	Configures the dynamic memory read strategy.	-	0x0	R/W	<a href="#">117</a>
0x2009 C030	EMCDynamictRP	Selects the precharge command period.	-	0x0F	R/W	<a href="#">118</a>
0x2009 C034	EMCDynamictRAS	Selects the active to precharge command period.	-	0xF	R/W	<a href="#">119</a>
0x2009 C038	EMCDynamictSREX	Selects the self-refresh exit time.	-	0xF	R/W	<a href="#">120</a>
0x2009 C03C	EMCDynamictAPR	Selects the last-data-out to active command time.	-	0xF	R/W	<a href="#">121</a>
0x2009 C040	EMCDynamictDAL	Selects the data-in to active command time.	-	0xF	R/W	<a href="#">122</a>
0x2009 C044	EMCDynamictWR	Selects the write recovery time.	-	0xF	R/W	<a href="#">123</a>
0x2009 C048	EMCDynamictRC	Selects the active to active command period.	-	0x1F	R/W	<a href="#">124</a>
0x2009 C04C	EMCDynamictRFC	Selects the auto-refresh period.	-	0x1F	R/W	<a href="#">125</a>
0x2009 C050	EMCDynamictXSR	Selects the exit self-refresh to active command time.	-	0x1F	R/W	<a href="#">126</a>
0x2009 C054	EMCDynamictRRD	Selects the active bank A to active bank B latency.	-	0xF	R/W	<a href="#">127</a>
0x2009 C058	EMCDynamictMRD	Selects the load mode register to active command time.	-	0xF	R/W	<a href="#">128</a>
0x2009 C080	EMCStaticExtendedWait	Selects time for long static memory read and write transfers.	-	0x0	R/W	<a href="#">129</a>
0x2009 C100	EMCDynamicConfig0	Selects the configuration information for EMC_DYCS[0].	-	0x0	R/W	<a href="#">130</a>
0x2009 C104	EMCDynamicRasCas0	Selects the RAS and CAS latencies for EMC_DYCS[0].	-	0x303	R/W	<a href="#">132</a>
0x2009 C120	EMCDynamicConfig1	Selects the configuration information for EMC_DYCS[1].	-	0x0	R/W	<a href="#">130</a>
0x2009 C124	EMCDynamicRasCas1	Selects the RAS and CAS latencies for EMC_DYCS[1].	-	0x303	R/W	<a href="#">132</a>
0x2009 C140	EMCDynamicConfig2	Selects the configuration information for EMC_DYCS[2].	-	0x0	R/W	<a href="#">130</a>
0x2009 C144	EMCDynamicRasCas2	Selects the RAS and CAS latencies for EMC_DYCS[2].	-	0x303	R/W	<a href="#">132</a>
0x2009 C160	EMCDynamicConfig3	Selects the configuration information for EMC_DYCS[3].	-	0x0	R/W	<a href="#">130</a>
0x2009 C164	EMCDynamicRasCas3	Selects the RAS and CAS latencies for EMC_DYCS[3].	-	0x303	R/W	<a href="#">132</a>

Table 111. Summary of EMC registers ...continued

Address	Register Name	Description	Warm Reset Value [1]	POR Reset Value [1]	Type	Table
0x2009 C200	EMCStaticConfig0	Selects the memory configuration for $\overline{\text{EMC\_CS}}[0]$ .	-	0x0	R/W	<a href="#">133</a>
0x2009 C204	EMCStaticWaitWen0	Selects the delay from $\overline{\text{EMC\_CS}}[0]$ to write enable.	-	0x0	R/W	<a href="#">134</a>
0x2009 C208	EMCStaticWaitOen0	Selects the delay from $\overline{\text{EMC\_CS}}[0]$ or address change, whichever is later, to output enable.	-	0x0	R/W	<a href="#">135</a>
0x2009 C20C	EMCStaticWaitRd0	Selects the delay from $\overline{\text{EMC\_CS}}[0]$ to a read access.	-	0x1F	R/W	<a href="#">136</a>
0x2009 C210	EMCStaticWaitPage0	Selects the delay for asynchronous page mode sequential accesses for $\overline{\text{EMC\_CS}}[0]$ .	-	0x1F	R/W	<a href="#">137</a>
0x2009 C214	EMCStaticWaitWr0	Selects the delay from $\overline{\text{EMC\_CS}}[0]$ to a write access.	-	0x1F	R/W	<a href="#">138</a>
0x2009 C218	EMCStaticWaitTurn0	Selects the number of bus turnaround cycles $\overline{\text{EMC\_CS}}[0]$ .	-	0xF	R/W	<a href="#">139</a>
0x2009 C220	EMCStaticConfig1	Selects the memory configuration for $\overline{\text{EMC\_CS}}[1]$ .	-	0x0	R/W	<a href="#">133</a>
0x2009 C224	EMCStaticWaitWen1	Selects the delay from $\overline{\text{EMC\_CS}}[1]$ to write enable.	-	0x0	R/W	<a href="#">134</a>
0x2009 C228	EMCStaticWaitOen1	Selects the delay from $\overline{\text{EMC\_CS}}[1]$ or address change, whichever is later, to output enable.	-	0x0	R/W	<a href="#">135</a>
0x2009 C22C	EMCStaticWaitRd1	Selects the delay from $\overline{\text{EMC\_CS}}[1]$ to a read access.	-	0x1F	R/W	<a href="#">136</a>
0x2009 C230	EMCStaticWaitPage1	Selects the delay for asynchronous page mode sequential accesses for $\overline{\text{EMC\_CS}}[1]$ .	-	0x1F	R/W	<a href="#">137</a>
0x2009 C234	EMCStaticWaitWr1	Selects the delay from $\overline{\text{EMC\_CS}}[1]$ to a write access.	-	0x1F	R/W	<a href="#">138</a>
0x2009 C238	EMCStaticWaitTurn1	Selects the number of bus turnaround cycles for $\overline{\text{EMC\_CS}}[1]$ .	-	0xF	R/W	<a href="#">139</a>
0x2009 C240	EMCStaticConfig2	Selects the memory configuration for $\overline{\text{EMC\_CS}}[2]$ .	-	0x0	R/W	<a href="#">133</a>
0x2009 C244	EMCStaticWaitWen2	Selects the delay from $\overline{\text{EMC\_CS}}[2]$ to write enable.	-	0x0	R/W	<a href="#">134</a>
0x2009 C248	EMCStaticWaitOen2	Selects the delay from $\overline{\text{EMC\_CS}}[2]$ or address change, whichever is later, to output enable.	-	0x0	R/W	<a href="#">135</a>
0x2009 C24C	EMCStaticWaitRd2	Selects the delay from $\overline{\text{EMC\_CS}}[2]$ to a read access.	-	0x1F	R/W	<a href="#">136</a>
0x2009 C250	EMCStaticWaitPage2	Selects the delay for asynchronous page mode sequential accesses for $\overline{\text{EMC\_CS}}[2]$ .	-	0x1F	R/W	<a href="#">137</a>
0x2009 C254	EMCStaticWaitWr2	Selects the delay from $\overline{\text{EMC\_CS}}[2]$ to a write access.	-	0x1F	R/W	<a href="#">138</a>
0x2009 C258	EMCStaticWaitTurn2	Selects the number of bus turnaround cycles for $\overline{\text{EMC\_CS}}[2]$ .	-	0xF	R/W	<a href="#">139</a>
0x2009 C260	EMCStaticConfig3	Selects the memory configuration for $\overline{\text{EMC\_CS}}[3]$ .	-	0x0	R/W	<a href="#">133</a>
0x2009 C264	EMCStaticWaitWen3	Selects the delay from $\overline{\text{EMC\_CS}}[3]$ to write enable.	-	0x0	R/W	<a href="#">134</a>
0x2009 C268	EMCStaticWaitOen3	Selects the delay from $\overline{\text{EMC\_CS}}[3]$ or address change, whichever is later, to output enable.	-	0x0	R/W	<a href="#">135</a>
0x2009 C26C	EMCStaticWaitRd3	Selects the delay from $\overline{\text{EMC\_CS}}[3]$ to a read access.	-	0x1F	R/W	<a href="#">136</a>
0x2009 C270	EMCStaticWaitPage3	Selects the delay for asynchronous page mode sequential accesses for $\overline{\text{EMC\_CS}}[3]$ .	-	0x1F	R/W	<a href="#">137</a>
0x2009 C274	EMCStaticWaitWr3	Selects the delay from $\overline{\text{EMC\_CS}}[3]$ to a write access.	-	0x1F	R/W	<a href="#">138</a>

Table 111. Summary of EMC registers ...continued

Address	Register Name	Description	Warm Reset Value <a href="#">[1]</a>	POR Reset Value <a href="#">[1]</a>	Type	Table
0x2009 C278	EMCStaticWaitTurn3	Selects the number of bus turnaround cycles for EMC_CS[3].	-	0xF	R/W	<a href="#">139</a>
0x400F C1DC	EMCDLYCTL	Selects values for the 4 programmable delays associated with SDRAM operation.	-	0x210	R/W	<a href="#">140</a>
0x400F C1E0	EMCCAL	Controls the calibration counter for programmable delays and returns the result value.	-	0x1F00	R/W	<a href="#">141</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 10.12.1 EMC Control register (EMCControl - 0x2009 C000)

The EMCControl register is a read/write register that controls operation of the memory controller. The control bits can be altered during normal operation. [Table 112](#) shows the bit assignments for the EMCControl register.

**Table 112. EMC Control register (EMCControl - address 0x2009 C000) bit description**

Bit	Symbol	Value	Description	Reset Value
0	EMC Enable (E)		Indicates if the EMC is enabled or disabled:	1
		0	Disabled	
		1	Enabled (POR and warm reset value).  Disabling the EMC reduces power consumption. When the memory controller is disabled the memory is not refreshed. The memory controller is enabled by setting the enable bit, or by reset.  This bit must only be modified when the EMC is in idle state. <sup>[1]</sup>	
1	Address mirror (M)		Indicates normal or reset memory map:	1
		0	Normal memory map.	
		1	Reset memory map. Static memory $\overline{\text{EMC\_CS1}}$ is mirrored onto $\overline{\text{EMC\_CS0}}$ and $\overline{\text{EMC\_DYCS0}}$ (POR reset value).  On POR, $\overline{\text{EMC\_CS1}}$ is mirrored to both $\overline{\text{EMC\_CS0}}$ and $\overline{\text{EMC\_DYCS0}}$ memory areas. Clearing the M bit enables $\overline{\text{EMC\_CS0}}$ and $\overline{\text{EMC\_DYCS0}}$ memory to be accessed.	
2	Low-power mode (L)		Indicates normal, or low-power mode:	0
		0	Normal mode (warm reset value).	
		1	Low-power mode.  Entering low-power mode reduces memory controller power consumption. Dynamic memory is refreshed as necessary. The memory controller returns to normal functional mode by clearing the low-power mode bit (L), or by POR.  This bit must only be modified when the EMC is in idle state. <sup>[1]</sup>	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] The external memory cannot be accessed in low-power or disabled state. If a memory access is performed an AHB error response is generated. The EMC registers can be programmed in low-power and/or disabled state.

### 10.12.2 EMC Status register (EMCStatus - 0x2009 C004)

The read-only EMCStatus register provides EMC status information. [Table 113](#) shows the bit assignments for the EMCStatus register.

**Table 113. EMC Status register (EMCStatus - address 0x2009 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	Busy (B)		This bit is used to ensure that the memory controller enters the low-power or disabled mode cleanly by determining if the memory controller is busy or not:	1
		0	EMC is idle (warm reset value).	
		1	EMC is busy performing memory transactions, commands, auto-refresh cycles, or is in self-refresh mode (POR reset value).	

**Table 113. EMC Status register (EMCStatus - address 0x2009 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
1	Write buffer status (S)		This bit enables the EMC to enter low-power mode or disabled mode cleanly:	0
		0	Write buffers empty (POR reset value)	
		1	Write buffers contain data.	
2	Self-refresh acknowledge (SA)		This bit indicates the operating mode of the EMC:	1
		0	Normal mode	
		1	Self-refresh mode (POR reset value).	
31:3	-		Reserved. The value read from a reserved bit is not defined.	NA

### 10.12.3 EMC Configuration register (EMCConfig - 0x2009 C008)

The EMCConfig register configures the operation of the memory controller. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This register is accessed with one wait state. [Table 114](#) shows the bit assignments for the EMCConfig register.

**Table 114. EMC Configuration register (EMCConfig - address 0x2009 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
0			Endian mode:	0
		0	Little-endian mode (POR reset value).	
		1	Big-endian mode.	
			On power-on reset, the value of the endian bit is 0. All data must be flushed in the EMC before switching between little-endian and big-endian modes.	
7:1	-		Reserved. Read value is undefined, only zero should be written.	NA
8			CCLK: CLKOUT[1:0] ratio:	0
		0	1:1 (POR reset value)	
		1	1:2 (this option is not available on the LPC178x/177x)	
			This bit must contain 0 for proper operation of the EMC.	
31:9	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.4 Dynamic Memory Control register (EMCDynamicControl - 0x2009 C020)

The EMCDynamicControl register controls dynamic memory operation. The control bits can be altered during normal operation. [Table 115](#) shows the bit assignments for the EMCDynamicControl register.

**Table 115. Dynamic Control register (EMCDynamicControl - address 0x2009 C020) bit description**

Bit	Symbol	Value	Description	Reset Value
0	Dynamic memory clock enable (CE)	0	Clock enable of idle devices are deasserted to save power (POR reset value).	0
		1	All clock enables are driven HIGH continuously. <a href="#">[1]</a>	

Table 115. Dynamic Control register (EMCDynamicControl - address 0x2009 C020) bit description

Bit	Symbol	Value	Description	Reset Value
1	Dynamic memory clock control (CS)	0	CLKOUT stops when all SDRAMs are idle and during self-refresh mode.	1
		1	CLKOUT runs continuously (POR reset value).  When clock control is LOW the output clock CLKOUT is stopped when there are no SDRAM transactions. The clock is also stopped during self-refresh mode.	
2	Self-refresh request, EMCSREFREQ (SR)	0	Normal mode.	1
		1	Enter self-refresh mode (POR reset value).  By writing 1 to this bit self-refresh can be entered under software control. Writing 0 to this bit returns the EMC to normal mode.  The self-refresh acknowledge bit in the EMCStatus register must be polled to discover the current operating mode of the EMC. <a href="#">[2]</a>	
4:3	-		Reserved. Read value is undefined, only zero should be written.	NA
5	Memory clock control (MMC)	0	CLKOUT enabled (POR reset value).	0
		1	CLKOUT disabled. <a href="#">[3]</a>	
6	-		Reserved. Read value is undefined, only zero should be written.	NA
8:7	SDRAM initialization (I)	00	Issue SDRAM NORMAL operation command (POR reset value).	00
		01	Issue SDRAM MODE command.	
		10	Issue SDRAM PALL (precharge all) command.	
		11	Issue SDRAM NOP (no operation) command	
12:9	-		Reserved. Read value is undefined, only zero should be written.	NA
13	Low-power SDRAM deep-sleep mode (DP)	0	Normal operation (POR reset value).	0
		1	Enter Deep-sleep mode.	
31:14	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] Clock enable must be HIGH during SDRAM initialization.

[2] The memory controller exits from power-on reset with the self-refresh bit HIGH. To enter normal functional mode set this bit LOW.

[3] Disabling CLKOUT can be performed if there are no SDRAM memory transactions. When enabled this bit can be used in conjunction with the dynamic memory clock control (CS) field.

**Remark:** Deep-sleep mode can be entered by setting the deep-sleep mode (DP) bit, the dynamic memory clock enable bit (CE), and the dynamic clock control bit (CS) to one. The device is then put into a low-power mode where the device is powered down and no longer refreshed. All data in the memory is lost.

### 10.12.5 Dynamic Memory Refresh Timer register (EMCDynamicRefresh - 0x2009 C024)

The EMCDynamicRefresh register configures dynamic memory operation. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. However, these control bits can, if necessary, be altered during normal operation. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed. [Table 116](#) shows the bit assignments for the EMCDynamicRefresh register.

**Table 116. Dynamic Memory Refresh Timer register (EMCDynamicRefresh - address 0x2009 C024) bit description**

Bit	Symbol	Value	Description	Reset Value
10:0	Refresh timer (REFRESH)		Indicates the multiple of 16 CCLKs between SDRAM refresh cycles.	0
		0x0	Refresh disabled (POR reset value).	
		0x1	0x7FF = n x 16 = 16n CCLKs between SDRAM refresh cycles.	
			For example: 0x1 = 1 x 16 = 16 CCLKs between SDRAM refresh cycles. 0x8 = 8 x 16 = 128 CCLKs between SDRAM refresh cycles.	
31:11	-		Reserved. Read value is undefined, only zero should be written.	NA

For example, for the refresh period of 16  $\mu$ s, and a CCLK frequency of 50 MHz, the following value must be programmed into this register:

$$(16 \times 10^{-6} \times 50 \times 10^6) / 16 = 50 \text{ or } 0x32$$

If auto-refresh through warm reset is requested (by setting the EMC\_Reset\_Disable bit), the timing of auto-refresh must be adjusted to allow a sufficient refresh rate when the clock rate is reduced during the wake-up period of a reset cycle. During this period, the EMC (and all other portions of the LPC178x/177x that are being clocked) run from the IRC oscillator at 12 MHz. So, 12 MHz must be considered the CCLK rate for refresh calculations if auto-refresh through warm reset is requested.

Note: The refresh cycles are evenly distributed. However, there might be slight variations when the auto-refresh command is issued depending on the status of the memory controller.

### 10.12.6 Dynamic Memory Read Configuration register (EMCDynamicReadConfig - 0x2009 C028)

The EMCDynamicReadConfig register configures the dynamic memory read strategy. This register must only be modified during system initialization. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 117](#) shows the bit assignments for the EMCDynamicReadConfig register.



**Table 117. Dynamic Memory Read Configuration register (EMCDynamicReadConfig - address 0x2009 C028) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Read data strategy (RD)	00	Clock out delayed strategy, using CLKOUT (command not delayed, clock out delayed). POR reset value.	0x0
		01	Command delayed strategy, using EMCCLKDELAY (command delayed, clock out not delayed).	
		10	Command delayed strategy plus one clock cycle, using EMCCLKDELAY (command delayed, clock out not delayed).	
		11	Command delayed strategy plus two clock cycles, using EMCCLKDELAY (command delayed, clock out not delayed).	
31:2	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.7 Dynamic Memory Precharge Command Period register (EMCDynamicTRP - 0x2009 C030)

The EMCDynamicTRP register enables you to program the precharge command period, tRP. This register must only be modified during system initialization. This value is normally found in SDRAM data sheets as tRP. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 118](#) shows the bit assignments for the EMCDynamicTRP register.

**Table 118. Dynamic Memory Precharge Command Period register (EMCDynamicTRP - address 0x2009 C030) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Precharge command period (tRP)	0x0 - 0xE	n + 1 clock cycles. The delay is in EMCCLK cycles.	0x0F
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.8 Dynamic Memory Active to Precharge Command Period register (EMCDynamicTRAS - 0x2009 C034)

The EMCDynamicTRAS register enables you to program the active to precharge command period, tRAS. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tRAS. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 119](#) shows the bit assignments for the EMCDynamicTRAS register.

**Table 119. Dynamic Memory Active to Precharge Command Period register (EMCDynamicRAS - address 0x2009 C034) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Active to precharge command period (tRAS)	0x0 -	n + 1 clock cycles. The delay is in EMCCLK cycles.	0xF
		0xE		
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.9 Dynamic Memory Self-refresh Exit Time register (EMCDynamicSREX - 0x2009 C038)

The EMCDynamicTSREX register enables you to program the self-refresh exit time, tSREX. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tSREX, for devices without this parameter you use the same value as tXSR. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 120](#) shows the bit assignments for the EMCDynamicTSREX register.

**Table 120. Dynamic Memory Self-refresh Exit Time register (EMCDynamicSREX - address 0x2009 C038) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Self-refresh exit time (tSREX)	0x0 -	n + 1 clock cycles. The delay is in CCLK cycles.	0xF
		0xE		
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.10 Dynamic Memory Last Data Out to Active Time register (EMCDynamicAPR - 0x2009 C03C)

The EMCDynamicTAPR register enables you to program the last-data-out to active command time, tAPR. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tAPR. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 121](#) shows the bit assignments for the EMCDynamicTAPR register.

**Table 121. Dynamic Memory Last Data Out to Active Time register (EMCDynamicTAPR - address 0x2009 C03C) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Last-data-out to active command time (tAPR)	0x0 -	n + 1 clock cycles. The delay is in CCLK cycles.	0xF
		0xE		
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.11 Dynamic Memory Data-in to Active Command Time register (EMCDynamicTDAL - 0x2009 C040)

The EMCDynamicTDAL register enables you to program the data-in to active command time, tDAL. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tDAL, or tAPW. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 122](#) shows the bit assignments for the EMCDynamicTDAL register.

**Table 122. Dynamic Memory Data-in to Active Command Time register (EMCDynamicTDAL - address 0x2009 C040) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Data-in to active command (tDAL)	0x0 -	n clock cycles. The delay is in CCLK cycles.	0xF
		0xE		
		0xF	15 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.12 Dynamic Memory Write Recovery Time register (EMCDynamicTWR - 0x2009 C044)

The EMCDynamicTWR register enables you to program the write recovery time, tWR. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tWR, tDPL, tRWL, or tRDL. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 123](#) shows the bit assignments for the EMCDynamicTWR register.

**Table 123. Dynamic Memory Write recover Time register (EMCDynamicWR - address 0x2009 C044) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Write recovery time (tWR)	0x0 - 0xE	n + 1 clock cycles. The delay is in CCLK cycles.	0xF
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.13 Dynamic Memory Active to Active Command Period register (EMCDynamicTRC - 0x2009 C048)

The EMCDynamicTRC register enables you to program the active to active command period, tRC. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tRC. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 124](#) shows the bit assignments for the EMCDynamicTRC register.

**Table 124. Dynamic Memory Active to Active Command Period register (EMCDynamicTRC - address 0x2009 C048) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Active to active command period (tRC)	0x0 - 0x1E	n + 1 clock cycles. The delay is in CCLK cycles.	0x1F
		0xF	32 clock cycles (POR reset value).	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.14 Dynamic Memory Auto-refresh Period register (EMCDynamicTRFC - 0x2009 C04C)

The EMCDynamicTRFC register enables you to program the auto-refresh period, and auto-refresh to active command period, tRFC. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tRFC, or sometimes as tRC. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 125](#) shows the bit assignments for the EMCDynamicTRFC register.

**Table 125. Dynamic Memory Auto-refresh Period register (EMCDynamicRFC - address 0x2009 C04C) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Auto-refresh period and auto-refresh to active command period (tRFC)	0x0 - 0x1E	n + 1 clock cycles. The delay is in CCLK cycles.	0x1F
		0xF	32 clock cycles (POR reset value).	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.15 Dynamic Memory Exit Self-refresh register (EMCDynamicXSR - 0x2009 C050)

The EMCDynamicXSR register enables you to program the exit self-refresh to active command time, tXSR. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tXSR. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 126](#) shows the bit assignments for the EMCDynamicXSR register.

**Table 126. Dynamic Memory Exit Self-refresh register (EMCDynamicXSR - address 0x2009 C050) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Exit self-refresh to active command time (tXSR)	0x0 - 0x1E	n + 1 clock cycles. The delay is in CCLK cycles.	0x1F
		0xF	32 clock cycles (POR reset value).	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.16 Dynamic Memory Active Bank A to Active Bank B Time register (EMCDynamicTRRD - 0x2009 C054)

The EMCDynamicTRRD register enables you to program the active bank A to active bank B latency, tRRD. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tRRD. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 127](#) shows the bit assignments for the EMCDynamicTRRD register.

**Table 127. Dynamic Memory Active Bank A to Active Bank B Time register (EMCDynamictRRD - address 0x2009 C054) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Active bank A to active bank B latency (tRRD)	0x0 - 0xE	n + 1 clock cycles. The delay is in CCLK cycles.	0xF
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.17 Dynamic Memory Load Mode register to Active Command Time (EMCDynamictMRD - 0x2009 C058)

The EMCDynamictMRD register enables you to program the load mode register to active command time, tMRD. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This value is normally found in SDRAM data sheets as tMRD, or tRSA. This register is accessed with one wait state.

Note: This register is used for all four dynamic memory chip selects. Therefore the worst case value for all of the chip selects must be programmed.

[Table 128](#) shows the bit assignments for the EMCDynamictMRD register.

**Table 128. Dynamic Memory Load Mode register to Active Command Time (EMCDynamictMRD - address 0x2009 C058) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Load mode register to active command time (tMRD)	0x0 - 0xE	n + 1 clock cycles. The delay is in CCLK cycles.	0xF
		0xF	16 clock cycles (POR reset value).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.18 Static Memory Extended Wait register (EMCStaticExtendedWait - 0x2009 C080)

ExtendedWait (EW) bit in the EMCStaticConfig register is set. It is recommended that this register is modified during system initialization, or when there are no current or outstanding transactions. However, if necessary, these control bits can be altered during normal operation. This register is accessed with one wait state.

[Table 129](#) shows the bit assignments for the EMCStaticExtendedWait registers.

**Table 129. Static Memory Extended Wait register (EMCStaticExtendedWait - address 0x2009 C080) bit description**

Bit	Symbol	Value	Description	Reset Value
9:0	Extended wait time out (EXTENDEDWAIT)	0x0	16 clock cycles (POR reset value). The delay is in CCLK cycles.	
		0x1	(n+1) x16 clock cycles.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

For example, for a static memory read/write transfer time of 16 μs, and a CCLK frequency of 50 MHz, the following value must be programmed into this register:  $(16 \times 10^{-6} \times 50 \times 10^6) / 16 - 1 = 49$

### 10.12.19 Dynamic Memory Configuration registers (EMCDynamicConfig0-3 - 0x2009 C100, 120, 140, 160)

The EMCDynamicConfig0-3 registers enable you to program the configuration information for the relevant dynamic memory chip select. These registers are normally only modified during system initialization. These registers are accessed with one wait state.

[Table 130](#) shows the bit assignments for the EMCDynamicConfig0-3 registers.

**Table 130. Dynamic Memory Configuration registers (EMCDynamicConfig0-3 - address 0x2009 C100, 0x2009 C120, 0x2009 C140, 0x2009 C160) bit description**

Bit	Symbol	Value	Description	Reset Value
2:0	-		Reserved. Read value is undefined, only zero should be written.	NA
4:3	Memory device (MD)	00	SDRAM (POR reset value).	00
		01	Low-power SDRAM.	
		10	Micron SyncFlash.	
		11	Reserved.	
6:5	-		Reserved. Read value is undefined, only zero should be written.	NA
12:7	Address mapping (AM)		See <a href="#">Table 131</a>	0
		0	000000 = reset value. <sup>[1]</sup>	
13	-		Reserved. Read value is undefined, only zero should be written.	NA
14	Address mapping (AM)		See <a href="#">Table 131</a>	0
		0	0 = reset value.	
18:15	-		Reserved. Read value is undefined, only zero should be written.	NA
19	Buffer enable (B)	0	Buffer disabled for accesses to this chip select (POR reset value).	0
		1	Buffer enabled for accesses to this chip select. <sup>[2]</sup>	
20	Write protect (P)	0	Writes not protected (POR reset value).	0
		1	Writes protected.	
31:21	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] The SDRAM column and row width and number of banks are computed automatically from the address mapping.

[2] The buffers must be disabled during SDRAM and SyncFlash initialization. They must also be disabled when performing SyncFlash commands. The buffers must be enabled during normal operation.

Address mappings that are not shown in [Table 131](#) are reserved.

**Table 131. Address mapping**

14	12	11:9	8:7	Description
16 bit external bus address mapping (Row, Bank, Column)				
0	0	000	00	16 Mb (2Mx8), 2 banks, row length = 11, column length = 9
0	0	000	01	16 Mb (1Mx16), 2 banks, row length = 11, column length = 8
0	0	001	00	64 Mb (8Mx8), 4 banks, row length = 12, column length = 9
0	0	001	01	64 Mb (4Mx16), 4 banks, row length = 12, column length = 8

Table 131. Address mapping

14	12	11:9	8:7	Description
0	0	010	00	128 Mb (16Mx8), 4 banks, row length = 12, column length = 10
0	0	010	01	128 Mb (8Mx16), 4 banks, row length = 12, column length = 9
0	0	011	00	256 Mb (32Mx8), 4 banks, row length = 13, column length = 10
0	0	011	01	256 Mb (16Mx16), 4 banks, row length = 13, column length = 9
0	0	100	00	512 Mb (64Mx8), 4 banks, row length = 13, column length = 11
0	0	100	01	512 Mb (32Mx16), 4 banks, row length = 13, column length = 10
16 bit external bus address mapping (Bank, Row, Column)				
0	1	000	00	16 Mb (2Mx8), 2 banks, row length = 11, column length = 9
0	1	000	01	16 Mb (1Mx16), 2 banks, row length = 11, column length = 8
0	1	001	00	64 Mb (8Mx8), 4 banks, row length = 12, column length = 9
0	1	001	01	64 Mb (4Mx16), 4 banks, row length = 12, column length = 8
0	1	010	00	128 Mb (16Mx8), 4 banks, row length = 12, column length = 10
0	1	010	01	128 Mb (8Mx16), 4 banks, row length = 12, column length = 9
0	1	011	00	256 Mb (32Mx8), 4 banks, row length = 13, column length = 10
0	1	011	01	256 Mb (16Mx16), 4 banks, row length = 13, column length = 9
0	1	100	00	512 Mb (64Mx8), 4 banks, row length = 13, column length = 11
0	1	100	01	512 Mb (32Mx16), 4 banks, row length = 13, column length = 10
32 bit external bus address mapping (Row, Bank, Column)				
1	0	000	00	16 Mb (2Mx8), 2 banks, row length = 11, column length = 9
1	0	000	01	16 Mb (1Mx16), 2 banks, row length = 11, column length = 8
1	0	001	00	64 Mb (8Mx8), 4 banks, row length = 12, column length = 9
1	0	001	01	64 Mb (4Mx16), 4 banks, row length = 12, column length = 8
1	0	001	10	64 Mb (2Mx32), 4 banks, row length = 11, column length = 8
1	0	010	00	128 Mb (16Mx8), 4 banks, row length = 12, column length = 10
1	0	010	01	128 Mb (8Mx16), 4 banks, row length = 12, column length = 9
1	0	010	10	128 Mb (4Mx32), 4 banks, row length = 12, column length = 8
1	0	011	00	256 Mb (32Mx8), 4 banks, row length = 13, column length = 10
1	0	011	01	256 Mb (16Mx16), 4 banks, row length = 13, column length = 9
1	0	011	10	256 Mb (8Mx32), 4 banks, row length = 13, column length = 8
1	0	100	00	512 Mb (64Mx8), 4 banks, row length = 13, column length = 11
1	0	100	01	512 Mb (32Mx16), 4 banks, row length = 13, column length = 10
32 bit external bus address mapping (Bank, Row, Column)				
1	1	000	00	16 Mb (2Mx8), 2 banks, row length = 11, column length = 9
1	1	000	01	16 Mb (1Mx16), 2 banks, row length = 11, column length = 8
1	1	001	00	64 Mb (8Mx8), 4 banks, row length = 12, column length = 9
1	1	001	01	64 Mb (4Mx16), 4 banks, row length = 12, column length = 8
1	1	001	10	64 Mb (2Mx32), 4 banks, row length = 11, column length = 8
1	1	010	00	128 Mb (16Mx8), 4 banks, row length = 12, column length = 10
1	1	010	01	128 Mb (8Mx16), 4 banks, row length = 12, column length = 9
1	1	010	10	128 Mb (4Mx32), 4 banks, row length = 12, column length = 8
1	1	011	00	256 Mb (32Mx8), 4 banks, row length = 13, column length = 10



**Table 131. Address mapping**

14	12	11:9	8:7	Description
1	1	011	01	256 Mb (16Mx16), 4 banks, row length = 13, column length = 9
1	1	011	10	256 Mb (8Mx32), 4 banks, row length = 13, column length = 8
1	1	100	00	512 Mb (64Mx8), 4 banks, row length = 13, column length = 11
1	1	100	01	512 Mb (32Mx16), 4 banks, row length = 13, column length = 10

A chip select can be connected to a single memory device, in this case the chip select data bus width is the same as the device width. Alternatively the chip select can be connected to a number of external devices. In this case the chip select data bus width is the sum of the memory device data bus widths.

For example, for a chip select connected to:

- a 32 bit wide memory device, choose a 32 bit wide address mapping.
- a 16 bit wide memory device, choose a 16 bit wide address mapping.
- four x 8 bit wide memory devices, choose a 32 bit wide address mapping.
- two x 8 bit wide memory devices, choose a 16 bit wide address mapping.

The SDRAM bank select pins BA1 and BA0 are connected to address lines A14 and A13, respectively.

### 10.12.20 Dynamic Memory RAS & CAS Delay registers (EMCDynamicRASCAS0-3 - 0x2009 C104, 124, 144, 164)

The EMCDynamicRasCas0-3 registers enable you to program the RAS and CAS latencies for the relevant dynamic memory. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are accessed with one wait state.

Note: The values programmed into these registers must be consistent with the values used to initialize the SDRAM memory device.

[Table 132](#) shows the bit assignments for the EMCDynamicRasCas0-3 registers.

**Table 132. Dynamic Memory RAS & CAS Delay registers (EMCDynamicRasCas0-3 - address 0x2009 C104, 0x2009 C124, 0x2009 C144, 0x2009 C164) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	RAS latency (active to read/write delay) (RAS)	00	Reserved.	11
		01	One CCLK cycle.	
		10	Two CCLK cycles.	
		11	Three CCLK cycles (POR reset value).	
7:2	-		Reserved. Read value is undefined, only zero should be written.	NA

**Table 132. Dynamic Memory RAS & CAS Delay registers (EMCDynamicRasCas0-3 - address 0x2009 C104, 0x2009 C124, 0x2009 C144, 0x2009 C164) bit description**

Bit	Symbol	Value	Description	Reset Value
9:8	CAS latency (CAS)	00	Reserved.	11
		01	One CCLK cycle.	
		10	Two CCLK cycles.	
		11	Three CCLK cycles (POR reset value).	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.21 Static Memory Configuration registers (EMCStaticConfig0-3 - 0x2009 C200, 220, 240, 260)

The EMCStaticConfig0-3 registers configure the static memory configuration. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are accessed with one wait state.

[Table 133](#) shows the bit assignments for the EMCStaticConfig0-3 registers. Note that synchronous burst mode memory devices are not supported.

**Table 133. Static Memory Configuration registers (EMCStaticConfig0-3 - address 0x2009 C200, 0x2009 C220, 0x2009 C240, 0x2009 C260) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Memory width (MW)	00	8 bit (POR reset value).	0
		01	16 bit.	
		10	32 bit.	
		11	Reserved.	
2	-		Reserved. Read value is undefined, only zero should be written.	NA
3	Page mode (PM)		In page mode the EMC can burst up to four external accesses. Therefore devices with asynchronous page mode burst four or higher devices are supported. Asynchronous page mode burst two devices are not supported and must be accessed normally.	0
		0	Disabled (POR reset value).	
		1	Asynchronous page mode enabled (page length four).	
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
6	Chip select polarity (PC)		The value of the chip select polarity on power-on reset is 0.	0
		0	Active LOW chip select.	
		1	Active HIGH chip select.	

**Table 133. Static Memory Configuration registers (EMCStaticConfig0-3 - address 0x2009 C200, 0x2009 C220, 0x2009 C240, 0x2009 C260) bit description**

Bit	Symbol	Value	Description	Reset Value
7	Byte lane state (PB)		The byte lane state bit, PB, enables different types of memory to be connected. For byte-wide static memories the BLSn[3:0] signal from the EMC is usually connected to $\overline{WE}$ (write enable). In this case for reads all the BLSn[3:0] bits must be HIGH. This means that the byte lane state (PB) bit must be LOW.	0
		0	For reads all the bits in BLSn[3:0] are HIGH. For writes the respective active bits in BLSn[3:0] are LOW (POR reset value).	
		1	For reads the respective active bits in BLSn[3:0] are LOW. For writes the respective active bits in BLSn[3:0] are LOW.	
8	Extended wait (EW)		Extended wait (EW) uses the EMCStaticExtendedWait register to time both the read and write transfers rather than the EMCStaticWaitRd and EMCStaticWaitWr registers. This enables much longer transactions. <sup>[1]</sup>	0
		0	Extended wait disabled (POR reset value).	
		1	Extended wait enabled.	
18:9	-		Reserved. Read value is undefined, only zero should be written.	NA
19	Buffer enable (B) <sup>[2]</sup>	0	Buffer disabled (POR reset value).	0
		1	Buffer enabled.	
20	Write protect (P)	0	Writes not protected (POR reset value).	0
		1	Write protected.	
31:21	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] Extended wait and page mode cannot be selected simultaneously.

[2] EMC may perform burst read access even when the buffer enable bit is cleared.

### 10.12.22 Static Memory Write Enable Delay registers (EMCStaticWaitWen0-3 - 0x2009 C204, 224, 244, 264)

The EMCStaticWaitWen0-3 registers enable you to program the delay from the chip select to the write enable. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are accessed with one wait state.

[Table 134](#) shows the bit assignments for the EMCStaticWaitWen0-3 registers.

**Table 134. Static Memory Write Enable Delay registers (EMCStaticWaitWen0-3 - address 0x2009 C204, 0x2009 C224, 0x2009 C244, 0x2009 C264) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Wait write enable (WAITWEN)		Delay from chip select assertion to write enable.	0x0
		0x0	One CCLK cycle delay between assertion of chip select and write enable (POR reset value).	
		0x1 - 0xF	(n + 1) CCLK cycle delay. The delay is (WAITWEN + 1) x tCCLK.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.23 Static Memory Output Enable Delay registers (EMCStaticWaitOen0-3 - 0x2009 C208, 228, 248, 268)

The EMCStaticWaitOen0-3 registers enable you to program the delay from the chip select or address change, whichever is later, to the output enable. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are accessed with one wait state.

[Table 135](#) shows the bit assignments for the EMCStaticWaitOen0-3 registers.

**Table 135. Static Memory Output Enable delay registers (EMCStaticWaitOen03 - address 0x2009 C208, 0x2009 C228, 0x2009 C248, 0x2009 C268) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Wait output enable (WAITOEN)		Delay from chip select assertion to output enable.	0x0
		0x0	No delay (POR reset value).	
		0x1 - 0xF	n cycle delay. The delay is WAITOEN x tCCLK.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.24 Static Memory Read Delay registers (EMCStaticWaitRd0-3 - 0x2009 C20C, 22C, 24C, 26C)

The EMCStaticWaitRd0-3 registers enable you to program the delay from the chip select to the read access. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. It is not used if the extended wait bit is enabled in the EMCStaticConfig0-3 registers. These registers are accessed with one wait state.

[Table 136](#) shows the bit assignments for the EMCStaticWaitRd0-3 registers.

**Table 136. Static Memory Read Delay registers (EMCStaticWaitRd0-3 - address 0x2009 C20C, 0x2009 C22C, 0x2009 C24C, 0x2009 C26C) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Non-page mode read wait states or asynchronous page mode read first access wait state (WAITRD)	0x0 - 0x1E 0x1F	Non-page mode read or asynchronous page mode read, first read only:  (n + 1) CCLK cycles for read accesses. For non-sequential reads, the wait state time is (WAITRD + 1) x tCCLK.  32 CCLK cycles for read accesses (POR reset value).	0x1F
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.25 Static Memory Page Mode Read Delay registers (EMCStaticwaitPage0-3 - 0x2009 C210, 230, 250, 270)

The EMCStaticWaitPage0-3 registers enable you to program the delay for asynchronous page mode sequential accesses. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. This register is accessed with one wait state.

[Table 137](#) shows the bit assignments for the EMCStaticWaitPage0-3 registers.

**Table 137. Static Memory Page Mode Read Delay registers0-3 (EMCStaticWaitPage0-3 - address 0x2009 C210, 0x2009 C230, 0x2009 C250, 0x2009 C270) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Asynchronous page mode read after the first read wait states (WAITPAGE)	0x0 - 0x1E 0x1F	Number of wait states for asynchronous page mode read accesses after the first read:  (n+ 1) CCLK cycle read access time. For asynchronous page mode read for sequential reads, the wait state time for page mode accesses after the first read is (WAITPAGE + 1) x tCCLK.  32 CCLK cycle read access time (POR reset value).	0x1F
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.26 Static Memory Write Delay registers (EMCStaticWaitWr0-3 - 0x2009 C214, 234, 254, 274)

The EMCStaticWaitWr0-3 registers enable you to program the delay from the chip select to the write access. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are not used if the extended wait (EW) bit is enabled in the EMCStaticConfig register. These registers are accessed with one wait state.

[Table 138](#) shows the bit assignments for the EMCStaticWaitWr0-3 registers.

**Table 138. Static Memory Write Delay registers0-3 (EMCStaticWaitWr - address 0x2009 C214, 0x2009 C234, 0x2009 C254, 0x2009 C274) bit description**

Bit	Symbol	Value	Description	Reset Value
4:0	Write wait states (WAITWR)	0x0 - 0x1E 0x1F	SRAM wait state time for write accesses after the first read: (n + 2) CCLK cycle write access time. The wait state time for write accesses after the first read is WAITWR (n + 2) x tCCLK. 33 CCLK cycle write access time (POR reset value).	0x1F
31:5	-	-	Reserved. Read value is undefined, only zero should be written.	NA

### 10.12.27 Static Memory Turn Round Delay registers (EMCStaticWaitTurn0-3 - 0x2009 C218, 238, 258, 278)

The EMCStaticWaitTurn0-3 registers enable you to program the number of bus turnaround cycles. It is recommended that these registers are modified during system initialization, or when there are no current or outstanding transactions. This can be ensured by waiting until the EMC is idle, and then entering low-power, or disabled mode. These registers are accessed with one wait state.

[Table 139](#) shows the bit assignments for the EMCStaticWaitTurn0-3 registers.

**Table 139. Static Memory Turn Round Delay registers0-3 (EMCStaticWaitTurn0-3 - address 0x2009 C218, 0x2009 C238, 0x2009 C258, 0x2009 C278) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	Bus turnaround cycles (WAITTURN)	0x0 - 0xE 0xF	Reserved, do not use this value. (n + 1) CCLK turnaround cycles. Bus turnaround time is (WAITTURN + 1) x tCCLK. 16 CCLK turnaround cycles (POR reset value).	0xF
31:4	-	-	Reserved. Read value is undefined, only zero should be written.	NA

To prevent bus contention on the external memory data bus, the WAITTURN field controls the number of bus turnaround cycles added between static memory read and write accesses. The WAITTURN field also controls the number of turnaround cycles between static memory and dynamic memory accesses.

### 10.12.28 Delay Control register (EMCDLYCTL - 0x400F C1DC)

The EMCDLYCTL register controls on-chip programmable delays that can be used to fine tune timing to external SDRAM memories. Delays can be configured in increments of approximately 250 picoseconds up to a maximum of roughly 7.75 ns. See [Section 10.5.6](#) for an overview of the programmable delays. [Figure 14](#) shows the detailed connections of the programmable delays. [Table 140](#) shows the bit assignments in EMCDLYCTL.

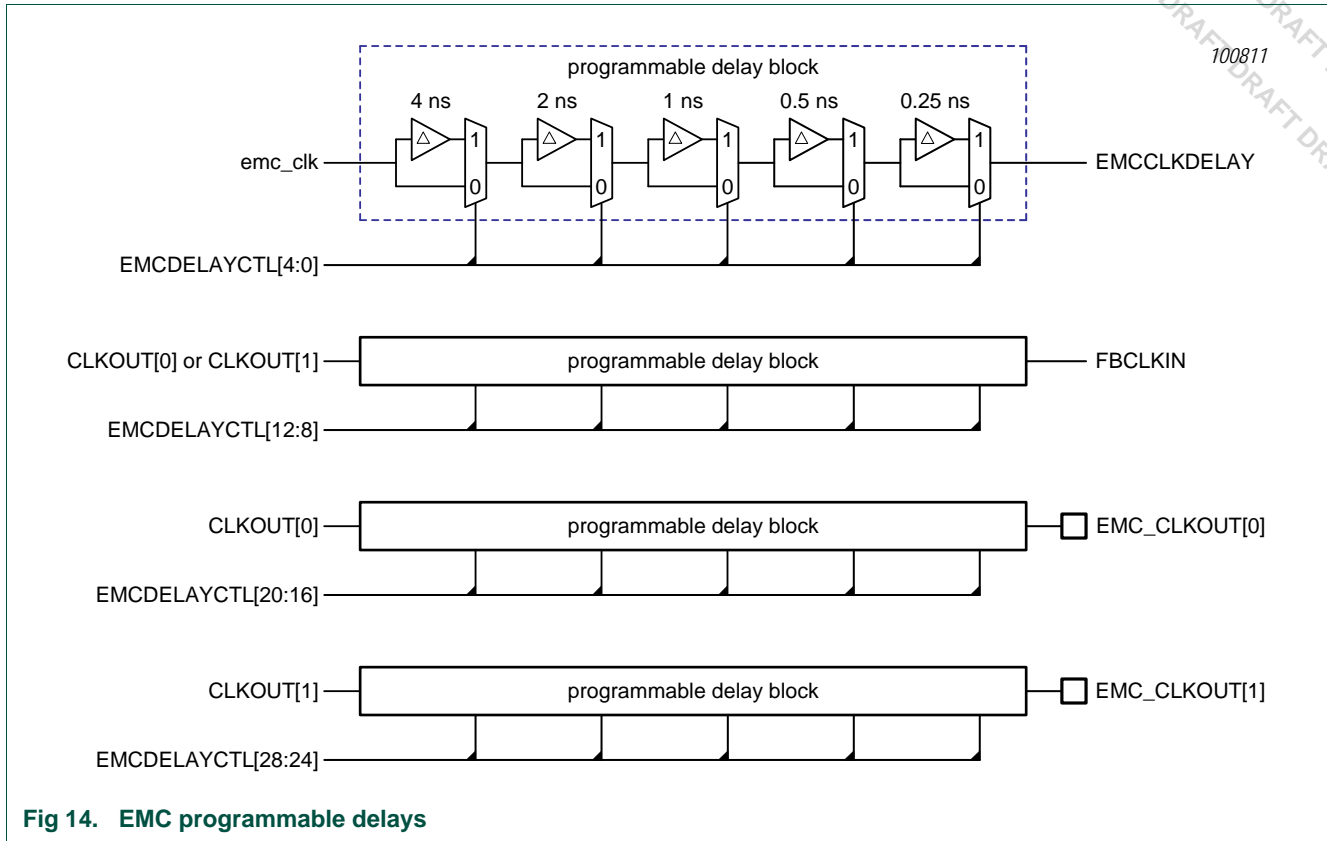


Fig 14. EMC programmable delays

Table 140. Delay Control register (EMCDLYCTL - 0x400F C1DC) bit description

Bit	Symbol	Description	Reset Value
4:0	CMDDLY	Programmable delay value for EMC outputs in command delayed mode. See <a href="#">Section 10.12.6</a> . The delay amount is roughly (CMDDLY+1) * 250 picoseconds.	0x10
11:5	-	Reserved. Read value is undefined, only zero should be written.	NA
12:8	FBCLKDLY	Programmable delay value for the feedback clock that controls input data sampling. See <a href="#">Section 10.5.3</a> . The delay amount is roughly (FBCLKDLY+1) * 250 picoseconds.	0x02
15:9	-	Reserved. Read value is undefined, only zero should be written.	NA
20:16	CLKOUT0DLY	Programmable delay value for the CLKOUT[0] output. This would typically be used in clock delayed mode. See <a href="#">Section 10.12.6</a> The delay amount is roughly (CLKOUT0DLY+1) * 250 picoseconds.	0
23:21	-	Reserved. Read value is undefined, only zero should be written.	NA
28:24	CLKOUT1DLY	Programmable delay value for the CLKOUT[1] output. This would typically be used in clock delayed mode. See <a href="#">Section 10.12.6</a> The delay amount is roughly (CLKOUT1DLY+1) * 250 picoseconds.	0
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA

10.12.29 EMC Calibration register (EMCCAL - 0x400F C1E0)

The EMCCAL register allows calibration of the EMC programmable delays by providing a real-time representation of the value of those delays. Delay settings that are in use in the application can be calibrated to compensate for intrinsic differences between devices, and for changes in ambient conditions. Figure 15 below shows the delay calibration circuit. Table 141 shows the bit assignments in EMCCAL.

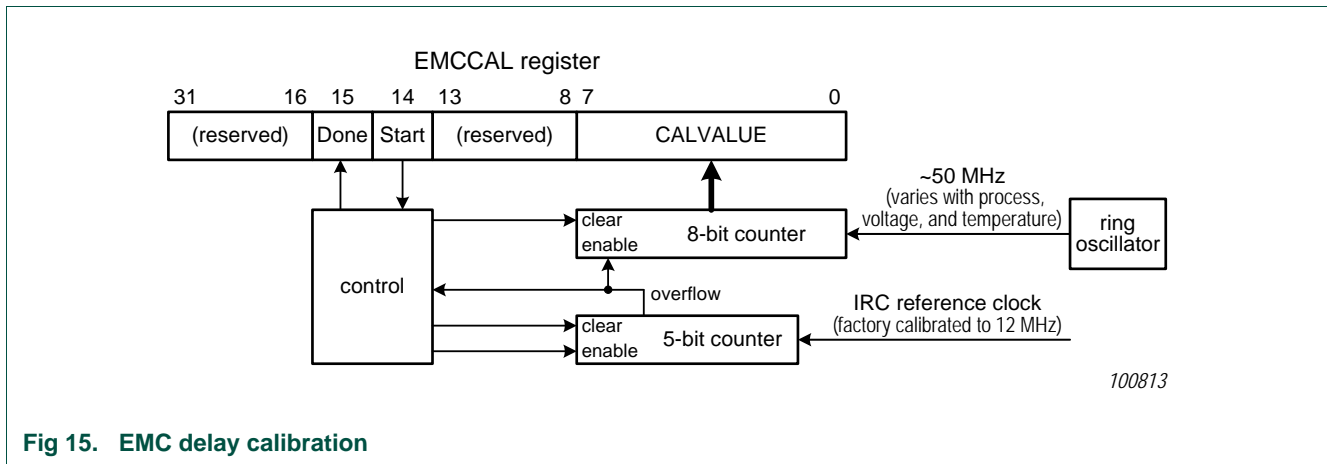


Fig 15. EMC delay calibration

Table 141. EMC Calibration register (EMCCAL - 0x400F C1E0) bit description

Bit	Symbol	Description	Reset Value
7:0	CALVALUE	Returns the count of the approximately 50 MHz ring oscillator that occur during 32 clocks of the IRC oscillator. This represents the composite effect of processing variation, internal regulator supply voltage, and ambient temperature.	0
13:8	-	Reserved. Read value is undefined, only zero should be written.	NA
14	START	Start control bit for the EMC calibration counter. Writing a 1 to this bit begins the measurement process. This bit is cleared automatically when the measurement is complete.	0
15	DONE	Measurement completion flag. this bit is set when a calibration measurement is completed. This bit is cleared automatically when the START bit is set.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

Procedure for calibrating programmable delays

1. Write 1 to the START bit of the EMCCAL register.
2. Wait until the DONE bit of the same register becomes 1. Other operations can be done during this time, the calibration requires 32 clocks of the 12 MHz IRC clock, or about 2.7 microseconds.
3. Read the calibration value from the bottom 8 bits of the EMCCAL register. A typical value at room temperature is 0x86.
4. Adjust one or more programmable delays if needed based on the calibration result.

The calibration procedure should typically be repeated periodically, depending on how rapidly ambient conditions may change in the application environment.



### 10.13 External memory interface

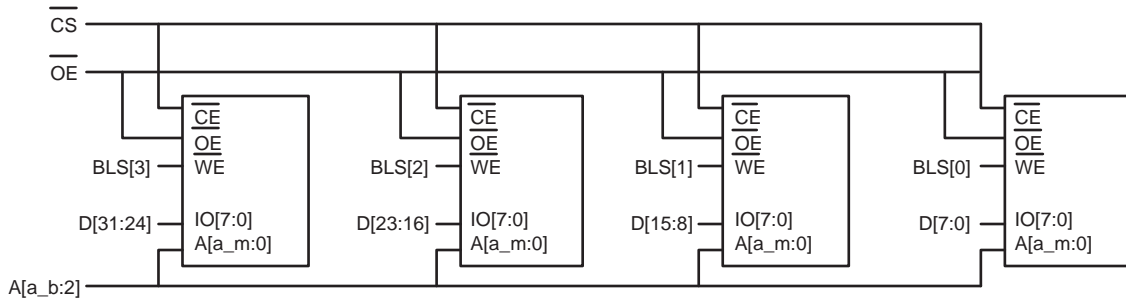
External memory interfacing depends on the bank width (32, 16 or 8 bit selected via MW bits in corresponding EMCStaticConfig register).

If a memory bank is configured to be 32 bits wide, address lines A0 and A1 can be used as non-address lines. If a memory bank is configured to 16 bits wide, A0 is not required. However, 8 bit wide memory banks do require all address lines down to A0. Configuring A1 and/or A0 line(s) to provide address or non-address function is accomplished using the IOCON registers (see [Section 8.4.1](#)).

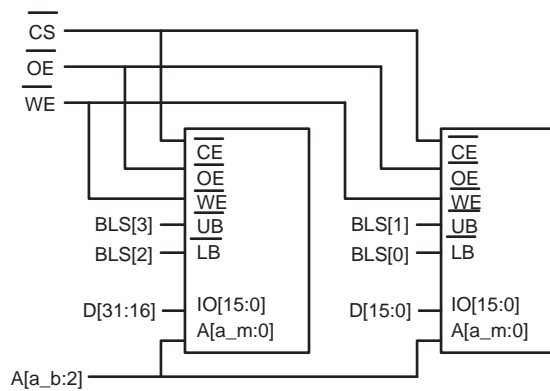
Symbol "a\_b" in the following figures refers to the highest order address line in the data bus. Symbol "a\_m" refers to the highest order address line of the memory chip used in the external memory interface.

If the external memory is used as external boot memory for flashless devices, refer to [Section 7.2](#) on how to connect the EMC. Also see [Section 7.2 "Boot control"](#). The memory bank width for memory banks 1 and 2 is determined by the setting of the two BOOT1/0 pins.

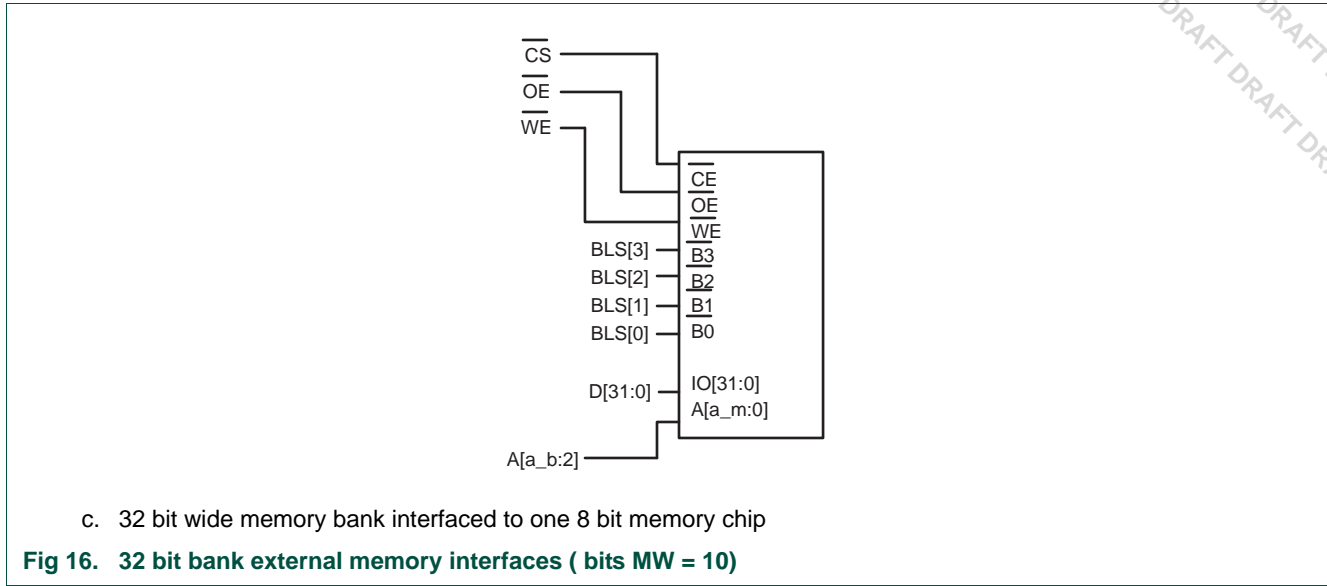
#### 10.13.1 32-bit wide memory bank connection



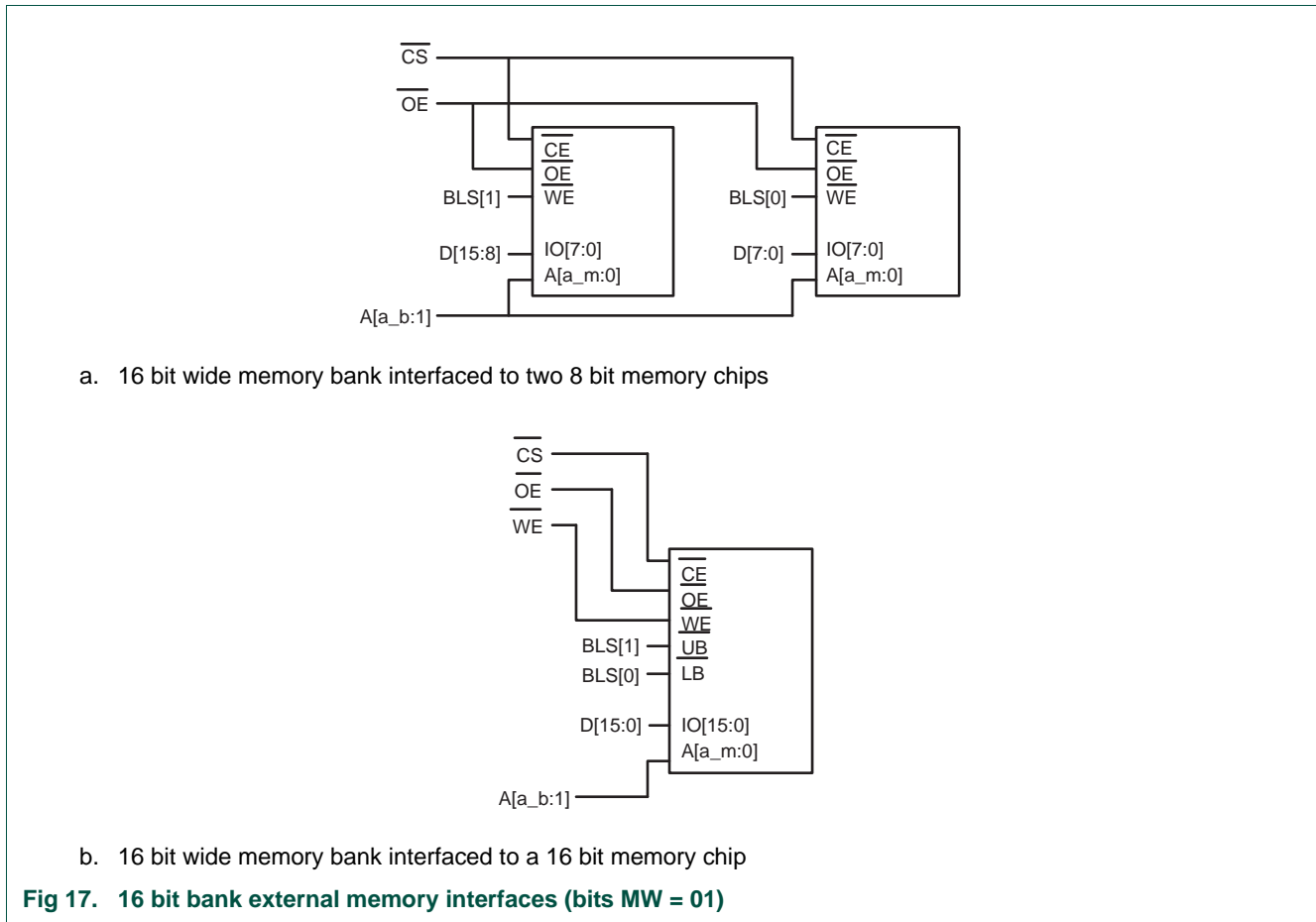
a. 32 bit wide memory bank interfaced to four 8 bit memory chips



b. 32 bit wide memory bank interfaced to two 16 bit memory chips



10.13.2 16-bit wide memory bank connection



10.13.3 8-bit wide memory bank connection

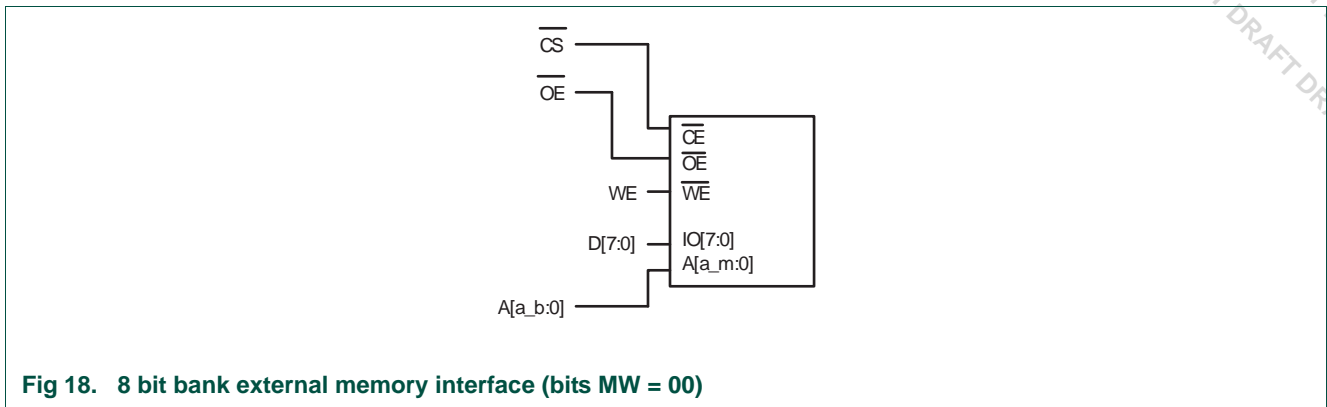


Fig 18. 8 bit bank external memory interface (bits MW = 00)

10.13.4 Memory configuration example

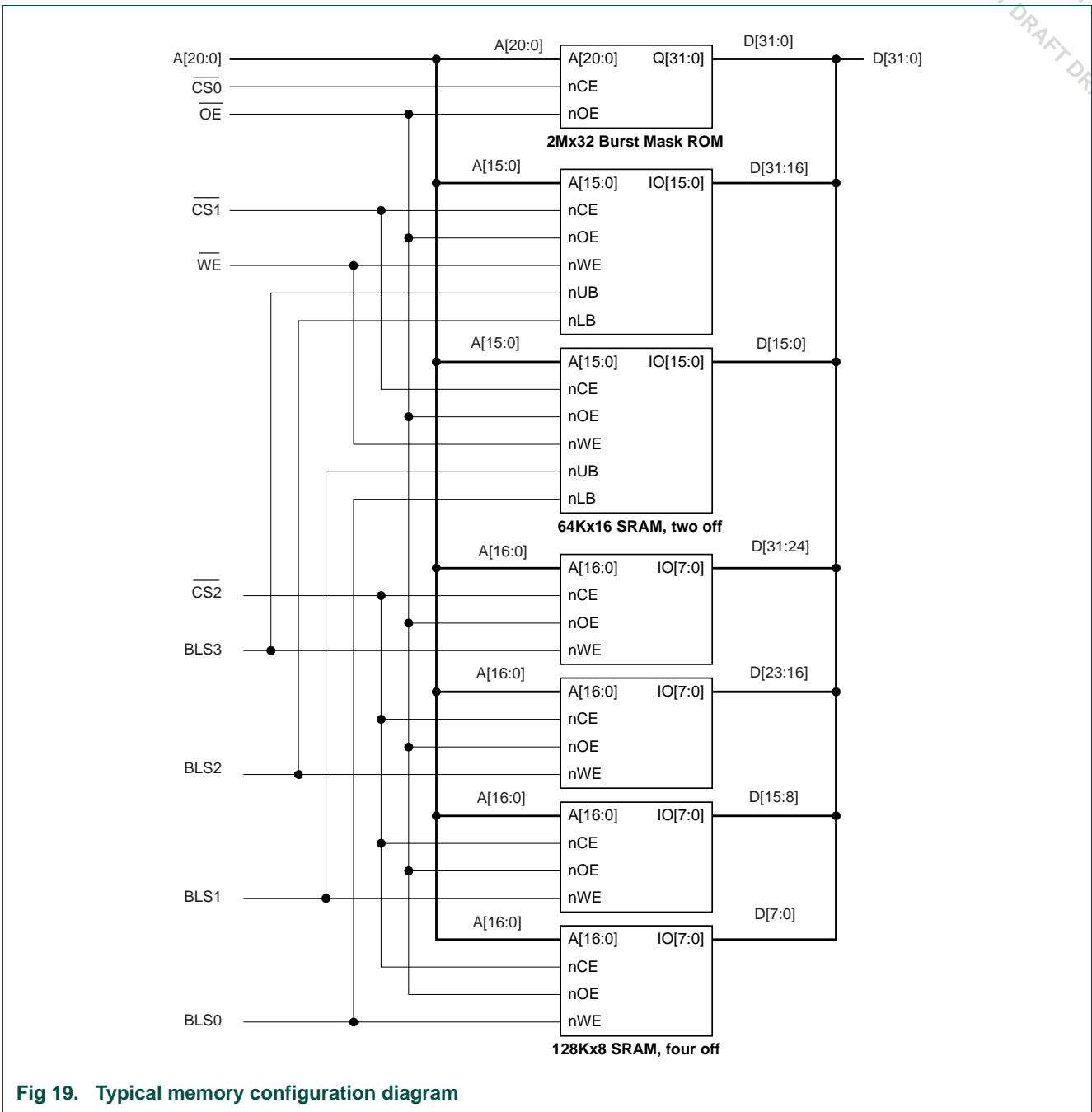


Fig 19. Typical memory configuration diagram

### 11.1 Basic configuration

The Ethernet controller is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCENET.  
**Remark:** On reset, the Ethernet block is disabled (PCENET = 0).
2. Clock: See [Table 30](#).
3. Pins: Enable Ethernet pins and select their modes through the IOCON registers, see [Section 8.4.1](#).
4. Wake-up: Activity on the Ethernet port can wake up the microcontroller from Power-down mode, see [Section 4.7.8](#).
5. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
6. Initialization: See [Section 11.17.2](#).

### 11.2 Introduction

The Ethernet block contains a full featured 10 Mbps or 100 Mbps Ethernet MAC (Media Access Controller) designed to provide optimized performance through the use of DMA hardware acceleration. Features include a generous suite of control registers, half or full duplex operation, flow control, control frames, hardware acceleration for transmit retry, receive packet filtering and wake-up on LAN activity. Automatic frame transmission and reception with Scatter-Gather DMA off-loads many operations from the CPU.

The Ethernet block is an AHB master that drives the AHB bus matrix. Through the matrix, it has access to all on-chip RAM memories. A recommended use of RAM by the Ethernet is to use one of the RAM blocks exclusively for Ethernet traffic. That RAM would then be accessed only by the Ethernet and the CPU, and possibly the GPDMA, giving maximum bandwidth to the Ethernet function.

The Ethernet block interfaces between an off-chip Ethernet PHY using the MII (Media Independent Interface) or RMII (reduced MII) protocol and the on-chip MIIM (Media Independent Interface Management) serial bus, also referred to as MDIO (Management Data Input/Output).

**Table 142. Ethernet acronyms, abbreviations, and definitions**

Acronym or Abbreviation	Definition
AHB	Advanced High-performance bus
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
Double-word	64-bit entity
FCS	Frame Check Sequence (CRC)
Fragment	A (part of an) Ethernet frame; one or multiple fragments can add up to a single Ethernet frame.

Table 142. Ethernet acronyms, abbreviations, and definitions

Acronym or Abbreviation	Definition
Frame	An Ethernet frame consists of destination address, source address, length type field, payload and frame check sequence.
Half-word	16-bit entity
LAN	Local Area Network
MAC	Media Access Control sublayer
MII	Media Independent Interface
MIIM	MII management
Octet	An 8-bit data entity, used in lieu of "byte" by IEEE 802.3
Packet	A frame that is transported across Ethernet; a packet consists of a preamble, a start of frame delimiter and an Ethernet frame.
PHY	Ethernet Physical Layer
RMII	Reduced MII
Rx	Receive
TCP/IP	Transmission Control Protocol / Internet Protocol. The most common high-level protocol used with Ethernet.
Tx	Transmit
VLAN	Virtual LAN
WoL	Wake-up on LAN
Word	32-bit entity

## 11.3 Features

- Ethernet standards support:
  - Supports 10 or 100 Mbps PHY devices including 10 Base-T, 100 Base-TX, 100 Base-FX, and 100 Base-T4.
  - Fully compliant with IEEE standard 802.3.
  - Fully compliant with 802.3x Full Duplex Flow Control and Half Duplex back pressure.
  - Flexible transmit and receive frame options.
  - VLAN frame support.
- Memory management:
  - Independent transmit and receive buffers memory mapped to shared SRAM.
  - DMA managers with scatter/gather DMA and arrays of frame descriptors.
  - Memory traffic optimized by buffering and prefetching.
- Enhanced Ethernet features:
  - Receive filtering.
  - Multicast and broadcast frame support for both transmit and receive.
  - Optional automatic FCS insertion (CRC) for transmit.
  - Selectable automatic transmit frame padding.
  - Over-length frame support for both transmit and receive allows any length frames.

- Promiscuous receive mode.
- Automatic collision backoff and frame retransmission.
- Includes power management by clock switching.
- Wake-on-LAN power management support allows system wake-up: using the receive filters or a magic frame detection filter.
- Physical interface:
  - Attachment of external PHY chip through standard Media Independent Interface (MII) or standard Reduced MII (RMII) interface, software selectable.
  - PHY register access is available via the Media Independent Interface Management (MIIM) interface.

### 11.4 Architecture and operation

Figure 20 shows the internal architecture of the Ethernet block.

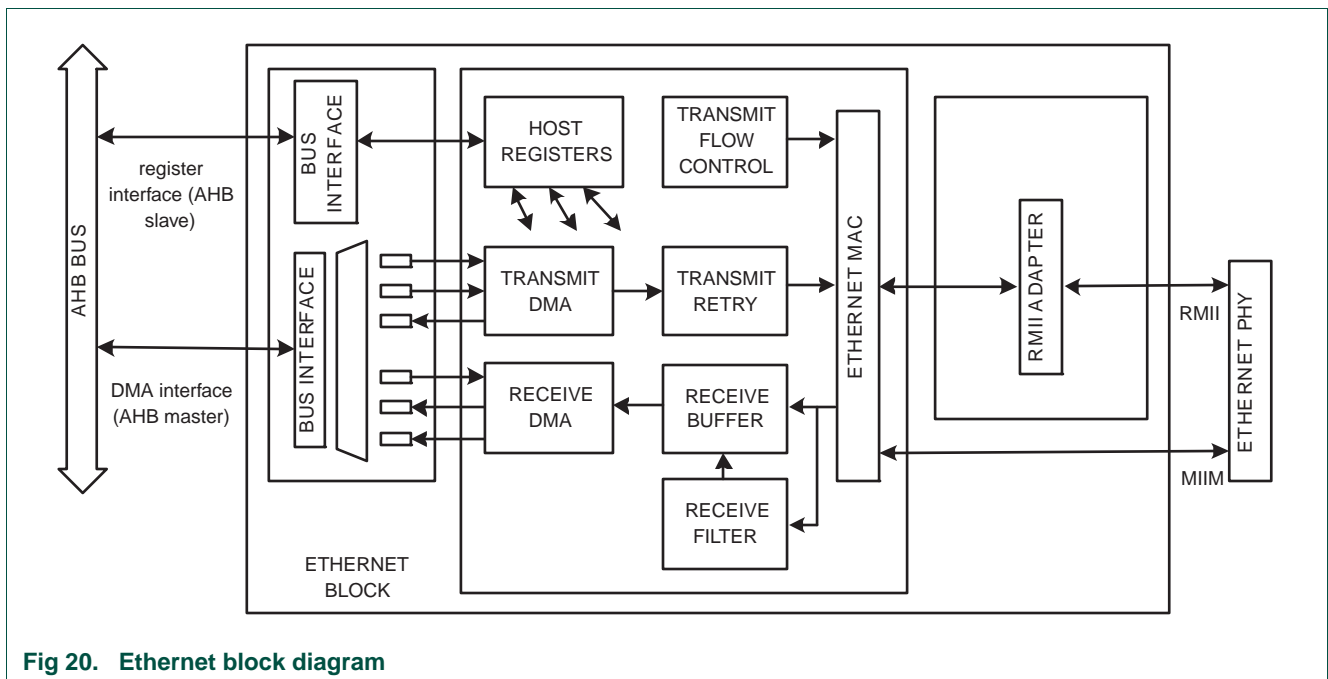


Fig 20. Ethernet block diagram

The block diagram for the Ethernet block consists of:

- The host registers module containing the registers in the software view and handling AHB accesses to the Ethernet block. The host registers connect to the transmit and receive data path as well as the MAC.
- The DMA to AHB interface. This provides an AHB master connection that allows the Ethernet block to access on-chip SRAM for reading of descriptors, writing of status, and reading and writing data buffers.
- The Ethernet MAC, which interfaces to the off-chip PHY via an MII or RMII interface.
- The transmit data path, including:

- The transmit DMA manager which reads descriptors and data from memory and writes status to memory.
- The transmit retry module handling Ethernet retry and abort situations.
- The transmit flow control module which can insert Ethernet pause frames.
- The receive data path, including:
  - The receive DMA manager which reads descriptors from memory and writes data and status to memory.
  - The Ethernet MAC which detects frame types by parsing part of the frame header.
  - The receive filter which can filter out certain Ethernet frames by applying different filtering schemes.
  - The receive buffer implementing a delay for receive frames to allow the filter to filter out certain frames before storing them to memory.

## 11.5 DMA engine functions

The Ethernet block is designed to provide optimized performance via DMA hardware acceleration. Independent scatter/gather DMA engines connected to the AHB bus off-load many data transfers from the CPU.

Descriptors, which are stored in memory, contain information about fragments of incoming or outgoing Ethernet frames. A fragment may be an entire frame or a much smaller amount of data. Each descriptor contains a pointer to a memory buffer that holds data associated with a fragment, the size of the fragment buffer, and details of how the fragment will be transmitted or received.

Descriptors are stored in arrays in memory, which are located by pointer registers in the Ethernet block. Other registers determine the size of the arrays, point to the next descriptor in each array that will be used by the DMA engine, and point to the next descriptor in each array that will be used by the Ethernet device driver.

## 11.6 Overview of DMA operation

The DMA engine makes use of a Receive descriptor array and a Transmit descriptor array in memory. All or part of an Ethernet frame may be contained in a memory buffer associated with a descriptor. When transmitting, the transmit DMA engine uses as many descriptors as needed (one or more) to obtain (gather) all of the parts of a frame, and sends them out in sequence. When receiving, the receive DMA engine also uses as many descriptors as needed (one or more) to find places to store (scatter) all of the data in the received frame.

The base address registers for the descriptor array, registers indicating the number of descriptor array entries, and descriptor array input/output pointers are contained in the Ethernet block. The descriptor entries and all transmit and receive packet data are stored in memory which is not a part of the Ethernet block. The descriptor entries tell where related frame data is stored in memory, certain aspects of how the data is handled, and the result status of each Ethernet transaction.



Hardware in the DMA engine controls how data incoming from the Ethernet MAC is saved to memory, causes fragment related status to be saved, and advances the hardware receive pointer for incoming data. Driver software must handle the disposition of received data, changing of descriptor data addresses (to avoid unnecessary data movement), and advancing the software receive pointer. The two pointers create a circular queue in the descriptor array and allow both the DMA hardware and the driver software to know which descriptors (if any) are available for their use, including whether the descriptor array is empty or full.

Similarly, driver software must set up pointers to data that will be transmitted by the Ethernet MAC, giving instructions for each fragment of data, and advancing the software transmit pointer for outgoing data. Hardware in the DMA engine reads this information and sends the data to the Ethernet MAC interface when possible, updating the status and advancing the hardware transmit pointer.

### 11.7 Ethernet packet

Figure 21 illustrates the different fields in an Ethernet packet.

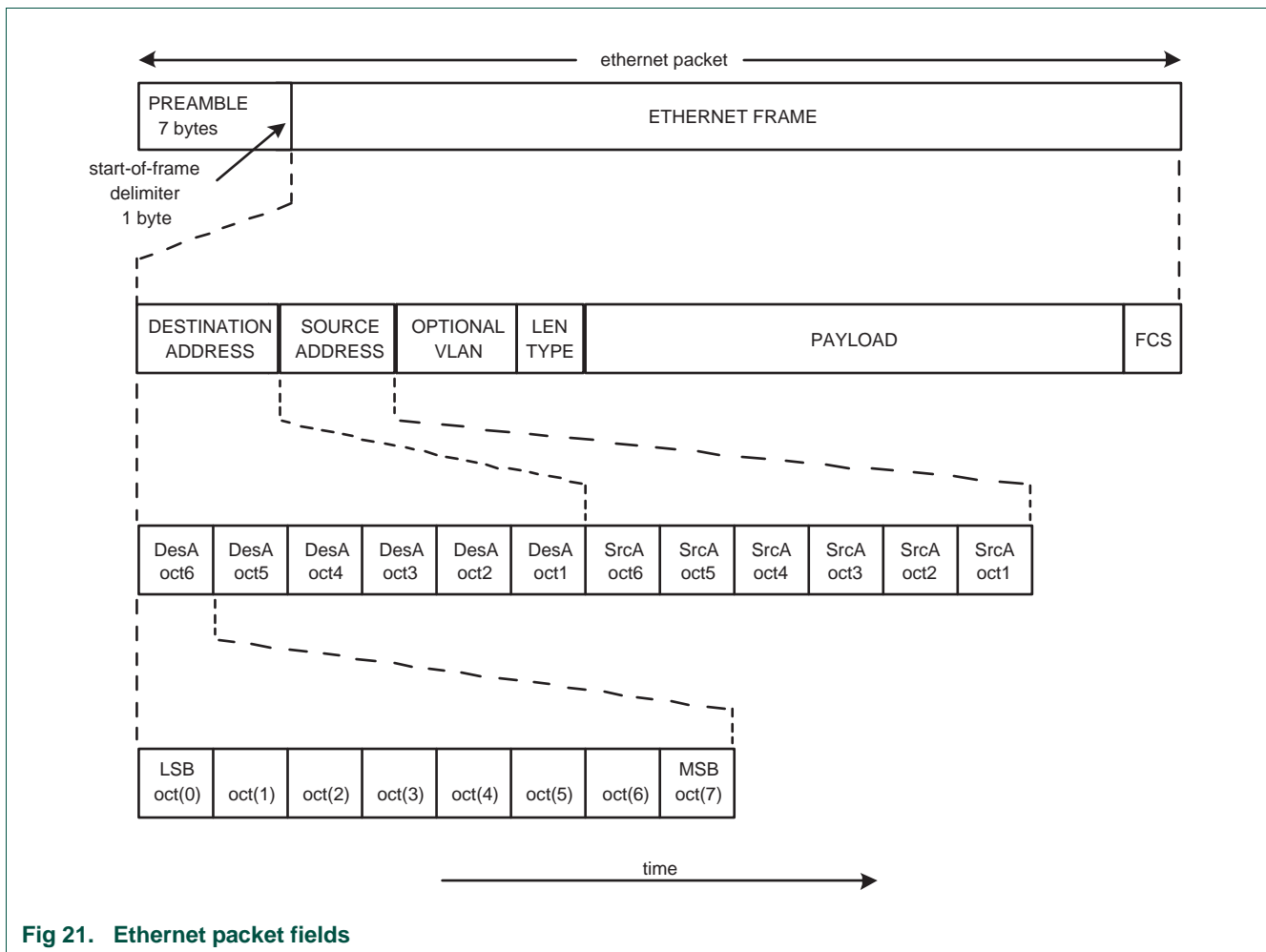


Fig 21. Ethernet packet fields

A packet consists of a preamble, a start-of-frame delimiter and an Ethernet frame.

The Ethernet frame consists of the destination address, the source address, an optional VLAN field, the length/type field, the payload and the frame check sequence.

Each address consists of 6 bytes where each byte consists of 8 bits. Bits are transferred starting with the least significant bit.

## 11.8 Overview

### 11.8.1 Partitioning

The Ethernet block and associated device driver software offer the functionality of the Media Access Control (MAC) sublayer of the data link layer in the OSI reference model (see IEEE std 802.3). The MAC sublayer offers the service of transmitting and receiving frames to the next higher protocol level, the MAC client layer, typically the Logical Link Control sublayer. The device driver software implements the interface to the MAC client layer. It sets up registers in the Ethernet block, maintains descriptor arrays pointing to frames in memory and receives results back from the Ethernet block through interrupts. When a frame is transmitted, the software partially sets up the Ethernet frames by providing pointers to the destination address field, source address field, the length/type field, the MAC client data field and optionally the CRC in the frame check sequence field. Preferably concatenation of frame fields should be done by using the scatter/gather functionality of the Ethernet core to avoid unnecessary copying of data. The hardware adds the preamble and start frame delimiter fields and can optionally add the CRC, if requested by software. When a packet is received the hardware strips the preamble and start frame delimiter and passes the rest of the packet - the Ethernet frame - to the device driver, including destination address, source address, length/type field, MAC client data and frame check sequence (FCS).

Apart from the MAC, the Ethernet block contains receive and transmit DMA managers that control receive and transmit data streams between the MAC and the AHB interface. Frames are passed via descriptor arrays located in host memory, so that the hardware can process many frames without software/CPU support. Frames can consist of multiple fragments that are accessed with scatter/gather DMA. The DMA managers optimize memory bandwidth using prefetching and buffering.

A receive filter block is used to identify received frames that are not addressed to this Ethernet station, so that they can be discarded. The Rx filters include a perfect address filter and a hash filter.

Wake-on-LAN power management support makes it possible to wake the system up from a power-down state -a state in which some of the clocks are switched off -when wake-up frames are received over the LAN. Wake-up frames are recognized by the receive filtering modules or by a Magic Frame detection technology. System wake-up occurs by triggering an interrupt.

An interrupt logic block raises and masks interrupts and keeps track of the cause of interrupts. The interrupt block sends an interrupt request signal to the host system. Interrupts can be enabled, cleared and set by software.

Support for IEEE 802.3/clause 31 flow control is implemented in the flow control block. Receive flow control frames are automatically handled by the MAC. Transmit flow control frames can be initiated by software. In half duplex mode, the flow control module will generate back pressure by sending out continuous preamble only, interrupted by pauses to prevent the jabber limit from being exceeded.

The Ethernet block has both a standard Media Independent Interface (MII) bus and a Reduced Media Independent Interface (RMII) to connect to an external Ethernet PHY chip. MII or RMII mode can be selected by the RMII bit in the Command register. The standard nibble-wide MII interface allows a low speed data connection to the PHY chip: 2.5 MHz at 10 Mbps or 25 MHz at 100 Mbps. The RMII interface allows a low pin count double clock data connection to the PHY. Registers in the PHY chip are accessed via the AHB interface through the serial management connection of the MIIM bus, typically operating at 2.5 MHz.

### 11.8.2 Example PHY Devices

Some examples of compatible PHY devices are shown in [Table 143](#).

**Table 143. Example PHY Devices**

Manufacturer	Part Number(s)
Broadcom	BCM5221
ICS	ICS1893
Intel	LXT971A
LSI Logic	L80223, L80225, L80227
Micrel	KS8721
National	DP83847, DP83846, DP83843
SMSC	LAN83C185

## 11.9 Pin description

[Table 144](#) shows the signals used for connecting the Media Independent Interface (MII), and [Table 145](#) shows the signals used for connecting the Reduced Media Independent Interface (RMII) to the external PHY.

**Table 144. Ethernet MII pin descriptions**

Pin Name	Type	Pin Description
ENET_TX_EN	Output	Transmit data enable, active low.
ENET_TXD[3:0]	Output	Transmit data, 4 bits.
ENET_TX_ER	Output	Transmit error.
ENET_TX_CLK	Input	Transmitter clock.
ENET_RX_DV	Input	Receive data valid.
ENET_RXD[3:0]	Input	Receive data, 4 bits.
ENET_RX_ER	Input	Receive error.
ENET_RX_CLK	Input	Receive clock.
ENET_COL	Input	Collision detect.
ENET_CRD	Input	Carrier sense.

**Table 145. Ethernet RMII pin descriptions**

Pin Name	Type	Pin Description
ENET_TX_EN	Output	Transmit data enable, active low.
ENET_TXD[1:0]	Output	Transmit data, 2 bits
ENET_RXD[1:0]	Input	Receive data, 2 bits.
ENET_RX_ER	Input	Receive error.
ENET_CRD	Input	ENET_CRD_DV. Carrier sense/data valid.
ENET_RX_CLK	Input	ENET_REF_CLK. Reference clock.

[Table 146](#) shows the signals used for Media Independent Interface Management (MIIM) to the external PHY.

**Table 146. Ethernet MIIM pin descriptions**

Pin Name	Type	Pin Description
ENET_MDC	Output	MIIM clock.
ENET_MDIO	Input/Output	MI data input and output

## 11.10 Registers and software interface

The software interface of the Ethernet block consists of a register view and the format definitions for the transmit and receive descriptors. These two aspects are addressed in the next two subsections.

### 11.10.1 Register map

[Table 147](#) lists the registers, register addresses and other basic information. The total AHB address space required is 4 kilobytes.

After a hard reset or a soft reset via the RegReset bit of the Command register all bits in all registers are reset to 0 unless stated otherwise in the following register descriptions.

Some registers will have unused bits which will return a 0 on a read via the AHB interface. Writing to unused register bits of an otherwise writable register will not have side effects.

The register map consists of registers in the Ethernet MAC and registers around the core for controlling DMA transfers, flow control and filtering.

Reading from reserved addresses or reserved bits leads to unpredictable data. Writing to reserved addresses or reserved bits has no effect.

Reading of write-only registers will return a read error on the AHB interface. Writing of read-only registers will return a write error on the AHB interface.

**Table 147. Ethernet register definitions**

Name	Description	Access	Reset Value	Address	Table
<b>MAC registers</b>					
MAC1	MAC configuration register 1.	R/W	0x8000	0x2008 4000	<a href="#">148</a>
MAC2	MAC configuration register 2.	R/W	0	0x2008 4004	<a href="#">149</a>
IPGT	Back-to-Back Inter-Packet-Gap register.	R/W	0	0x2008 4008	<a href="#">151</a>
IPGR	Non Back-to-Back Inter-Packet-Gap register.	R/W	0	0x2008 400C	<a href="#">152</a>
CLRT	Collision window / Retry register.	R/W	0x370F	0x2008 4010	<a href="#">153</a>
MAXF	Maximum Frame register.	R/W	0x0600	0x2008 4014	<a href="#">154</a>
SUPP	PHY Support register.	R/W	0	0x2008 4018	<a href="#">155</a>
TEST	Test register.	R/W	0	0x2008 401C	<a href="#">156</a>
MCFG	MII Mgmt Configuration register.	R/W	0	0x2008 4020	<a href="#">157</a>
MCMD	MII Mgmt Command register.	R/W	0	0x2008 4024	<a href="#">159</a>
MADR	MII Mgmt Address register.	R/W	0	0x2008 4028	<a href="#">160</a>
MWTD	MII Mgmt Write Data register.	WO	0	0x2008 402C	<a href="#">161</a>
MRDD	MII Mgmt Read Data register.	RO	0	0x2008 4030	<a href="#">162</a>
MIND	MII Mgmt Indicators register.	RO	0	0x2008 4034	<a href="#">163</a>
SA0	Station Address 0 register.	R/W	0	0x2008 4040	<a href="#">164</a>
SA1	Station Address 1 register.	R/W	0	0x2008 4044	<a href="#">165</a>
SA2	Station Address 2 register.	R/W	0	0x2008 4048	<a href="#">166</a>
<b>Control registers</b>					
Command	Command register.	R/W	0	0x2008 4100	<a href="#">167</a>

Table 147. Ethernet register definitions

Name	Description	Access	Reset Value	Address	Table
Status	Status register.	RO	0	0x2008 4104	<a href="#">168</a>
RxDescriptor	Receive descriptor base address register.	R/W	0	0x2008 4108	<a href="#">169</a>
RxStatus	Receive status base address register.	R/W	0	0x2008 410C	<a href="#">170</a>
RxDescriptorNumber	Receive number of descriptors register.	R/W	0	0x2008 4110	<a href="#">171</a>
RxProduceIndex	Receive produce index register.	RO	0	0x2008 4114	<a href="#">172</a>
RxConsumeIndex	Receive consume index register.	R/W	0	0x2008 4118	<a href="#">173</a>
TxDescriptor	Transmit descriptor base address register.	R/W	0	0x2008 411C	<a href="#">174</a>
TxStatus	Transmit status base address register.	R/W	0	0x2008 4120	<a href="#">175</a>
TxDescriptorNumber	Transmit number of descriptors register.	R/W	0	0x2008 4124	<a href="#">176</a>
TxProduceIndex	Transmit produce index register.	R/W	0	0x2008 4128	<a href="#">177</a>
TxConsumeIndex	Transmit consume index register.	RO	0	0x2008 412C	<a href="#">178</a>
TSV0	Transmit status vector 0 register.	RO	0	0x2008 4158	<a href="#">179</a>
TSV1	Transmit status vector 1 register.	RO	0	0x2008 415C	<a href="#">180</a>
RSV	Receive status vector register.	RO	0	0x2008 4160	<a href="#">181</a>
FlowControlCounter	Flow control counter register.	R/W	0	0x2008 4170	<a href="#">182</a>
FlowControlStatus	Flow control status register.	RO	0	0x2008 4174	<a href="#">183</a>
<b>Rx filter registers</b>					
RxFilterCtrl	Receive filter control register.	R/W	0	0x2008 4200	<a href="#">184</a>
RxFilterWoLStatus	Receive filter WoL status register.	RO	0	0x2008 4204	<a href="#">185</a>
RxFilterWoLClear	Receive filter WoL clear register.	WO	0	0x2008 4208	<a href="#">186</a>
HashFilterL	Hash filter table LSBs register.	R/W	0	0x2008 4210	<a href="#">187</a>
HashFilterH	Hash filter table MSBs register.	R/W	0	0x2008 4214	<a href="#">188</a>
<b>Module control registers</b>					
IntStatus	Interrupt status register.	RO	0	0x2008 4FE0	<a href="#">189</a>
IntEnable	Interrupt enable register.	R/W	0	0x2008 4FE4	<a href="#">190</a>
IntClear	Interrupt clear register.	WO	0	0x2008 4FE8	<a href="#">191</a>
IntSet	Interrupt set register.	WO	0	0x2008 4FEC	<a href="#">192</a>
PowerDown	Power-down register.	R/W	0	0x2008 4FF4	<a href="#">193</a>

The third column in the table lists the accessibility of the register: read-only, write-only, read/write.

All AHB register write transactions except for accesses to the interrupt registers are posted i.e. the AHB transaction will complete before write data is actually committed to the register. Accesses to the interrupt registers will only be completed by accepting the write data when the data has been committed to the register.

## 11.11 Ethernet MAC register definitions

This section defines the bits in the individual registers of the Ethernet block register map.

### 11.11.1 MAC Configuration Register 1 (MAC1 - 0x2008 4000)

The MAC configuration register 1 (MAC1) has an address of 0x2008 4000. Its bit definition is shown in [Table 148](#).

**Table 148. MAC Configuration register 1 (MAC1 - address 0x2008 4000) bit description**

Bit	Symbol	Function	Reset value
0	RECEIVE ENABLE	Set this to allow receive frames to be received. Internally the MAC synchronizes this control bit to the incoming receive stream.	0
1	PASS ALL RECEIVE FRAMES	When enabled (set to '1'), the MAC will pass all frames regardless of type (normal vs. Control). When disabled, the MAC does not pass valid Control frames.	0
2	RX FLOW CONTROL	When enabled (set to '1'), the MAC acts upon received PAUSE Flow Control frames. When disabled, received PAUSE Flow Control frames are ignored.	0
3	TX FLOW CONTROL	When enabled (set to '1'), PAUSE Flow Control frames are allowed to be transmitted. When disabled, Flow Control frames are blocked.	0
4	LOOPBACK	Setting this bit will cause the MAC Transmit interface to be looped back to the MAC Receive interface. Clearing this bit results in normal operation.	0
7:5	-	Unused	0x0
8	RESET TX	Setting this bit will put the Transmit Function logic in reset.	0
9	RESET MCS / TX	Setting this bit resets the MAC Control Sublayer / Transmit logic. The MCS logic implements flow control.	0
10	RESET RX	Setting this bit will put the Ethernet receive logic in reset.	0
11	RESET MCS / RX	Setting this bit resets the MAC Control Sublayer / Receive logic. The MCS logic implements flow control.	0x0
13:12	-	Reserved. Read value is undefined, only zero should be written.	0x0
14	SIMULATION RESET	Setting this bit will cause a reset to the random number generator within the Transmit Function.	0
15	SOFT RESET	Setting this bit will put all modules within the MAC in reset except the Host Interface.	1
31:16	-	Reserved. Read value is undefined, only zero should be written.	0x0

### 11.11.2 MAC Configuration Register 2 (MAC2 - 0x2008 4004)

The MAC configuration register 2 (MAC2) has an address of 0x2008 4004. Its bit definition is shown in [Table 149](#).

**Table 149. MAC Configuration register 2 (MAC2 - address 0x2008 4004) bit description**

Bit	Symbol	Function	Reset value
0	FULL-DUPLEX	When enabled (set to '1'), the MAC operates in Full-Duplex mode. When disabled, the MAC operates in Half-Duplex mode.	0
1	FRAME LENGTH CHECKING	When enabled (set to '1'), both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length then the check is performed. Mismatches are reported in the StatusInfo word for each received frame.	0
2	HUGE FRAME ENABLE	When enabled (set to '1'), frames of any length are transmitted and received.	0

Table 149. MAC Configuration register 2 (MAC2 - address 0x2008 4004) bit description

Bit	Symbol	Function	Reset value
3	DELAYED CRC	This bit determines the number of bytes, if any, of proprietary header information that exist on the front of IEEE 802.3 frames. When 1, four bytes of header (ignored by the CRC function) are added. When 0, there is no proprietary header.	0
4	CRC ENABLE	Set this bit to append a CRC to every frame whether padding was required or not. Must be set if PAD/CRC ENABLE is set. Clear this bit if frames presented to the MAC contain a CRC.	0
5	PAD / CRC ENABLE	Set this bit to have the MAC pad all short frames. Clear this bit if frames presented to the MAC have a valid length. This bit is used in conjunction with AUTO PAD ENABLE and VLAN PAD ENABLE. See <a href="#">Table 151</a> - Pad Operation for details on the pad function.	0
6	VLAN PAD ENABLE	Set this bit to cause the MAC to pad all short frames to 64 bytes and append a valid CRC. Consult <a href="#">Table 151</a> - Pad Operation for more information on the various padding features. <b>Note:</b> This bit is ignored if PAD / CRC ENABLE is cleared.	0
7	AUTO DETECT PAD ENABLE	Set this bit to cause the MAC to automatically detect the type of frame, either tagged or un-tagged, by comparing the two octets following the source address with 0x8100 (VLAN Protocol ID) and pad accordingly. <a href="#">Table 151</a> - Pad Operation provides a description of the pad function based on the configuration of this register. <b>Note:</b> This bit is ignored if PAD / CRC ENABLE is cleared.	0
8	PURE PREAMBLE ENFORCEMENT	When enabled (set to '1'), the MAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with an incorrect preamble is discarded. When disabled, no preamble checking is performed.	0
9	LONG PREAMBLE ENFORCEMENT	When enabled (set to '1'), the MAC only allows receive packets which contain preamble fields less than 12 bytes in length. When disabled, the MAC allows any length preamble as per the Standard.	0
11:10	-	Reserved. Read value is undefined, only zero should be written.	0x0
12	NO BACKOFF	When enabled (set to '1'), the MAC will immediately retransmit following a collision rather than using the Binary Exponential Backoff algorithm as specified in the Standard.	0
13	BACK PRESSURE / NO BACKOFF	When enabled (set to '1'), after the MAC incidentally causes a collision during back pressure, it will immediately retransmit without backoff, reducing the chance of further collisions and ensuring transmit packets get sent.	0
14	EXCESS DEFER	When enabled (set to '1') the MAC will defer to carrier indefinitely as per the Standard. When disabled, the MAC will abort when the excessive deferral limit is reached.	0
31:15	-	Reserved. Read value is undefined, only zero should be written.	0x0

Table 150. Pad operation

Type	Auto detect pad enable MAC2 [7]	VLAN pad enable MAC2 [6]	Pad/CRC enable MAC2 [5]	Action
Any	x	x	0	No pad or CRC check
Any	0	0	1	Pad to 60 bytes, append CRC
Any	x	1	1	Pad to 64 bytes, append CRC
Any	1	0	1	If untagged, pad to 60 bytes and append CRC. If VLAN tagged: pad to 64 bytes and append CRC.



### 11.11.3 Back-to-Back Inter-Packet-Gap Register (IPGT - 0x2008 4008)

The Back-to-Back Inter-Packet-Gap register (IPGT) has an address of 0x2008 4008. Its bit definition is shown in [Table 151](#).

**Table 151. Back-to-back Inter-packet-gap register (IPGT - address 0x2008 4008) bit description**

Bit	Symbol	Function	Reset value
6:0	BACK-TO-BACK INTER-PACKET-GAP	This is a programmable field representing the nibble time offset of the minimum possible period between the end of any transmitted packet to the beginning of the next. In Full-Duplex mode, the register value should be the desired period in nibble times minus 3. In Half-Duplex mode, the register value should be the desired period in nibble times minus 6. In Full-Duplex the recommended setting is 0x15 (21d), which represents the minimum IPG of 960 ns (in 100 Mbps mode) or 9.6 μs (in 10 Mbps mode). In Half-Duplex the recommended setting is 0x12 (18d), which also represents the minimum IPG of 960 ns (in 100 Mbps mode) or 9.6 μs (in 10 Mbps mode).	0x0
31:7	-	Reserved. Read value is undefined, only zero should be written.	0x0

### 11.11.4 Non Back-to-Back Inter-Packet-Gap Register (IPGR - 0x2008 400C)

The Non Back-to-Back Inter-Packet-Gap register (IPGR) has an address of 0x2008 400C. Its bit definition is shown in [Table 152](#).

**Table 152. Non Back-to-back Inter-packet-gap register (IPGR - address 0x2008 400C) bit description**

Bit	Symbol	Function	Reset value
6:0	NON-BACK-TO-BACK INTER-PACKET-GAP PART2	This is a programmable field representing the Non-Back-to-Back Inter-Packet-Gap. The recommended value is 0x12 (18d), which represents the minimum IPG of 960 ns (in 100 Mbps mode) or 9.6 μs (in 10 Mbps mode).	0x0
7	-	Reserved. Read value is undefined, only zero should be written.	0x0
14:8	NON-BACK-TO-BACK INTER-PACKET-GAP PART1	This is a programmable field representing the optional carrierSense window referenced in IEEE 802.3/4.2.3.2.1 'Carrier Deference'. If carrier is detected during the timing of IPGR1, the MAC defers to carrier. If, however, carrier becomes active after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thus ensuring fair access to medium. Its range of values is 0x0 to IPGR2. The recommended value is 0xC (12d)	0x0
31:15	-	Reserved. Read value is undefined, only zero should be written.	0x0

### 11.11.5 Collision Window / Retry Register (CLRT - 0x2008 4010)

The Collision window / Retry register (CLRT) has an address of 0x2008 4010. Its bit definition is shown in [Table 153](#).

**Table 153. Collision Window / Retry register (CLRT - address 0x2008 4010) bit description**

Bit	Symbol	Function	Reset value
3:0	RETRANSMISSION MAXIMUM	This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The Standard specifies the attemptLimit to be 0xF (15d). See IEEE 802.3/4.2.3.2.5.	0xF
7:4	-	Reserved. Read value is undefined, only zero should be written.	0x0
13:8	COLLISION WINDOW	This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. The default value of 0x37 (55d) represents a 56 byte window following the preamble and SFD.	0x37
31:14	-	Reserved. Read value is undefined, only zero should be written.	NA

### 11.11.6 Maximum Frame Register (MAXF - 0x2008 4014)

The Maximum Frame register (MAXF) has an address of 0x2008 4014. Its bit definition is shown in [Table 154](#).

**Table 154. Maximum Frame register (MAXF - address 0x2008 4014) bit description**

Bit	Symbol	Function	Reset value
15:0	MAXIMUM FRAME LENGTH	This field resets to the value 0x0600, which represents a maximum receive frame of 1536 octets. An untagged maximum size Ethernet frame is 1518 octets. A tagged frame adds four octets for a total of 1522 octets. If a shorter maximum length restriction is desired, program this 16-bit field.	0x0600
31:16	-	Unused	0x0

### 11.11.7 PHY Support Register (SUPP - 0x2008 4018)

The PHY Support register (SUPP) has an address of 0x2008 4018. The SUPP register provides additional control over the RMII interface. The bit definition of this register is shown in [Table 155](#).

**Table 155. PHY Support register (SUPP - address 0x2008 4018) bit description**

Bit	Symbol	Function	Reset value
7:0	-	Unused	0x0
8	SPEED	This bit configures the Reduced MII logic for the current operating speed. When set, 100 Mbps mode is selected. When cleared, 10 Mbps mode is selected.	0
31:9	-	Unused	0x0

Unused bits in the PHY support register should be left as zeroes.

### 11.11.8 Test Register (TEST - 0x2008 401C)

The Test register (TEST) has an address of 0x2008 401C. The bit definition of this register is shown in [Table 156](#). These bits are used for testing purposes only.

**Table 156. Test register (TEST - address 0x2008 401C) bit description**

Bit	Symbol	Function	Reset value
0	SHORTCUT PAUSE QUANTA	This bit reduces the effective PAUSE quanta from 64 byte-times to 1 byte-time.	0
1	TEST PAUSE	This bit causes the MAC Control sublayer to inhibit transmissions, just as if a PAUSE Receive Control frame with a nonzero pause time parameter was received.	0
2	TEST BACKPRESSURE	Setting this bit will cause the MAC to assert backpressure on the link. Backpressure causes preamble to be transmitted, raising carrier sense. A transmit packet from the system will be sent during backpressure.	0
31:3	-	Unused	0x0

### 11.11.9 MII Mgmt Configuration Register (MCFG - 0x2008 4020)

The MII Mgmt Configuration register (MCFG) has an address of 0x2008 4020. The bit definition of this register is shown in [Table 157](#).

**Table 157. MII Mgmt Configuration register (MCFG - address 0x2008 4020) bit description**

Bit	Symbol	Function	Reset value
0	SCAN INCREMENT	Set this bit to cause the MII Management hardware to perform read cycles across a range of PHYs. When set, the MII Management hardware will perform read cycles from address 1 through the value set in PHY ADDRESS[4:0]. Clear this bit to allow continuous reads of the same PHY.	0
1	SUPPRESS PREAMBLE	Set this bit to cause the MII Management hardware to perform read/write cycles without the 32-bit preamble field. Clear this bit to cause normal cycles to be performed. Some PHYs support suppressed preamble.	0
5:2	CLOCK SELECT	This field is used by the clock divide logic in creating the MII Management Clock (MDC) which IEEE 802.3u defines to be no faster than 2.5 MHz. Some PHYs support clock rates up to 12.5 MHz, however. The AHB bus clock (HCLK) is divided by the specified amount. Refer to <a href="#">Table 158</a> below for the definition of values for this field.	0
14:6	-	Unused	0x0
15	RESET MII MGMT	This bit resets the MII Management hardware.	0
31:16	-	Unused	0x0

**Table 158. Clock select encoding**

Clock Select	Bit 5	Bit 4	Bit 3	Bit 2	Maximum AHB clock supported
Host Clock divided by 4	0	0	0	x	10
Host Clock divided by 6	0	0	1	0	15
Host Clock divided by 8	0	0	1	1	20
Host Clock divided by 10	0	1	0	0	25
Host Clock divided by 14	0	1	0	1	35
Host Clock divided by 20	0	1	1	0	50
Host Clock divided by 28	0	1	1	1	70
Host Clock divided by 36	1	0	0	0	80 <sup>[1]</sup>
Host Clock divided by 40	1	0	0	1	90 <sup>[1]</sup>
Host Clock divided by 44	1	0	1	0	100 <sup>[1]</sup>

**Table 158. Clock select encoding**

Clock Select	Bit 5	Bit 4	Bit 3	Bit 2	Maximum AHB clock supported
Host Clock divided by 48	1	0	1	1	120 <sup>[1]</sup>
Host Clock divided by 52	1	1	0	0	130 <sup>[1]</sup>
Host Clock divided by 56	1	1	0	1	140 <sup>[1]</sup>
Host Clock divided by 60	1	1	1	0	150 <sup>[1]</sup>
Host Clock divided by 64	1	1	1	1	160 <sup>[1]</sup>

[1] The maximum AHB clock rate allowed is limited to the maximum CPU clock rate for the device.

### 11.11.10 MII Mgmt Command Register (MCMD - 0x2008 4024)

The MII Mgmt Command register (MCMD) has an address of 0x2008 4024. The bit definition of this register is shown in [Table 159](#).

**Table 159. MII Mgmt Command register (MCMD - address 0x2008 4024) bit description**

Bit	Symbol	Function	Reset value
0	READ	This bit causes the MII Management hardware to perform a single Read cycle. The Read data is returned in Register MRDD (MII Mgmt Read Data).	0
1	SCAN	This bit causes the MII Management hardware to perform Read cycles continuously. This is useful for monitoring Link Fail for example.	0
31:2	-	Unused	0x0

### 11.11.11 MII Mgmt Address Register (MADR - 0x2008 4028)

The MII Mgmt Address register (MADR) has an address of 0x2008 4028. The bit definition of this register is shown in [Table 160](#).

**Table 160. MII Mgmt Address register (MADR - address 0x2008 4028) bit description**

Bit	Symbol	Function	Reset value
4:0	REGISTER ADDRESS	This field represents the 5-bit Register Address field of Mgmt cycles. Up to 32 registers can be accessed.	0x0
7:5	-	Unused	0x0
12:8	PHY ADDRESS	This field represents the 5-bit PHY Address field of Mgmt cycles. Up to 31 PHYs can be addressed (0 is reserved).	0x0
31:13	-	Unused	0x0

### 11.11.12 MII Mgmt Write Data Register (MWTD - 0x2008 402C)

The MII Mgmt Write Data register (MWTD) is a write-only register with an address of 0x2008 402C. The bit definition of this register is shown in [Table 161](#).

**Table 161. MII Mgmt Write Data register (MWTD - address 0x2008 402C) bit description**

Bit	Symbol	Function	Reset value
15:0	WRITE DATA	When written, an MII Mgmt write cycle is performed using the 16-bit data and the pre-configured PHY and Register addresses from the MII Mgmt Address register (MADR).	0x0
31:16	-	Unused	0x0

### 11.11.13 MII Mgmt Read Data Register (MRDD - 0x2008 4030)

The MII Mgmt Read Data register (MRDD) is a read-only register with an address of 0x2008 4030. The bit definition of this register is shown in [Table 162](#).

**Table 162. MII Mgmt Read Data register (MRDD - address 0x2008 4030) bit description**

Bit	Symbol	Function	Reset value
15:0	READ DATA	Following an MII Mgmt Read Cycle, the 16-bit data can be read from this location.	0x0
31:16	-	Unused	0x0

### 11.11.14 MII Mgmt Indicators Register (MIND - 0x2008 4034)

The MII Mgmt Indicators register (MIND) is a read-only register with an address of 0x2008 4034. The bit definition of this register is shown in [Table 163](#).

**Table 163. MII Mgmt Indicators register (MIND - address 0x2008 4034) bit description**

Bit	Symbol	Function	Reset value
0	BUSY	When '1' is returned - indicates MII Mgmt is currently performing an MII Mgmt Read or Write cycle.	0
1	SCANNING	When '1' is returned - indicates a scan operation (continuous MII Mgmt Read cycles) is in progress.	0
2	NOT VALID	When '1' is returned - indicates MII Mgmt Read cycle has not completed and the Read Data is not yet valid.	0
3	MII Link Fail	When '1' is returned - indicates that an MII Mgmt link fail has occurred.	0
31:4	-	Unused	0x0

Here are two examples to access PHY via the MII Management Controller.

For PHY Write if scan is not used:

1. Write 0 to MCMD
2. Write PHY address and register address to MADR
3. Write data to MWTD
4. Wait for busy bit to be cleared in MIND

For PHY Read if scan is not used:

1. Write 1 to MCMD
2. Write PHY address and register address to MADR
3. Wait for busy bit to be cleared in MIND
4. Write 0 to MCMD
5. Read data from MRDD

### 11.11.15 Station Address 0 Register (SA0 - 0x2008 4040)

The Station Address 0 register (SA0) has an address of 0x2008 4040. The bit definition of this register is shown in [Table 164](#).

**Table 164. Station Address register (SA0 - address 0x2008 4040) bit description**

Bit	Symbol	Function	Reset value
7:0	STATION ADDRESS, 2nd octet	This field holds the second octet of the station address.	0x0
15:8	STATION ADDRESS, 1st octet	This field holds the first octet of the station address.	0x0
31:16	-	Unused	0x0

The station address is used for perfect address filtering and for sending pause control frames. For the ordering of the octets in the packet please refer to [Figure 21](#).

### 11.11.16 Station Address 1 Register (SA1 - 0x2008 4044)

The Station Address 1 register (SA1) has an address of 0x2008 4044. The bit definition of this register is shown in [Table 165](#).

**Table 165. Station Address register (SA1 - address 0x2008 4044) bit description**

Bit	Symbol	Function	Reset value
7:0	STATION ADDRESS, 4th octet	This field holds the fourth octet of the station address.	0x0
15:8	STATION ADDRESS, 3rd octet	This field holds the third octet of the station address.	0x0
31:16	-	Unused	0x0

The station address is used for perfect address filtering and for sending pause control frames. For the ordering of the octets in the packet please refer to [Figure 21](#).

### 11.11.17 Station Address 2 Register (SA2 - 0x2008 4048)

The Station Address 2 register (SA2) has an address of 0x2008 4048. The bit definition of this register is shown in [Table 166](#).

**Table 166. Station Address register (SA2 - address 0x2008 4048) bit description**

Bit	Symbol	Function	Reset value
7:0	STATION ADDRESS, 6th octet	This field holds the sixth octet of the station address.	0x0
15:8	STATION ADDRESS, 5th octet	This field holds the fifth octet of the station address.	0x0
31:16	-	Unused	0x0

The station address is used for perfect address filtering and for sending pause control frames. For the ordering of the octets in the packet please refer to [Figure 21](#).

## 11.12 Control register definitions

### 11.12.1 Command Register (Command - 0x2008 4100)

The Command register (Command) register has an address of 0x2008 4100. Its bit definition is shown in [Table 167](#).

**Table 167. Command register (Command - address 0x2008 4100) bit description**

Bit	Symbol	Function	Reset value
0	RxEnable	Enable receive.	0
1	TxEnable	Enable transmit.	0
2	-	Unused	0x0
3	RegReset	When a “1” is written, all datapaths and the host registers are reset. The MAC needs to be reset separately.	0
4	TxReset	When a “1” is written, the transmit datapath is reset.	0
5	RxReset	When a “1” is written, the receive datapath is reset.	0
6	PassRuntFrame	When set to “1”, passes runt frames smaller than 64 bytes to memory unless they have a CRC error. If “0” runt frames are filtered out.	0
7	PassRXFilter	When set to “1”, disables receive filtering i.e. all frames received are written to memory.	0
8	TxFlowControl	Enable IEEE 802.3 / clause 31 flow control sending pause frames in full duplex and continuous preamble in half duplex.	0
9	RMII	When set to “1”, RMII mode is selected; if “0”, MII mode is selected.	0
10	FullDuplex	When set to “1”, indicates full duplex operation.	0
31:11	-	Unused	0x0

All bits can be written and read. The Tx/RxReset bits are write-only, reading will return a 0.

### 11.12.2 Status Register (Status - 0x2008 4104)

The Status register (Status) is a read-only register with an address of 0x2008 4104. Its bit definition is shown in [Table 168](#).

**Table 168. Status register (Status - address 0x2008 4104) bit description**

Bit	Symbol	Function	Reset value
0	RxStatus	If 1, the receive channel is active. If 0, the receive channel is inactive.	0
1	TxStatus	If 1, the transmit channel is active. If 0, the transmit channel is inactive.	0
31:2	-	Unused	0x0

The values represent the status of the two channels/data paths. When the status is 1, the channel is active, meaning:

- It is enabled and the Rx/TxEnable bit is set in the Command register or it just got disabled while still transmitting or receiving a frame.
- Also, for the transmit channel, the transmit queue is not empty i.e. ProduceIndex != ConsumeIndex.
- Also, for the receive channel, the receive queue is not full i.e. ProduceIndex != ConsumeIndex - 1.

The status transitions from active to inactive if the channel is disabled by a software reset of the Rx/TxEnable bit in the Command register and the channel has committed the status and data of the current frame to memory. The status also transitions to inactive if the transmit queue is empty or if the receive queue is full and status and data have been committed to memory.

### 11.12.3 Receive Descriptor Base Address Register (RxDescriptor - 0x2008 4108)

The Receive Descriptor base address register (RxDescriptor) has an address of 0x2008 4108. Its bit definition is shown in [Table 169](#).

**Table 169. Receive Descriptor Base Address register (RxDescriptor - address 0x2008 4108) bit description**

Bit	Symbol	Function	Reset value
1:0	-	Fixed to '00'	-
31:2	RxDescriptor	MSBs of receive descriptor base address.	0x0

The receive descriptor base address is a byte address aligned to a word boundary i.e. LSB 1:0 are fixed to "00". The register contains the lowest address in the array of descriptors.

### 11.12.4 Receive Status Base Address Register (RxStatus - 0x2008 410C)

The receive descriptor base address is a byte address aligned to a word boundary i.e. LSB 1:0 are fixed to "00". The register contains the lowest address in the array of descriptors.

**Table 170. Receive Status Base Address register (RxStatus - address 0x2008 410C) bit description**

Bit	Symbol	Function	Reset value
2:0	-	Fixed to '000'	-
31:3	RxStatus	MSBs of receive status base address.	0x0

The receive status base address is a byte address aligned to a double word boundary i.e. LSB 2:0 are fixed to "000".

### 11.12.5 Receive Number of Descriptors Register (RxDescriptor - 0x2008 4110)

The Receive Number of Descriptors register (RxDescriptorNumber) has an address of 0x2008 4110. Its bit definition is shown in [Table 171](#).

**Table 171. Receive Number of Descriptors register (RxDescriptor - address 0x2008 4110) bit description**

Bit	Symbol	Function	Reset value
15:0	RxDescriptorNumber	Number of descriptors in the descriptor array for which RxDescriptor is the base address. The number of descriptors is minus one encoded.	0x0
31:16	-	Unused	0x0

The receive number of descriptors register defines the number of descriptors in the descriptor array for which RxDescriptor is the base address. The number of descriptors should match the number of statuses. The register uses minus one encoding i.e. if the array has 8 elements, the value in the register should be 7.



### 11.12.6 Receive Produce Index Register (RxProduceIndex - 0x2008 4114)

The Receive Produce Index register (RxProduceIndex) is a read-only register with an address of 0x2008 4114. Its bit definition is shown in [Table 172](#).

**Table 172. Receive Produce Index register (RxProduceIndex - address 0x2008 4114) bit description**

Bit	Symbol	Function	Reset value
15:0	RxProduceIndex	Index of the descriptor that is going to be filled next by the receive datapath.	0x0
31:16	-	Unused	0x0

The receive produce index register defines the descriptor that is going to be filled next by the hardware receive process. After a frame has been received, hardware increments the index. The value is wrapped to 0 once the value of RxDescriptorNumber has been reached. If the RxProduceIndex equals RxConsumeIndex - 1, the array is full and any further frames being received will cause a buffer overrun error.

### 11.12.7 Receive Consume Index Register (RxConsumeIndex - 0x2008 4118)

The Receive consume index register (RxConsumeIndex) has an address of 0x2008 4118. Its bit definition is shown in [Table 173](#).

**Table 173. Receive Consume Index register (RxConsumeIndex - address 0x2008 4118) bit description**

Bit	Symbol	Function	Reset value
15:0	RxConsumeIndex	Index of the descriptor that is going to be processed next by the receive	
31:16	-	Unused	0x0

The receive consume register defines the descriptor that is going to be processed next by the software receive driver. The receive array is empty as long as RxProduceIndex equals RxConsumeIndex. As soon as the array is not empty, software can process the frame pointed to by RxConsumeIndex. After a frame has been processed by software, software should increment the RxConsumeIndex. The value must be wrapped to 0 once the value of RxDescriptorNumber has been reached. If the RxProduceIndex equals RxConsumeIndex - 1, the array is full and any further frames being received will cause a buffer overrun error.

### 11.12.8 Transmit Descriptor Base Address Register (TxDescriptor - 0x2008 411C)

The Transmit Descriptor base address register (TxDescriptor) has an address of 0x2008 411C. Its bit definition is shown in [Table 174](#).

**Table 174. Transmit Descriptor Base Address register (TxDescriptor - address 0x2008 411C) bit description**

Bit	Symbol	Function	Reset value
1:0	-	Fixed to '00'	-
31:2	TxDescriptor	MSBs of transmit descriptor base address.	0x0

The transmit descriptor base address is a byte address aligned to a word boundary i.e. LSB 1:0 are fixed to "00". The register contains the lowest address in the array of descriptors.

### 11.12.9 Transmit Status Base Address Register (TxStatus - 0x2008 4120)

The Transmit Status base address register (TxStatus) has an address of 0x2008 4120. Its bit definition is shown in [Table 175](#).

**Table 175. Transmit Status Base Address register (TxStatus - address 0x2008 4120) bit description**

Bit	Symbol	Function	Reset value
1:0	-	Fixed to '00'	-
31:2	TxStatus	MSBs of transmit status base address.	0x0

The transmit status base address is a byte address aligned to a word boundary i.e. LSB 1:0 are fixed to "00". The register contains the lowest address in the array of statuses.

### 11.12.10 Transmit Number of Descriptors Register (TxDescriptorNumber - 0x2008 4124)

The Transmit Number of Descriptors register (TxDescriptorNumber) has an address of 0x2008 4124. Its bit definition is shown in [Table 176](#).

**Table 176. Transmit Number of Descriptors register (TxDescriptorNumber - address 0x2008 4124) bit description**

Bit	Symbol	Function	Reset value
15:0	TxDescriptorNumber	Number of descriptors in the descriptor array for which TxDescriptor is the base address. The register is minus one encoded.	
31:16	-	Unused	0x0

The transmit number of descriptors register defines the number of descriptors in the descriptor array for which TxDescriptor is the base address. The number of descriptors should match the number of statuses. The register uses minus one encoding i.e. if the array has 8 elements, the value in the register should be 7.

### 11.12.11 Transmit Produce Index Register (TxProduceIndex - 0x2008 4128)

The Transmit Produce Index register (TxProduceIndex) has an address of 0x2008 4128. Its bit definition is shown in [Table 177](#).

**Table 177. Transmit Produce Index register (TxProduceIndex - address 0x2008 4128) bit description**

Bit	Symbol	Function	Reset value
15:0	TxProduceIndex	Index of the descriptor that is going to be filled next by the transmit software driver.	0x0
31:16	-	Unused	0x0

The transmit produce index register defines the descriptor that is going to be filled next by the software transmit driver. The transmit descriptor array is empty as long as TxProduceIndex equals TxConsumeIndex. If the transmit hardware is enabled, it will start transmitting frames as soon as the descriptor array is not empty. After a frame has been processed by software, it should increment the TxProduceIndex. The value must be wrapped to 0 once the value of TxDescriptorNumber has been reached. If the TxProduceIndex equals TxConsumeIndex - 1 the descriptor array is full and software should stop producing new descriptors until hardware has transmitted some frames and updated the TxConsumeIndex.

### 11.12.12 Transmit Consume Index Register (TxConsumeIndex - 0x2008 412C)

The Transmit Consume Index register (TxConsumeIndex) is a read-only register with an address of 0x2008 412C. Its bit definition is shown in [Table 178](#).

**Table 178. Transmit Consume Index register (TxConsumeIndex - address 0x2008 412C) bit description**

Bit	Symbol	Function	Reset value
15:0	TxConsumeIndex	Index of the descriptor that is going to be transmitted next by the transmit datapath.	0x0
31:16	-	Unused	0x0

The transmit consume index register defines the descriptor that is going to be transmitted next by the hardware transmit process. After a frame has been transmitted hardware increments the index, wrapping the value to 0 once the value of TxDescriptorNumber has been reached. If the TxConsumeIndex equals TxProduceIndex the descriptor array is empty and the transmit channel will stop transmitting until software produces new descriptors.

### 11.12.13 Transmit Status Vector 0 Register (TSV0 - 0x2008 4158)

The Transmit Status Vector 0 register (TSV0) is a read-only register with an address of 0x2008 4158. The transmit status vector registers store the most recent transmit status returned by the MAC. Since the status vector consists of more than 4 bytes, status is distributed over two registers TSV0 and TSV1. These registers are provided for debug purposes, because the communication between driver software and the Ethernet block takes place primarily through the frame descriptors. The status register contents are valid as long as the internal status of the MAC is valid and should typically only be read when the transmit and receive processes are halted.

[Table 179](#) lists the bit definitions of the TSV0 register.

**Table 179. Transmit Status Vector 0 register (TSV0 - address 0x2008 4158) bit description**

Bit	Symbol	Function	Reset value
0	CRC error	The attached CRC in the packet did not match the internally generated CRC.	0
1	Length check error	Indicates the frame length field does not match the actual number of data items and is not a type field.	0
2	Length out of range <sup>[1]</sup>	Indicates that frame type/length field was larger than 1500 bytes.	0
3	Done	Transmission of packet was completed.	0
4	Multicast	Packet's destination was a multicast address.	0
5	Broadcast	Packet's destination was a broadcast address.	0
6	Packet Defer	Packet was deferred for at least one attempt, but less than an excessive defer.	0
7	Excessive Defer	Packet was deferred in excess of 6071 nibble times in 100 Mbps or 24287 bit times in 10 Mbps mode.	0
8	Excessive Collision	Packet was aborted due to exceeding of maximum allowed number of collisions.	0
9	Late Collision	Collision occurred beyond collision window, 512 bit times.	0
10	Giant	Byte count in frame was greater than can be represented in the transmit byte count field in TSV1.	0
11	Underrun	Host side caused buffer underrun.	0

**Table 179. Transmit Status Vector 0 register (TSV0 - address 0x2008 4158) bit description**

Bit	Symbol	Function	Reset value
27:12	Total bytes	The total number of bytes transferred including collided attempts.	0x0
28	Control frame	The frame was a control frame.	0
29	Pause	The frame was a control frame with a valid PAUSE opcode.	0
30	Backpressure	Carrier-sense method backpressure was previously applied.	0
31	VLAN	Frame's length/type field contained 0x8100 which is the VLAN protocol identifier.	0

[1] The EMAC doesn't distinguish the frame type and frame length, so, e.g. when the IP(0x8000) or ARP(0x0806) packets are received, it compares the frame type with the max length and gives the "Length out of range" error. In fact, this bit is not an error indication, but simply a statement by the chip regarding the status of the received frame.

### 11.12.14 Transmit Status Vector 1 Register (TSV1 - 0x2008 415C)

The Transmit Status Vector 1 register (TSV1) is a read-only register with an address of 0x2008 415C. The transmit status vector registers store the most recent transmit status returned by the MAC. Since the status vector consists of more than 4 bytes, status is distributed over two registers TSV0 and TSV1. These registers are provided for debug purposes, because the communication between driver software and the Ethernet block takes place primarily through the frame descriptors. The status register contents are valid as long as the internal status of the MAC is valid and should typically only be read when the transmit and receive processes are halted. [Table 180](#) lists the bit definitions of the TSV1 register.

**Table 180. Transmit Status Vector 1 register (TSV1 - address 0x2008 415C) bit description**

Bit	Symbol	Function	Reset value
15:0	Transmit byte count	The total number of bytes in the frame, not counting the collided bytes.	0x0
19:16	Transmit collision count	Number of collisions the current packet incurred during transmission attempts. The maximum number of collisions (16) cannot be represented.	0x0
31:20	-	Unused	0x0

### 11.12.15 Receive Status Vector Register (RSV - 0x2008 4160)

The Receive status vector register (RSV) is a read-only register with an address of 0x2008 4160. The receive status vector register stores the most recent receive status returned by the MAC. This register is provided for debug purposes, because the communication between driver software and the Ethernet block takes place primarily through the frame descriptors. The status register contents are valid as long as the internal status of the MAC is valid and should typically only be read when the transmit and receive processes are halted.

[Table 181](#) lists the bit definitions of the RSV register.

**Table 181. Receive Status Vector register (RSV - address 0x2008 4160) bit description**

Bit	Symbol	Function	Reset value
15:0	Received byte count	Indicates length of received frame.	0x0
16	Packet previously ignored	Indicates that a packet was dropped.	0
17	RXDV event previously seen	Indicates that the last receive event seen was not long enough to be a valid packet.	0

**Table 181. Receive Status Vector register (RSV - address 0x2008 4160) bit description**

Bit	Symbol	Function	Reset value
18	Carrier event previously seen	Indicates that at some time since the last receive statistics, a carrier event was detected.	0
19	Receive code violation	Indicates that received PHY data does not represent a valid receive code.	0
20	CRC error	The attached CRC in the packet did not match the internally generated CRC.	0
21	Length check error	Indicates the frame length field does not match the actual number of data items and is not a type field.	0
22	Length out of range <sup>[1]</sup>	Indicates that frame type/length field was larger than 1518 bytes.	0
23	Receive OK	The packet had valid CRC and no symbol errors.	0
24	Multicast	The packet destination was a multicast address.	0
25	Broadcast	The packet destination was a broadcast address.	0
26	Dribble Nibble	Indicates that after the end of packet another 1-7 bits were received. A single nibble, called dribble nibble, is formed but not sent out.	0
27	Control frame	The frame was a control frame.	0
28	PAUSE	The frame was a control frame with a valid PAUSE opcode.	0
29	Unsupported Opcode	The current frame was recognized as a Control Frame but contains an unknown opcode.	0
30	VLAN	Frame's length/type field contained 0x8100 which is the VLAN protocol identifier.	0
31	-	Unused	0x0

[1] The EMAC doesn't distinguish the frame type and frame length, so, e.g. when the IP(0x8000) or ARP(0x0806) packets are received, it compares the frame type with the max length and gives the "Length out of range" error. In fact, this bit is not an error indication, but simply a statement by the chip regarding the status of the received frame.

### 11.12.16 Flow Control Counter Register (FlowControlCounter - 0x2008 4170)

The Flow Control Counter register (FlowControlCounter) has an address of 0x2008 4170. [Table 182](#) lists the bit definitions of the register.

**Table 182. Flow Control Counter register (FlowControlCounter - address 0x2008 4170) bit description**

Bit	Symbol	Function	Reset value
15:0	MirrorCounter	In full duplex mode the MirrorCounter specifies the number of cycles before re-issuing the Pause control frame.	0x0
31:16	PauseTimer	In full-duplex mode the PauseTimer specifies the value that is inserted into the pause timer field of a pause flow control frame. In half duplex mode the PauseTimer specifies the number of backpressure cycles.	0x0

### 11.12.17 Flow Control Status Register (FlowControlStatus - 0x2008 4174)

The Flow Control Status register (FlowControlStatus) is a read-only register with an address of 0x2008 4174. [Table 183](#) lists the bit definitions of the register.

**Table 183. Flow Control Status register (FlowControlStatus - address 0x2008 4174) bit description**

Bit	Symbol	Function	Reset value
15:0	MirrorCounterCurrent	In full duplex mode this register represents the current value of the datapath's mirror counter which counts up to the value specified by the MirrorCounter field in the FlowControlCounter register. In half duplex mode the register counts until it reaches the value of the PauseTimer bits in the FlowControlCounter register.	0x0
31:16	-	Unused	0x0

## 11.13 Receive filter register definitions

### 11.13.1 Receive Filter Control Register (RxFilterCtrl - 0x2008 4200)

The Receive Filter Control register (RxFilterCtrl) has an address of 0x2008 4200. [Table 184](#) lists the definition of the individual bits in the register.

**Table 184. Receive Filter Control register (RxFilterCtrl - address 0x2008 4200) bit description**

Bit	Symbol	Function	Reset value
0	AcceptUnicastEn	When set to '1', all unicast frames are accepted.	0
1	AcceptBroadcastEn	When set to '1', all broadcast frames are accepted.	0
2	AcceptMulticastEn	When set to '1', all multicast frames are accepted.	0
3	AcceptUnicastHashEn	When set to '1', unicast frames that pass the imperfect hash filter are accepted.	0
4	AcceptMulticastHashEn	When set to '1', multicast frames that pass the imperfect hash filter are accepted.	0
5	AcceptPerfectEn	When set to '1', the frames with a destination address identical to the station address are accepted.	0
11:6	-	Reserved. Read value is undefined, only zero should be written.	NA
12	MagicPacketEnWoL	When set to '1', the result of the magic packet filter will generate a WoL interrupt when there is a match.	0
13	RxFilterEnWoL	When set to '1', the result of the perfect address matching filter and the imperfect hash filter will generate a WoL interrupt when there is a match.	0
31:14	-	Unused	0x0

### 11.13.2 Receive Filter WoL Status Register (RxFilterWoLStatus - 0x2008 4204)

The Receive Filter Wake-up on LAN Status register (RxFilterWoLStatus) is a read-only register with an address of 0x2008 4204.

[Table 185](#) lists the definition of the individual bits in the register.

**Table 185. Receive Filter WoL Status register (RxFilterWoLStatus - address 0x2008 4204) bit description**

Bit	Symbol	Function	Reset value
0	AcceptUnicastWoL	When the value is '1', a unicast frames caused WoL.	0
1	AcceptBroadcastWoL	When the value is '1', a broadcast frame caused WoL.	0
2	AcceptMulticastWoL	When the value is '1', a multicast frame caused WoL.	0
3	AcceptUnicastHashWoL	When the value is '1', a unicast frame that passes the imperfect hash filter caused WoL.	0
4	AcceptMulticastHashWoL	When the value is '1', a multicast frame that passes the imperfect hash filter caused WoL.	0
5	AcceptPerfectWoL	When the value is '1', the perfect address matching filter caused WoL.	0
6	-	Unused	0x0
7	RxFilterWoL	When the value is '1', the receive filter caused WoL.	0
8	MagicPacketWoL	When the value is '1', the magic packet filter caused WoL.	0
31:9	-	Unused	0x0

The bits in this register record the cause for a WoL. Bits in RxFilterWoLStatus can be cleared by writing the RxFilterWoLClear register.

### 11.13.3 Receive Filter WoL Clear Register (RxFilterWoLClear - 0x2008 4208)

The Receive Filter Wake-up on LAN Clear register (RxFilterWoLClear) is a write-only register with an address of 0x2008 4208.

[Table 186](#) lists the definition of the individual bits in the register.

**Table 186. Receive Filter WoL Clear register (RxFilterWoLClear - address 0x2008 4208) bit description**

Bit	Symbol	Function	Reset value
0	AcceptUnicastWoLCIr	When a '1' is written to one of these bits (0 to 5), the corresponding status bit in the RxFilterWoLStatus register is cleared.	0
1	AcceptBroadcastWoLCIr		0
2	AcceptMulticastWoLCIr		0
3	AcceptUnicastHashWoLCIr		0
4	AcceptMulticastHashWoLCIr		0
5	AcceptPerfectWoLCIr		0
6	-	Unused	0x0
7	RxFilterWoLCIr	When a '1' is written to one of these bits (7 and/or 8), the corresponding status bit in the RxFilterWoLStatus register is cleared.	0
8	MagicPacketWoLCIr		0
31:9	-	Unused	0x0

The bits in this register are write-only; writing resets the corresponding bits in the RxFilterWoLStatus register.

### 11.13.4 Hash Filter Table LSBs Register (HashFilterL - 0x2008 4210)

The Hash Filter table LSBs register (HashFilterL) has an address of 0x2008 4210.

[Table 187](#) lists the bit definitions of the register. Details of Hash filter table use can be found in [Section 11.17.10 "Receive filtering" on page 248](#).

**Table 187. Hash Filter Table LSBs register (HashFilterL - address 0x2008 4210) bit description**

Bit	Symbol	Function	Reset value
31:0	HashFilterL	Bits 31:0 of the imperfect filter hash table for receive filtering.	0x0

### 11.13.5 Hash Filter Table MSBs Register (HashFilterH - 0x2008 4214)

The Hash Filter table MSBs register (HashFilterH) has an address of 0x2008 4214.

[Table 188](#) lists the bit definitions of the register. Details of Hash filter table use can be found in [Section 11.17.10 "Receive filtering" on page 248](#).

**Table 188. Hash Filter MSBs register (HashFilterH - address 0x2008 4214) bit description**

Bit	Symbol	Function	Reset value
31:0	HashFilterH	Bits 63:32 of the imperfect filter hash table for receive filtering.	0x0



## 11.14 Module control register definitions

### 11.14.1 Interrupt Status Register (IntStatus - 0x2008 4FE0)

The Interrupt Status register (IntStatus) is a read-only register with an address of 0x2008 4FE0. The interrupt status register bit definition is shown in [Table 189](#). Note that all bits are flip-flops with an asynchronous set in order to be able to generate interrupts if there are wake-up events while clocks are disabled.

**Table 189. Interrupt Status register (IntStatus - address 0x2008 4FE0) bit description**

Bit	Symbol	Function	Reset value
0	RxOverrunInt	Interrupt set on a fatal overrun error in the receive queue. The fatal interrupt should be resolved by a Rx soft-reset. The bit is not set when there is a nonfatal overrun error.	0
1	RxErrorInt	Interrupt trigger on receive errors: AlignmentError, RangeError, LengthError, SymbolError, CRCError or NoDescriptor or Overrun.	0
2	RxFinishedInt	Interrupt triggered when all receive descriptors have been processed i.e. on the transition to the situation where ProduceIndex == ConsumeIndex.	0
3	RxDoneInt	Interrupt triggered when a receive descriptor has been processed while the Interrupt bit in the Control field of the descriptor was set.	0
4	TxUnderrunInt	Interrupt set on a fatal underrun error in the transmit queue. The fatal interrupt should be resolved by a Tx soft-reset. The bit is not set when there is a nonfatal underrun error.	0
5	TxErrorInt	Interrupt trigger on transmit errors: LateCollision, ExcessiveCollision and ExcessiveDefer, NoDescriptor or Underrun.	0
6	TxFinishedInt	Interrupt triggered when all transmit descriptors have been processed i.e. on the transition to the situation where ProduceIndex == ConsumeIndex.	0
7	TxDoneInt	Interrupt triggered when a descriptor has been transmitted while the Interrupt bit in the Control field of the descriptor was set.	0
11:8	-	Unused	0x0
12	SoftInt	Interrupt triggered by software writing a 1 to the SoftIntSet bit in the IntSet register.	0
13	WakeUpInt	Interrupt triggered by a Wake-up event detected by the receive filter.	0
31:14	-	Unused	0x0

The interrupt status register is read-only. Setting can be done via the IntSet register. Reset can be accomplished via the IntClear register.

### 11.14.2 Interrupt Enable Register (IntEnable - 0x2008 4FE4)

The Interrupt Enable register (IntEnable) has an address of 0x2008 4FE4. The interrupt enable register bit definition is shown in [Table 190](#).

**Table 190. Interrupt Enable register (intEnable - address 0x2008 4FE4) bit description**

Bit	Symbol	Function	Reset value
0	RxOverrunIntEn	Enable for interrupt trigger on receive buffer overrun or descriptor underrun situations.	0
1	RxErrorIntEn	Enable for interrupt trigger on receive errors.	0
2	RxFinishedIntEn	Enable for interrupt triggered when all receive descriptors have been processed i.e. on the transition to the situation where ProduceIndex == ConsumeIndex.	0
3	RxDoneIntEn	Enable for interrupt triggered when a receive descriptor has been processed while the Interrupt bit in the Control field of the descriptor was set.	0

**Table 190. Interrupt Enable register (IntEnable - address 0x2008 4FE4) bit description**

Bit	Symbol	Function	Reset value
4	TxUnderrunIntEn	Enable for interrupt trigger on transmit buffer or descriptor underrun situations.	0
5	TxErrorIntEn	Enable for interrupt trigger on transmit errors.	0
6	TxFinishedIntEn	Enable for interrupt triggered when all transmit descriptors have been processed i.e. on the transition to the situation where ProduceIndex == ConsumeIndex.	0
7	TxDoneIntEn	Enable for interrupt triggered when a descriptor has been transmitted while the Interrupt bit in the Control field of the descriptor was set.	0
11:8	-	Unused	0x0
12	SoftIntEn	Enable for interrupt triggered by the SoftInt bit in the IntStatus register, caused by software writing a 1 to the SoftIntSet bit in the IntSet register.	0
13	WakeupIntEn	Enable for interrupt triggered by a Wake-up event detected by the receive filter.	0
31:14	-	Unused	0x0

### 11.14.3 Interrupt Clear Register (IntClear - 0x2008 4FE8)

The Interrupt Clear register (IntClear) is a write-only register with an address of 0x2008 4FE8. The interrupt clear register bit definition is shown in [Table 191](#).

**Table 191. Interrupt Clear register (IntClear - address 0x2008 4FE8) bit description**

Bit	Symbol	Function	Reset value
0	RxOverrunIntClr	Writing a '1' to one of these bits clears (0 to 7) the corresponding status bit in interrupt status register IntStatus.	0
1	RxErrorIntClr		0
2	RxFinishedIntClr		0
3	RxDoneIntClr		0
4	TxUnderrunIntClr		0
5	TxErrorIntClr		0
6	TxFinishedIntClr		0
7	TxDoneIntClr		0
11:8	-	Unused	0x0
12	SoftIntClr	Writing a '1' to one of these bits (12 and/or 13) clears the corresponding status bit in interrupt status register IntStatus.	0
13	WakeupIntClr		0
31:14	-	Unused	0x0

The interrupt clear register is write-only. Writing a 1 to a bit of the IntClear register clears the corresponding bit in the status register. Writing a 0 will not affect the interrupt status.

### 11.14.4 Interrupt Set Register (IntSet - 0x2008 4FEC)

The Interrupt Set register (IntSet) is a write-only register with an address of 0x2008 4FEC. The interrupt set register bit definition is shown in [Table 192](#).

**Table 192. Interrupt Set register (IntSet - address 0x2008 4FEC) bit description**

Bit	Symbol	Function	Reset value
0	RxOverrunIntSet	Writing a '1' to one of these bits (0 to 7) sets the corresponding status bit in interrupt status register IntStatus.	0
1	RxErrorIntSet		0
2	RxFinishedIntSet		0
3	RxDoneIntSet		0
4	TxUnderrunIntSet		0
5	TxErrorIntSet		0
6	TxFinishedIntSet		0
7	TxDoneIntSet		0
11:8	-	Unused	0x0
12	SoftIntSet	Writing a '1' to one of these bits (12 and/or 13) sets the corresponding status bit in interrupt status register IntStatus.	0
13	WakeUpIntSet		0
31:14	-	Unused	0x0

The interrupt set register is write-only. Writing a 1 to a bit of the IntSet register sets the corresponding bit in the status register. Writing a 0 will not affect the interrupt status.

### 11.14.5 Power-Down Register (PowerDown - 0x2008 4FF4)

The Power-Down register (PowerDown) is used to block all AHB accesses except accesses to the Power-Down register. The register has an address of 0x2008 4FF4. The bit definition of the register is listed in [Table 193](#).

**Table 193. Power-Down register (PowerDown - address 0x2008 4FF4) bit description**

Bit	Symbol	Function	Reset value
30:0	-	Unused	0x0
31	PowerDownMACAHB	If true, all AHB accesses will return a read/write error, except accesses to the Power-Down register.	0

Setting the bit will return an error on all read and write accesses on the MACAHB interface except for accesses to the Power-Down register.

## 11.15 Descriptor and status formats

This section defines the descriptor format for the transmit and receive scatter/gather DMA engines. Each Ethernet frame can consist of one or more fragments. Each fragment corresponds to a single descriptor. The DMA managers in the Ethernet block scatter (for receive) and gather (for transmit) multiple fragments for a single Ethernet frame.

### 11.15.1 Receive descriptors and statuses

Figure 22 depicts the layout of the receive descriptors in memory.

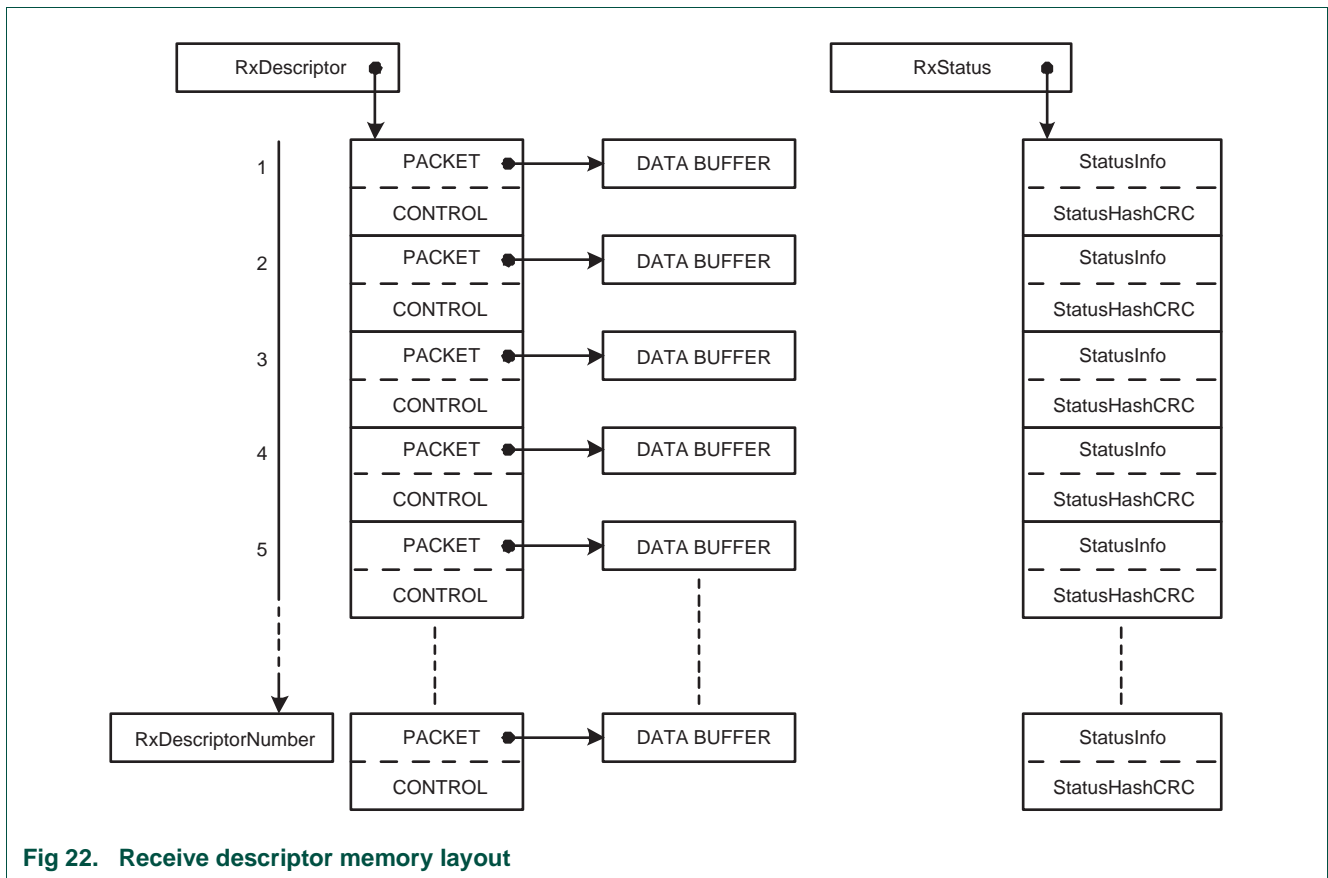


Fig 22. Receive descriptor memory layout

Receive descriptors are stored in an array in memory. The base address of the array is stored in the RxDescriptor register, and should be aligned on a 4 byte address boundary. The number of descriptors in the array is stored in the RxDescriptorNumber register using a minus one encoding style e.g. if the array has 8 elements the register value should be 7. Parallel to the descriptors there is an array of statuses. For each element of the descriptor array there is an associated status field in the status array. The base address of the status array is stored in the RxStatus register, and must be aligned on an 8 byte address boundary. During operation (when the receive data path is enabled) the RxDescriptor, RxStatus and RxDescriptorNumber registers should not be modified.

Two registers, RxConsumeIndex and RxProduceIndex, define the descriptor locations that will be used next by hardware and software. Both registers act as counters starting at 0 and wrapping when they reach the value of RxDescriptorNumber. The RxProduceIndex contains the index of the descriptor that is going to be filled with the next frame being

received. The RxConsumeIndex is programmed by software and is the index of the next descriptor that the software receive driver is going to process. When RxProduceIndex == RxConsumeIndex, the receive buffer is empty. When RxProduceIndex == RxConsumeIndex - 1 (taking wraparound into account), the receive buffer is full and newly received data would generate an overflow unless the software driver frees up one or more descriptors.

Each receive descriptor takes two word locations (8 bytes) in memory. Likewise each status field takes two words (8 bytes) in memory. Each receive descriptor consists of a pointer to the data buffer for storing receive data (Packet) and a control word (Control). The Packet field has a zero address offset, the control field has a 4 byte address offset with respect to the descriptor address as defined in [Table 194](#).

**Table 194. Receive Descriptor Fields**

Symbol	Address offset	Bytes	Description
Packet	0x0	4	Base address of the data buffer for storing receive data.
Control	0x4	4	Control information, see <a href="#">Table 195</a> .

The data buffer pointer (Packet) is a 32-bit, byte aligned address value containing the base address of the data buffer. The definition of the control word bits is listed in [Table 195](#).

**Table 195. Receive Descriptor Control Word**

Bit	Symbol	Description
10:0	Size	Size in bytes of the data buffer. This is the size of the buffer reserved by the device driver for a frame or frame fragment i.e. the byte size of the buffer pointed to by the Packet field. The size is -1 encoded e.g. if the buffer is 8 bytes the size field should be equal to 7.
30:11	-	Unused
31	Interrupt	If true generate an RxDone interrupt when the data in this frame or frame fragment and the associated status information has been committed to memory.

[Table 196](#) lists the fields in the receive status elements from the status array.

**Table 196. Receive Status Fields**

Symbol	Address offset	Bytes	Description
StatusInfo	0x0	4	Receive status return flags, see <a href="#">Table 198</a> .
StatusHashCRC	0x4	4	The concatenation of the destination address hash CRC and the source address hash CRC.

Each receive status consists of two words. The StatusHashCRC word contains a concatenation of the two 9-bit hash CRCs calculated from the destination and source addresses contained in the received frame. After detecting the destination and source addresses, StatusHashCRC is calculated once, then held for every fragment of the same frame.

The concatenation of the two CRCs is shown in [Table 197](#):

**Table 197. Receive Status HashCRC Word**

Bit	Symbol	Description
8:0	SAHashCRC	Hash CRC calculated from the source address.
15:9	-	Unused
24:16	DAHashCRC	Hash CRC calculated from the destination address.
31:25	-	Unused

The StatusInfo word contains flags returned by the MAC and flags generated by the receive data path reflecting the status of the reception. [Table 198](#) lists the bit definitions in the StatusInfo word.

**Table 198. Receive status information word**

Bit	Symbol	Description
10:0	RxSize	The size in bytes of the actual data transferred into one fragment buffer. In other words, this is the size of the frame or fragment as actually written by the DMA manager for one descriptor. This may be different from the Size bits of the Control field in the descriptor that indicate the size of the buffer allocated by the device driver. Size is -1 encoded e.g. if the buffer has 8 bytes the RxSize value will be 7.
17:11	-	Unused
18	ControlFrame	Indicates this is a control frame for flow control, either a pause frame or a frame with an unsupported opcode.
19	VLAN	Indicates a VLAN frame.
20	FailFilter	Indicates this frame has failed the Rx filter. These frames will not normally pass to memory. But due to the limitation of the size of the buffer, part of this frame may already be passed to memory. Once the frame is found to have failed the Rx filter, the remainder of the frame will be discarded without being passed to the memory. However, if the PassRxFilter bit in the Command register is set, the whole frame will be passed to memory.
21	Multicast	Set when a multicast frame is received.
22	Broadcast	Set when a broadcast frame is received.
23	CRCErr	The received frame had a CRC error.
24	SymbolError	The PHY reports a bit error over the PHY interface during reception.
25	LengthError	The frame length field value in the frame specifies a valid length, but does not match the actual data length.
26	RangeError <sup>[1]</sup>	The received packet exceeds the maximum packet size.
27	AlignmentError	An alignment error is flagged when dribble bits are detected and also a CRC error is detected. This is in accordance with IEEE std. 802.3/clause 4.3.2.
28	Overrun	Receive overrun. The adapter can not accept the data stream.
29	NoDescriptor	No new Rx descriptor is available and the frame is too long for the buffer size in the current receive descriptor.
30	LastFlag	When set to 1, indicates this descriptor is for the last fragment of a frame. If the frame consists of a single fragment, this bit is also set to 1.
31	Error	An error occurred during reception of this frame. This is a logical OR of AlignmentError, RangeError, LengthError, SymbolError, CRCErr, and Overrun.

[1] The EMAC doesn't distinguish the frame type and frame length, so, e.g. when the IP(0x8000) or ARP(0x0806) packets are received, it compares the frame type with the max length and gives the "Range" error. In fact, this bit is not an error indication, but simply a statement by the chip regarding the status of the received frame.

For multi-fragment frames, the value of the AlignmentError, RangeError, LengthError, SymbolError and CRCError bits in all but the last fragment in the frame will be 0; likewise the value of the FailFilter, Multicast, Broadcast, VLAN and ControlFrame bits is undefined. The status of the last fragment in the frame will copy the value for these bits from the MAC. All fragment statuses will have valid LastFrag, RxSize, Error, Overrun and NoDescriptor bits.

### 11.15.2 Transmit descriptors and statuses

Figure 23 depicts the layout of the transmit descriptors in memory.

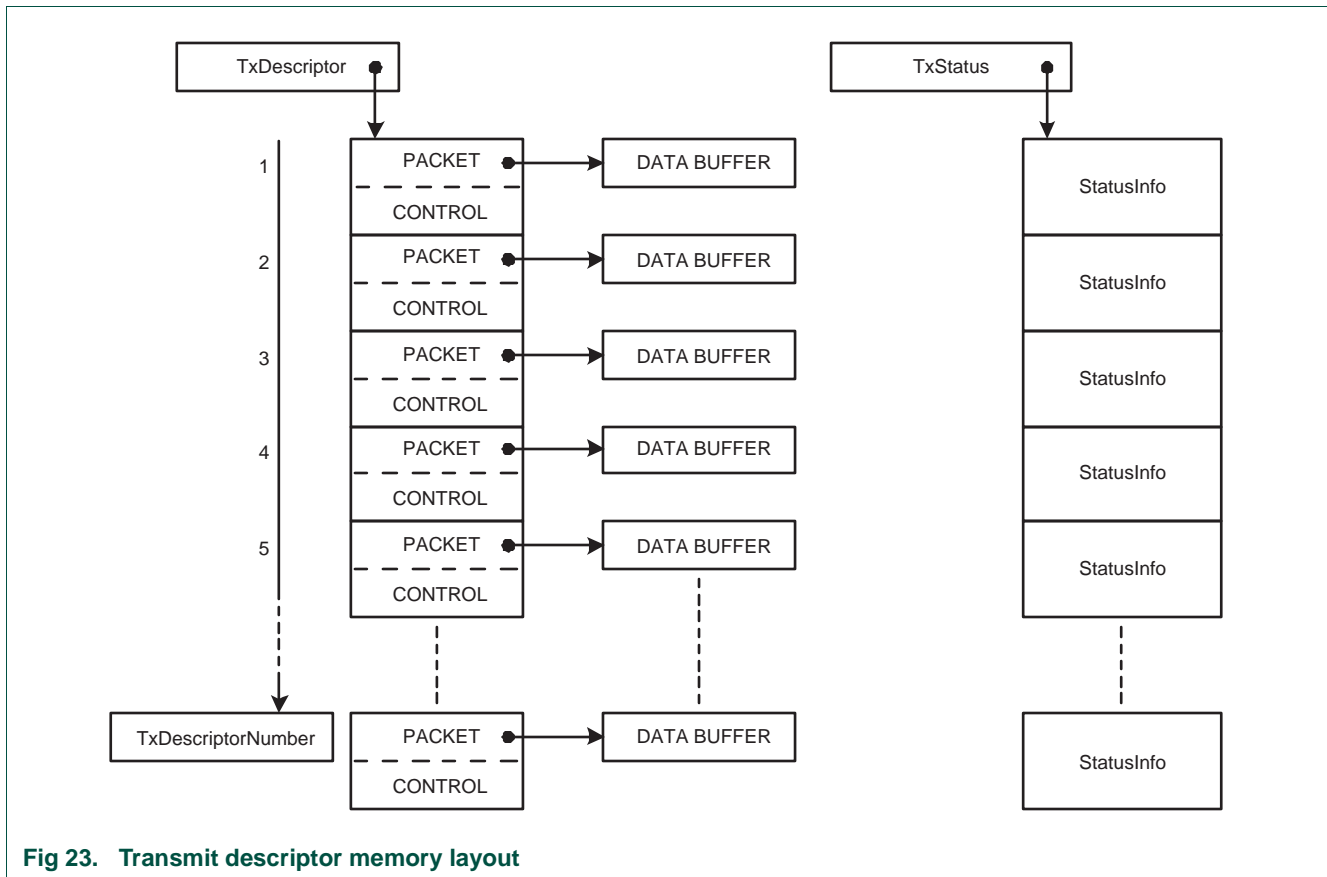


Fig 23. Transmit descriptor memory layout

Transmit descriptors are stored in an array in memory. The lowest address of the transmit descriptor array is stored in the TxDescriptor register, and must be aligned on a 4 byte address boundary. The number of descriptors in the array is stored in the TxDescriptorNumber register using a minus one encoding style i.e. if the array has 8 elements the register value should be 7. Parallel to the descriptors there is an array of statuses. For each element of the descriptor array there is an associated status field in the status array. The base address of the status array is stored in the TxStatus register, and must be aligned on a 4 byte address boundary. During operation (when the transmit data path is enabled) the TxDescriptor, TxStatus, and TxDescriptorNumber registers should not be modified.

Two registers, TxConsumeIndex and TxProduceIndex, define the descriptor locations that will be used next by hardware and software. Both register act as counters starting at 0 and wrapping when they reach the value of TxDescriptorNumber. The TxProduceIndex

contains the index of the next descriptor that is going to be filled by the software driver. The TxConsumeIndex contains the index of the next descriptor going to be transmitted by the hardware. When TxProduceIndex == TxConsumeIndex, the transmit buffer is empty. When TxProduceIndex == TxConsumeIndex - 1 (taking wraparound into account), the transmit buffer is full and the software driver cannot add new descriptors until the hardware has transmitted one or more frames to free up descriptors.

Each transmit descriptor takes two word locations (8 bytes) in memory. Likewise each status field takes one word (4 bytes) in memory. Each transmit descriptor consists of a pointer to the data buffer containing transmit data (Packet) and a control word (Control). The Packet field has a zero address offset, whereas the control field has a 4 byte address offset, see [Table 199](#).

**Table 199. Transmit descriptor fields**

Symbol	Address offset	Bytes	Description
Packet	0x0	4	Base address of the data buffer containing transmit data.
Control	0x4	4	Control information, see <a href="#">Table 200</a> .

The data buffer pointer (Packet) is a 32-bit, byte aligned address value containing the base address of the data buffer. The definition of the control word bits is listed in [Table 200](#).

**Table 200. Transmit descriptor control word**

Bit	Symbol	Description
10:0	Size	Size in bytes of the data buffer. This is the size of the frame or fragment as it needs to be fetched by the DMA manager. In most cases it will be equal to the byte size of the data buffer pointed to by the Packet field of the descriptor. Size is -1 encoded e.g. a buffer of 8 bytes is encoded as the Size value 7.
25:11	-	Unused
26	Override	Per frame override. If true, bits 30:27 will override the defaults from the MAC internal registers. If false, bits 30:27 will be ignored and the default values from the MAC will be used.
27	Huge	If true, enables huge frame, allowing unlimited frame sizes. When false, prevents transmission of more than the maximum frame length (MAXF[15:0]).
28	Pad	If true, pad short frames to 64 bytes.
29	CRC	If true, append a hardware CRC to the frame.
30	Last	If true, indicates that this is the descriptor for the last fragment in the transmit frame. If false, the fragment from the next descriptor should be appended.
31	Interrupt	If true, a TxDone interrupt will be generated when the data in this frame or frame fragment has been sent and the associated status information has been committed to memory.

[Table 201](#) shows the one field transmit status.

**Table 201. Transmit status fields**

Symbol	Address offset	Bytes	Description
StatusInfo	0x0	4	Transmit status return flags, see <a href="#">Table 202</a> .

The transmit status consists of one word which is the StatusInfo word. It contains flags returned by the MAC and flags generated by the transmit data path reflecting the status of the transmission. [Table 202](#) lists the bit definitions in the StatusInfo word.



**Table 202. Transmit status information word**

Bit	Symbol	Description
20:0	-	Unused
24:21	CollisionCount	The number of collisions this packet incurred, up to the Retransmission Maximum.
25	Defer	This packet incurred deferral, because the medium was occupied. This is not an error unless excessive deferral occurs.
26	ExcessiveDefer	This packet incurred deferral beyond the maximum deferral limit and was aborted.
27	ExcessiveCollision	Indicates this packet exceeded the maximum collision limit and was aborted.
28	LateCollision	An Out of window Collision was seen, causing packet abort.
29	Underrun	A Tx underrun occurred due to the adapter not producing transmit data.
30	NoDescriptor	The transmit stream was interrupted because a descriptor was not available.
31	Error	An error occurred during transmission. This is a logical OR of Underrun, LateCollision, ExcessiveCollision, and ExcessiveDefer.

For multi-fragment frames, the value of the LateCollision, ExcessiveCollision, ExcessiveDefer, Defer and CollisionCount bits in all but the last fragment in the frame will be 0. The status of the last fragment in the frame will copy the value for these bits from the MAC. All fragment statuses will have valid Error, NoDescriptor and Underrun bits.

## 11.16 Ethernet block functional description

This section defines the functions of the DMA capable 10/100 Ethernet MAC. After introducing the DMA concepts of the Ethernet block, and a description of the basic transmit and receive functions, this section elaborates on advanced features such as flow control, receive filtering, etc.

### 11.16.1 Overview

The Ethernet block can transmit and receive Ethernet packets from an off-chip Ethernet PHY connected through the MII/RMII interface. MII or RMII mode is selected by software.

Typically during system start-up, the Ethernet block will be initialized. Software initialization of the Ethernet block should include initialization of the descriptor and status arrays as well as the receiver fragment buffers.

**Remark:** when initializing the Ethernet block, it is important to first configure the PHY and insure that reference clocks (ENET\_REF\_CLK signal in RMII mode, or both ENET\_RX\_CLK and ENET\_TX\_CLK signals in MII mode) are present at the external pins and connected to the EMAC module (selecting the appropriate pins using the IOCON registers) prior to continuing with Ethernet configuration. Otherwise the CPU can become locked and no further functionality will be possible. This will cause JTAG lose communication with the target, if debug mode is being used.

To transmit a packet the software driver has to set up the appropriate Control registers and a descriptor to point to the packet data buffer before transferring the packet to hardware by incrementing the TxProduceIndex register. After transmission, hardware will increment TxConsumeIndex and optionally generate an interrupt.

The hardware will receive packets from the PHY and apply filtering as configured by the software driver. While receiving a packet the hardware will read a descriptor from memory to find the location of the associated receiver data buffer. Receive data is written in the data buffer and receive status is returned in the receive descriptor status word. Optionally an interrupt can be generated to notify software that a packet has been received. Note that the DMA manager will prefetch and buffer up to three descriptors.

### 11.16.2 AHB interface

The registers of the Ethernet block connect to an AHB slave interface to allow access to the registers from the CPU.

The AHB interface has a 32-bit data path, which supports only word accesses and has an address aperture of 4 kB. [Table 147](#) lists the registers of the Ethernet block.

All AHB write accesses to registers are posted except for accesses to the IntSet, IntClear and IntEnable registers. AHB write operations are executed in order.

If the PowerDown bit of the PowerDown register is set, all AHB read and write accesses will return a read or write error except for accesses to the PowerDown register.

#### Bus Errors

The Ethernet block generates errors for several conditions:

- The AHB interface will return a read error when there is an AHB read access to a write-only register; likewise a write error is returned when there is an AHB write access to the read-only register. An AHB read or write error will be returned on AHB read or write accesses to reserved registers. These errors are propagated back to the CPU. Registers defined as read-only and write-only are identified in [Table 147](#).
- If the PowerDown bit is set all accesses to AHB registers will result in an error response except for accesses to the PowerDown register.

## 11.17 Interrupts

The Ethernet block has a single interrupt request output to the CPU (via the NVIC).

The interrupt service routine must read the IntStatus register to determine the origin of the interrupt. All interrupt statuses can be set by software writing to the IntSet register; statuses can be cleared by software writing to the IntClear register.

The transmit and receive data paths can only set interrupt statuses, they cannot clear statuses. The SoftInt interrupt cannot be set by hardware and can be used by software for test purposes.

### 11.17.1 Direct Memory Access (DMA)

#### Descriptor arrays

The Ethernet block includes two DMA managers. The DMA managers make it possible to transfer frames directly to and from memory with little support from the processor and without the need to trigger an interrupt for each frame.

The DMA managers work with arrays of frame descriptors and statuses that are stored in memory. The descriptors and statuses act as an interface between the Ethernet hardware and the device driver software. There is one descriptor array for receive frames and one descriptor array for transmit frames. Using buffering for frame descriptors, the memory traffic and memory bandwidth utilization of descriptors can be kept small.

Each frame descriptor contains two 32-bit fields: the first field is a pointer to a data buffer containing a frame or a fragment, whereas the second field is a control word related to that frame or fragment.

The software driver must write the base addresses of the descriptor and status arrays in the TxDescriptor/RxDescriptor and TxStatus/RxStatus registers. The number of descriptors/statuses in each array must be written in the TxDescriptorNumber/RxDescriptorNumber registers. The number of descriptors in an array corresponds to the number of statuses in the associated status array.

Transmit descriptor arrays, receive descriptor arrays and transmit status arrays must be aligned on a 4 byte (32bit) address boundary, while the receive status array must be aligned on a 8 byte (64bit) address boundary.

#### Ownership of descriptors

Both device driver software and Ethernet hardware can read and write the descriptor arrays at the same time in order to produce and consume descriptors. A descriptor is "owned" either by the device driver or by the Ethernet hardware. Only the owner of a descriptor reads or writes its value. Typically, the sequence of use and ownership of descriptors and statuses is as follows: a descriptor is owned and set up by the device driver; ownership of the descriptor/status is passed by the device driver to the Ethernet block, which reads the descriptor and writes information to the status field; the Ethernet block passes ownership of the descriptor back to the device driver, which uses the status information and then recycles the descriptor to be used for another frame. Software must pre-allocate the memory used to hold the descriptor arrays.

Software can hand over ownership of descriptors and statuses to the hardware by incrementing (and wrapping if on the array boundary) the TxProduceIndex/RxConsumeIndex registers. Hardware hands over descriptors and status to software by updating the TxConsumeIndex/ RxProduceIndex registers.

After handing over a descriptor to the receive and transmit DMA hardware, device driver software should not modify the descriptor or reclaim the descriptor by decrementing the TxProduceIndex/ RxConsumeIndex registers because descriptors may have been prefetched by the hardware. In this case the device driver software will have to wait until the frame has been transmitted or the device driver has to soft-reset the transmit and/or receive data paths which will also reset the descriptor arrays.

### Sequential order with wrap-around

When descriptors are read from and statuses are written to the arrays, this is done in sequential order with wrap-around. Sequential order means that when the Ethernet block has finished reading/writing a descriptor/status, the next descriptor/status it reads/writes is the one at the next higher, adjacent memory address. Wrap around means that when the Ethernet block has finished reading/writing the last descriptor/status of the array (with the highest memory address), the next descriptor/status it reads/writes is the first descriptor/status of the array at the base address of the array.

### Full and Empty state of descriptor arrays

The descriptor arrays can be empty, partially full or full. A descriptor array is empty when all descriptors are owned by the producer. A descriptor array is partially full if both producer and consumer own part of the descriptors and both are busy processing those descriptors. A descriptor array is full when all descriptors (except one) are owned by the consumer, so that the producer has no more room to process frames. Ownership of descriptors is indicated with the use of a consume index and a produce index. The produce index is the first element of the array owned by the producer. It is also the index of the array element that is next going to be used by the producer of frames (it may already be busy using it and subsequent elements). The consume index is the first element of the array that is owned by the consumer. It is also the number of the array element next to be consumed by the consumer of frames (it and subsequent elements may already be in the process of being consumed). If the consume index and the produce index are equal, the descriptor array is empty and all array elements are owned by the producer. If the consume index equals the produce index plus one, then the array is full and all array elements (except the one at the produce index) are owned by the consumer. With a full descriptor array, still one array element is kept empty, to be able to easily distinguish the full or empty state by looking at the value of the produce index and consume index. An array must have at least 2 elements to be able to indicate a full descriptor array with a produce index of value 0 and a consume index of value 1. The wrap around of the arrays is taken into account when determining if a descriptor array is full, so a produce index that indicates the last element in the array and a consume index that indicates the first element in the array, also means the descriptor array is full. When the produce index and the consume index are unequal and the consume index is not the produce index plus one (with wrap around taken into account), then the descriptor array is partially full and both the consumer and producer own enough descriptors to be able to operate actively on the descriptor array.

### Interrupt bit

The descriptors have an Interrupt bit, which is programmed by software. When the Ethernet block is processing a descriptor and finds this bit set, it will allow triggering an interrupt (after committing status to memory) by passing the RxDoneInt or TxDoneInt bits in the IntStatus register to the interrupt output pin. If the Interrupt bit is not set in the descriptor, then the RxDoneInt or TxDoneInt are not set and no interrupt is triggered (note that the corresponding bits in IntEnable must also be set to trigger interrupts). This offers flexible ways of managing the descriptor arrays. For instance, the device driver could add 10 frames to the Tx descriptor array, and set the Interrupt bit in descriptor number 5 in the descriptor array. This would invoke the interrupt service routine before the transmit descriptor array is completely exhausted. The device driver could add another batch of frames to the descriptor array, without interrupting continuous transmission of frames.

### Frame fragments

For maximum flexibility in frame storage, frames can be split up into multiple frame fragments with fragments located in different places in memory. In this case one descriptor is used for each frame fragment. So, a descriptor can point to a single frame or to a fragment of a frame. By using fragments, scatter/gather DMA can be done: transmit frames are gathered from multiple fragments in memory and receive frames can be scattered to multiple fragments in memory.

By stringing together fragments it is possible to create large frames from small memory areas. Another use of fragments is to be able to locate a frame header and frame payload in different places and to concatenate them without copy operations in the device driver.

For transmissions, the Last bit in the descriptor Control field indicates if the fragment is the last in a frame; for receive frames, the LastFrag bit in the StatusInfo field of the status words indicates if the fragment is the last in the frame. If the Last(Frag) bit is 0 the next descriptor belongs to the same Ethernet frame, If the Last(Frag) bit is 1 the next descriptor is a new Ethernet frame.

### 11.17.2 Initialization

After reset, the Ethernet software driver needs to initialize the Ethernet block. During initialization the software needs to:

- Remove the soft reset condition from the MAC.
- Configure the PHY via the MIIM interface of the MAC.

**Remark:** it is important to configure the PHY and insure that reference clocks (ENET\_REF\_CLK signal in RMII mode, or both ENET\_RX\_CLK and ENET\_TX\_CLK signals in MII mode) are present at the external pins and connected to the EMAC module (selecting the appropriate pins using the IOCON registers) prior to continuing with Ethernet configuration. Otherwise the CPU can become locked and no further functionality will be possible. This will cause JTAG lose communication with the target, if debug mode is being used.

- Select MII or RMII mode
- Configure the transmit and receive DMA engines, including the descriptor arrays.
- Configure the host registers (MAC1,MAC2 etc.) in the MAC.
- Enable the receive and transmit data paths.

Depending on the PHY, the software needs to initialize registers in the PHY via the MII Management interface. The software can read and write PHY registers by programming the MCFG, MCMD, MADR registers of the MAC. Write data should be written to the MWTD register; read data and status information can be read from the MRDD and MIND registers.

The Ethernet block supports MII and RMII PHYs. During initialization software must select MII or RMII mode by programming the Command register.

Before switching to RMII mode the default soft reset (MAC1 register bit 15) has to be de-asserted. The clock(s) from the PHY must be running and internally connected during this operation.

Transmit and receive DMA engines should be initialized by the device driver by allocating the descriptor and status arrays in memory. Transmit and receive functions have their own dedicated descriptor and status arrays. The base addresses of these arrays need to be programmed in the TxDescriptor/TxStatus and RxDescriptor/RxStatus registers. The number of descriptors in an array matches the number of statuses in an array.

Please note that the transmit descriptors, receive descriptors and receive statuses are 8 bytes each while the transmit statuses are 4 bytes each. All descriptor arrays and transmit statuses need to be aligned on 4 byte boundaries; receive status arrays need to be aligned on 8 byte boundaries. The number of descriptors in the descriptor arrays needs to be written to the TxDescriptorNumber/RxDescriptorNumber registers using a -1 encoding i.e. the value in the registers is the number of descriptors minus one e.g. if the descriptor array has 4 descriptors the value of the number of descriptors register should be 3.

After setting up the descriptor arrays, frame buffers need to be allocated for the receive descriptors before enabling the receive data path. The Packet field of the receive descriptors needs to be filled with the base address of the frame buffer of that descriptor. Amongst others the Control field in the receive descriptor needs to contain the size of the data buffer using -1 encoding.

The receive data path has a configurable filtering function for discarding/ignoring specific Ethernet frames. The filtering function should also be configured during initialization.

After an assertion of the hardware reset, the soft reset bit in the MAC will be asserted. The soft reset condition must be removed before the Ethernet block can be enabled.

Enabling of the receive function is located in two places. The receive DMA manager needs to be enabled and the receive data path of the MAC needs to be enabled. To prevent overflow in the receive DMA engine the receive DMA engine should be enabled by setting the RxEnable bit in the Command register before enabling the receive data path in the MAC by setting the RECEIVE ENABLE bit in the MAC1 register.

The transmit DMA engine can be enabled at any time by setting the TxEnable bit in the Command register.

Before enabling the data paths, several options can be programmed in the MAC, such as automatic flow control, transmit to receive loop-back for verification, full/half duplex modes, etc.

Base addresses of descriptor arrays and descriptor array sizes cannot be modified without a (soft) reset of the receive and transmit data paths.

### 11.17.3 Transmit process

#### Overview

This section outlines the transmission process.

#### Device driver sets up descriptors and data

If the descriptor array is full the device driver should wait for the descriptor arrays to become not full before writing to a descriptor in the descriptor array. If the descriptor array is not full, the device driver should use the descriptor numbered TxProduceIndex of the array pointed to by TxDescriptor.

The Packet pointer in the descriptor is set to point to a data frame or frame fragment to be transmitted. The Size field in the Command field of the descriptor should be set to the number of bytes in the fragment buffer, -1 encoded. Additional control information can be indicated in the Control field in the descriptor (bits Interrupt, Last, CRC, Pad).

After writing the descriptor the descriptor needs to be handed over to the hardware by incrementing (and possibly wrapping) the TxProduceIndex register.

If the transmit data path is disabled, the device driver should not forget to enable the transmit data path by setting the TxEnable bit in the Command register.

When there is a multi-fragment transmission for fragments other than the last, the Last bit in the descriptor must be set to 0; for the last fragment the Last bit must be set to 1. To trigger an interrupt when the frame has been transmitted and transmission status has been committed to memory, set the Interrupt bit in the descriptor Control field to 1. To have the hardware add a CRC in the frame sequence control field of this Ethernet frame, set the CRC bit in the descriptor. This should be done if the CRC has not already been added by software. To enable automatic padding of small frames to the minimum required frame size, set the Pad bit in the Control field of the descriptor to 1. In typical applications bits CRC and Pad are both set to 1.

The device driver can set up interrupts using the IntEnable register to wait for a signal of completion from the hardware or can periodically inspect (poll) the progress of transmission. It can also add new frames at the end of the descriptor array, while hardware consumes descriptors at the start of the array.

The device driver can stop the transmit process by resetting the TxEnable bit in the Command register to 0. The transmission will not stop immediately; frames already being transmitted will be transmitted completely and the status will be committed to memory before deactivating the data path. The status of the transmit data path can be monitored by the device driver reading the TxStatus bit in the Status register.

As soon as the transmit data path is enabled and the corresponding TxConsumeIndex and TxProduceIndex are not equal i.e. the hardware still needs to process frames from the descriptor array, the TxStatus bit in the Status register will return to 1 (active).

#### Tx DMA manager reads the Tx descriptor array



When the TxEnable bit is set, the Tx DMA manager reads the descriptors from memory at the address determined by TxDescriptor and TxConsumeIndex. The number of descriptors requested is determined by the total number of descriptors owned by the hardware: TxProduceIndex - TxConsumeIndex. Block transferring descriptors minimizes memory loading. Read data returned from memory is buffered and consumed as needed.

### Tx DMA manager transmits data

After reading the descriptor the transmit DMA engine reads the associated frame data from memory and transmits the frame. After transfer completion, the Tx DMA manager writes status information back to the StatusInfo and StatusHashCRC words of the status field. The value of the TxConsumeIndex is only updated after status information has been committed to memory, which is checked by an internal tag protocol in the memory interface. The Tx DMA manager continues to transmit frames until the descriptor array is empty. If the transmit descriptor array is empty the TxStatus bit in the Status register will return to 0 (inactive). If the descriptor array is empty the Ethernet hardware will set the TxFinishedInt bit of the IntStatus register. The transmit data path will still be enabled.

The Tx DMA manager inspects the Last bit of the descriptor Control field when loading the descriptor. If the Last bit is 0, this indicates that the frame consists of multiple fragments. The Tx DMA manager gathers all the fragments from the host memory, visiting a string of frame descriptors, and sends them out as one Ethernet frame on the Ethernet connection. When the Tx DMA manager finds a descriptor with the Last bit in the Control field set to 1, this indicates the last fragment of the frame and thus the end of the frame is found.

### Update ConsumeIndex

Each time the Tx DMA manager commits a status word to memory it completes the transmission of a descriptor and it increments the TxConsumeIndex (taking wrap around into account) to hand the descriptor back to the device driver software. Software can re-use the descriptor for new transmissions after hardware has handed it back.

The device driver software can keep track of the progress of the DMA manager by reading the TxConsumeIndex register to see how far along the transmit process is. When the Tx descriptor array is emptied completely, the TxConsumeIndex register retains its last value.

### Write transmission status

After the frame has been transmitted over the MII/RMII bus, the StatusInfo word of the frame descriptor is updated by the DMA manager.

If the descriptor is for the last fragment of a frame (or for the whole frame if there are no fragments), then depending on the success or failure of the frame transmission, error flags (Error, LateCollision, ExcessiveCollision, Underrun, ExcessiveDefer, Defer) are set in the status. The CollisionCount field is set to the number of collisions the frame incurred, up to the Retransmission Maximum programmed in the Collision window/retry register of the MAC.

Statuses for all but the last fragment in the frame will be written as soon as the data in the frame has been accepted by the Tx DMA manager. Even if the descriptor is for a frame fragment other than the last fragment, the error flags are returned via the AHB interface. If the Ethernet block detects a transmission error during transmission of a (multi-fragment) frame, all remaining fragments of the frame are still read via the AHB interface. After an error, the remaining transmit data is discarded by the Ethernet block. If there are errors

during transmission of a multi-fragment frame the error statuses will be repeated until the last fragment of the frame. Statuses for all but the last fragment in the frame will be written as soon as the data in the frame has been accepted by the Tx DMA manager. These may include error information if the error is detected early enough. The status for the last fragment in the frame will only be written after the transmission has completed on the Ethernet connection. Thus, the status for the last fragment will always reflect any error that occurred anywhere in the frame.

The status of the last frame transmission can also be inspected by reading the TSV0 and TSV1 registers. These registers do not report statuses on a fragment basis and do not store information of previously sent frames. They are provided primarily for debug purposes, because the communication between driver software and the Ethernet block takes place through the frame descriptors. The status registers are valid as long as the internal status of the MAC is valid and should typically only be read when the transmit and receive processes are halted.

### Transmission error handling

If an error occurs during the transmit process, the Tx DMA manager will report the error via the transmission StatusInfo word written in the Status array and the IntStatus interrupt status register.

The transmission can generate several types of errors: LateCollision, ExcessiveCollision, ExcessiveDefer, Underrun, and NoDescriptor. All have corresponding bits in the transmission StatusInfo word. In addition to the separate bits in the StatusInfo word, LateCollision, ExcessiveCollision, and ExcessiveDefer are ORed together into the Error bit of the Status. Errors are also propagated to the IntStatus register; the TxError bit in the IntStatus register is set in the case of a LateCollision, ExcessiveCollision, ExcessiveDefer, or NoDescriptor error; Underrun errors are reported in the TxUnderrun bit of the IntStatus register.

Underrun errors can have three causes:

- The next fragment in a multi-fragment transmission is not available. This is a nonfatal error. A NoDescriptor status will be returned on the previous fragment and the TxError bit in IntStatus will be set.
- The transmission fragment data is not available when the Ethernet block has already started sending the frame. This is a nonfatal error. An Underrun status will be returned on transfer and the TxError bit in IntStatus will be set.
- The flow of transmission statuses stalls and a new status has to be written while a previous status still waits to be transferred across the memory interface. This is a fatal error which can only be resolved by a soft reset of the hardware.

The first and second situations are nonfatal and the device driver has to re-send the frame or have upper software layers re-send the frame. In the third case the hardware is in an undefined state and needs to be soft reset by setting the TxReset bit in the Command register.

After reporting a LateCollision, ExcessiveCollision, ExcessiveDefer or Underrun error, the transmission of the erroneous frame will be aborted, remaining transmission data and frame fragments will be discarded and transmission will continue with the next frame in the descriptor array.

Device drivers should catch the transmission errors and take action.

### Transmit triggers interrupts

The transmit data path can generate four different interrupt types:

- If the Interrupt bit in the descriptor Control field is set, the Tx DMA will set the TxDoneInt bit in the IntStatus register after sending the fragment and committing the associated transmission status to memory. Even if a descriptor (fragment) is not the last in a multi-fragment frame the Interrupt bit in the descriptor can be used to generate an interrupt.
- If the descriptor array is empty while the Ethernet hardware is enabled the hardware will set the TxFinishedInt bit of the IntStatus register.
- If the AHB interface does not consume the transmission statuses at a sufficiently high bandwidth the transmission may underrun in which case the TxUnderrun bit will be set in the IntStatus register. This is a fatal error which requires a soft reset of the transmission queue.
- In the case of a transmission error (LateCollision, ExcessiveCollision, or ExcessiveDefer) or a multi-fragment frame where the device driver did provide the initial fragments but did not provide the rest of the fragments (NoDescriptor) or in the case of a nonfatal overrun, the hardware will set the TxErrorInt bit of the IntStatus register.

All of the above interrupts can be enabled and disabled by setting or resetting the corresponding bits in the IntEnable register. Enabling or disabling does not affect the IntStatus register contents, only the propagation of the interrupt status to the CPU (via the NVIC).

The interrupts, either of individual frames or of the whole list, are a good means of communication between the DMA manager and the device driver, triggering the device driver to inspect the status words of descriptors that have been processed.

### Transmit example

[Figure 24](#) illustrates the transmit process in an example transmitting uses a frame header of 8 bytes and a frame payload of 12 bytes.

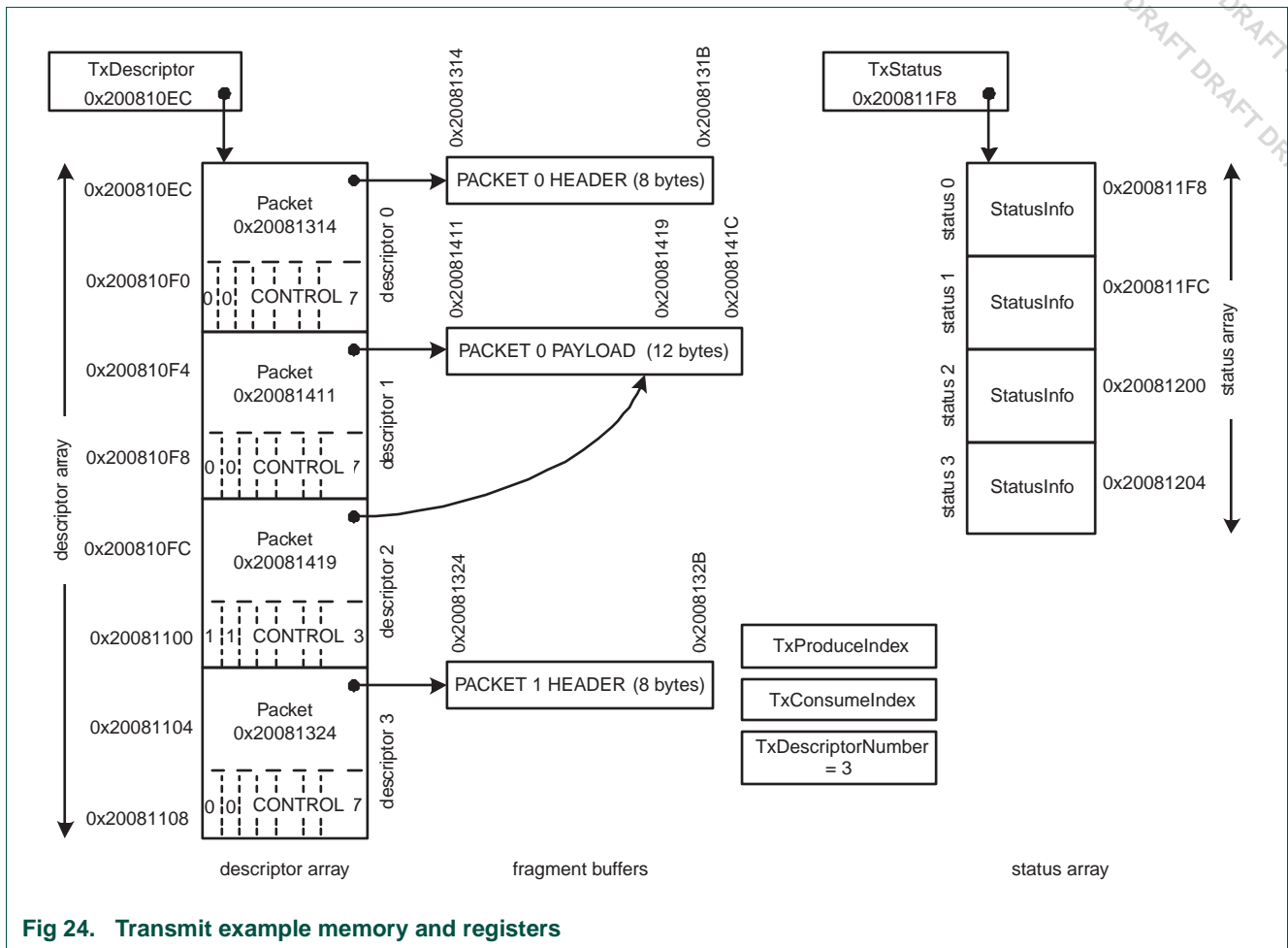


Fig 24. Transmit example memory and registers

After reset the values of the DMA registers will be zero. During initialization the device driver will allocate the descriptor and status array in memory. In this example, an array of four descriptors is allocated; the array is 4x2x4 bytes and aligned on a 4 byte address boundary. Since the number of descriptors matches the number of statuses the status array consists of four elements; the array is 4x1x4 bytes and aligned on a 4 byte address boundary. The device driver writes the base address of the descriptor array (0x2008 10EC) to the TxDescriptor register and the base address of the status array (0x2008 11F8) to the TxStatus register. The device driver writes the number of descriptors and statuses minus 1(3) to the TxDescriptorNumber register. The descriptors and statuses in the arrays need not be initialized, yet.

At this point, the transmit data path may be enabled by setting the TxEnable bit in the Command register. If the transmit data path is enabled while there are no further frames to send the TxFinishedInt interrupt flag will be set. To reduce the processor interrupt load only the desired interrupts can be enabled by setting the relevant bits in the IntEnable register.

Now suppose application software wants to transmit a frame of 12 bytes using a TCP/IP protocol (in real applications frames will be larger than 12 bytes). The TCP/IP stack will add a header to the frame. The frame header need not be immediately in front of the payload data in memory. The device driver can program the Tx DMA to collect header and payload data. To do so, the device driver will program the first descriptor to point at the

frame header; the Last flag in the descriptor will be set to false/0 to indicate a multi-fragment transmission. The device driver will program the next descriptor to point at the actual payload data. The maximum size of a payload buffer is 2 kB so a single descriptor suffices to describe the payload buffer. For the sake of the example though the payload is distributed across two descriptors. After the first descriptor in the array describing the header, the second descriptor in the array describes the initial 8 bytes of the payload; the third descriptor in the array describes the remaining 4 bytes of the frame. In the third descriptor the Last bit in the Control word is set to true/1 to indicate it is the last descriptor in the frame. In this example the Interrupt bit in the descriptor Control field is set in the last fragment of the frame in order to trigger an interrupt after the transmission completed. The Size field in the descriptor's Control word is set to the number of bytes in the fragment buffer, -1 encoded.

Note that in real device drivers, the payload will typically only be split across multiple descriptors if it is more than 2 kB. Also note that transmission payload data is forwarded to the hardware without the device driver copying it (zero copy device driver).

After setting up the descriptors for the transaction the device driver increments the TxProduceIndex register by 3 since three descriptors have been programmed. If the transmit data path was not enabled during initialization the device driver needs to enable the data path now.

If the transmit data path is enabled the Ethernet block will start transmitting the frame as soon as it detects the TxProduceIndex is not equal to TxConsumeIndex - both were zero after reset. The Tx DMA will start reading the descriptors from memory. The memory system will return the descriptors and the Ethernet block will accept them one by one while reading the transmit data fragments.

As soon as transmission read data is returned from memory, the Ethernet block will try to start transmission on the Ethernet connection via the MII/RMII interface.

After transmitting each fragment of the frame the Tx DMA will write the status of the fragment's transmission. Statuses for all but the last fragment in the frame will be written as soon as the data in the frame has been accepted by the Tx DMA manager. The status for the last fragment in the frame will only be written after the transmission has completed on the Ethernet connection.

Since the Interrupt bit in the descriptor of the last fragment is set, after committing the status of the last fragment to memory the Ethernet block will trigger a TxDoneInt interrupt, which triggers the device driver to inspect the status information.

In this example the device driver cannot add new descriptors as long as the Ethernet block has not incremented the TxConsumeIndex because the descriptor array is full (even though one descriptor is not programmed yet). Only after the hardware commits the status for the first fragment to memory and the TxConsumeIndex is set to 1 by the DMA manager can the device driver program the next (the fourth) descriptor. The fourth descriptor can already be programmed before completely transmitting the first frame.

In this example the hardware adds the CRC to the frame. If the device driver software adds the CRC, the CRC trailer can be considered another frame fragment which can be added by doing another gather DMA.

Each data byte is transmitted across the MII interface as two 4-bit values or the RMI interface as four 2-bit values. The Ethernet block adds the preamble, frame delimiter leader, and the CRC trailer if hardware CRC is enabled. Once transmission on the MII/RMI interface commences the transmission cannot be interrupted without generating an underrun error, which is why descriptors and data read commands are issued as soon as possible and pipelined.

Using an MII PHY, the data communication between the Ethernet block and the PHY is done at a 25 MHz rate. With an RMI PHY, the data communication between the Ethernet block and the PHY is at a 50 MHz rate. In 10 Mbps mode data will only be transmitted once every 10 clock cycles.

#### 11.17.4 Receive process

This section outlines the receive process including the activities in the device driver software.

##### Device driver sets up descriptors

After initializing the receive descriptor and status arrays to receive frames from the Ethernet connection, the receive data path should be enabled in the MAC1 register and the Control register.

During initialization, each Packet pointer in the descriptors is set to point to a data fragment buffer. The size of the buffer is stored in the Size bits of the Control field of the descriptor. Additionally, the Control field in the descriptor has an Interrupt bit. The Interrupt bit allows generation of an interrupt after a fragment buffer has been filled and its status has been committed to memory.

After the initialization and enabling of the receive data path, all descriptors are owned by the receive hardware and should not be modified by the software unless hardware hands over the descriptor by incrementing the RxProduceIndex, indicating that a frame has been received. The device driver is allowed to modify the descriptors after a (soft) reset of the receive data path.

##### Rx DMA manager reads Rx descriptor arrays

When the RxEnable bit in the Command register is set, the Rx DMA manager reads the descriptors from memory at the address determined by RxDescriptor and RxProduceIndex. The Ethernet block will start reading descriptors even before actual receive data arrives on the MII/RMI interface (descriptor prefetching). The block size of the descriptors to be read is determined by the total number of descriptors owned by the hardware: RxConsumeIndex - RxProduceIndex - 1. Block transferring of descriptors minimizes memory load. Read data returned from memory is buffered and consumed as needed.

##### RX DMA manager receives data

After reading the descriptor, the receive DMA engine waits for the MAC to return receive data from the MII/RMI interface that passes the receive filter. Receive frames that do not match the filtering criteria are not passed to memory. Once a frame passes the receive filter, the data is written in the fragment buffer associated with the descriptor. The Rx DMA does not write beyond the size of the buffer. When a frame is received that is larger than a descriptor's fragment buffer, the frame will be written to multiple fragment buffers of

consecutive descriptors. In the case of a multi-fragment reception, all but the last fragment in the frame will return a status where the LastFrag bit is set to 0. Only on the last fragment of a frame the LastFrag bit in the status will be set to 1. If a fragment buffer is the last of a frame, the buffer may not be filled completely. The first receive data of the next frame will be written to the fragment buffer of the next descriptor.

After receiving a fragment, the Rx DMA manager writes status information back to the StatusInfo and StatusHashCRC words of the status. The Ethernet block writes the size in bytes of a descriptor's fragment buffer in the RxSize field of the Status word. The value of the RxProduceIndex is only updated after the fragment data and the fragment status information has been committed to memory, which is checked by an internal tag protocol in the memory interface. The Rx DMA manager continues to receive frames until the descriptor array is full. If the descriptor array is full, the Ethernet hardware will set the RxFinishedInt bit of the IntStatus register. The receive data path will still be enabled. If the receive descriptor array is full any new receive data will generate an overflow error and interrupt.

### Update ProduceIndex

Each time the Rx DMA manager commits a data fragment and the associated status word to memory, it completes the reception of a descriptor and increments the RxProduceIndex (taking wrap around into account) in order to hand the descriptor back to the device driver software. Software can re-use the descriptor for new receptions by handing it back to hardware when the receive data has been processed.

The device driver software can keep track of the progress of the DMA manager by reading the RxProduceIndex register to see how far along the receive process is. When the Rx descriptor array is emptied completely, the RxProduceIndex retains its last value.

### Write reception status

After the frame has been received from the MII/RMII bus, the StatusInfo and StatusHashCRC words of the frame descriptor are updated by the DMA manager.

If the descriptor is for the last fragment of a frame (or for the whole frame if there are no fragments), then depending on the success or failure of the frame reception, error flags (Error, NoDescriptor, Overrun, AlignmentError, RangeError, LengthError, SymbolError, or CRCErrror) are set in StatusInfo. The RxSize field is set to the number of bytes actually written to the fragment buffer, -1 encoded. For fragments not being the last in the frame the RxSize will match the size of the buffer. The hash CRCs of the destination and source addresses of a packet are calculated once for all the fragments belonging to the same packet and then stored in every StatusHashCRC word of the statuses associated with the corresponding fragments. If the reception reports an error, any remaining data in the receive frame is discarded and the LastFrag bit will be set in the receive status field, so the error flags in all but the last fragment of a frame will always be 0.

The status of the last received frame can also be inspected by reading the RSV register. The register does not report statuses on a fragment basis and does not store information of previously received frames. RSV is provided primarily for debug purposes, because the communication between driver software and the Ethernet block takes place through the frame descriptors.

### Reception error handling

When an error occurs during the receive process, the Rx DMA manager will report the error via the receive StatusInfo written in the Status array and the IntStatus interrupt status register.

The receive process can generate several types of errors: AlignmentError, RangeError, LengthError, SymbolError, CRCError, Overrun, and NoDescriptor. All have corresponding bits in the receive StatusInfo. In addition to the separate bits in the StatusInfo, AlignmentError, RangeError, LengthError, SymbolError, and CRCError are ORed together into the Error bit of the StatusInfo. Errors are also propagated to the IntStatus register; the RxError bit in the IntStatus register is set if there is an AlignmentError, RangeError, LengthError, SymbolError, CRCError, or NoDescriptor error; nonfatal overrun errors are reported in the RxError bit of the IntStatus register; fatal Overrun errors are reported in the RxOverrun bit of the IntStatus register. On fatal overrun errors, the Rx data path needs to be soft reset by setting the RxReset bit in the Command register.

Overrun errors can have three causes:

- In the case of a multi-fragment reception, the next descriptor may be missing. In this case the NoDescriptor field is set in the status word of the previous descriptor and the RxError in the IntStatus register is set. This error is nonfatal.
- The data flow on the receiver data interface stalls, corrupting the packet. In this case the overrun bit in the status word is set and the RxError bit in the IntStatus register is set. This error is nonfatal.
- The flow of reception statuses stalls and a new status has to be written while a previous status still waits to be transferred across the memory interface. This error will corrupt the hardware state and requires the hardware to be soft reset. The error is detected and sets the Overrun bit in the IntStatus register.

The first overrun situation will result in an incomplete frame with a NoDescriptor status and the RxError bit in IntStatus set. Software should discard the partially received frame. In the second overrun situation the frame data will be corrupt which results in the Overrun status bit being set in the Status word while the IntError interrupt bit is set. In the third case receive errors cannot be reported in the receiver Status arrays which corrupts the hardware state; the errors will still be reported in the IntStatus register's Overrun bit. The RxReset bit in the Command register should be used to soft reset the hardware.

Device drivers should catch the above receive errors and take action.

### Receive triggers interrupts

The receive data path can generate four different interrupt types:

- If the Interrupt bit in the descriptor Control field is set, the Rx DMA will set the RxDoneInt bit in the IntStatus register after receiving a fragment and committing the associated data and status to memory. Even if a descriptor (fragment) is not the last in a multi-fragment frame, the Interrupt bit in the descriptor can be used to generate an interrupt.
- If the descriptor array is full while the Ethernet hardware is enabled, the hardware will set the RxFinishedInt bit of the IntStatus register.
- If the AHB interface does not consume receive statuses at a sufficiently high bandwidth, the receive status process may overrun, in which case the RxOverrun bit will be set in the IntStatus register.



- If there is a receive error (AlignmentError, RangeError, LengthError, SymbolError, or CRCError), or a multi-fragment frame where the device driver did provide descriptors for the initial fragments but did not provide the descriptors for the rest of the fragments, or if a nonfatal data Overrun occurred, the hardware will set the RxErrorInt bit of the IntStatus register.

All of the above interrupts can be enabled and disabled by setting or resetting the corresponding bits in the IntEnable register. Enabling or disabling does not affect the IntStatus register contents, only the propagation of the interrupt status to the CPU (via the NVIC).

The interrupts, either of individual frames or of the whole list, are a good means of communication between the DMA manager and the device driver, triggering the device driver to inspect the status words of descriptors that have been processed.

#### Device driver processes receive data

As a response to status (e.g. RxDoneInt) interrupts or polling of the RxProduceIndex, the device driver can read the descriptors that have been handed over to it by the hardware (RxProduceIndex - RxConsumeIndex). The device driver should inspect the status words in the status array to check for multi-fragment receptions and receive errors.

The device driver can forward receive data and status to upper software layers. After processing of data and status, the descriptors, statuses and data buffers may be recycled and handed back to hardware by incrementing the RxConsumeIndex.

#### Receive example

[Figure 25](#) illustrates the receive process in an example receiving a frame of 19 bytes.

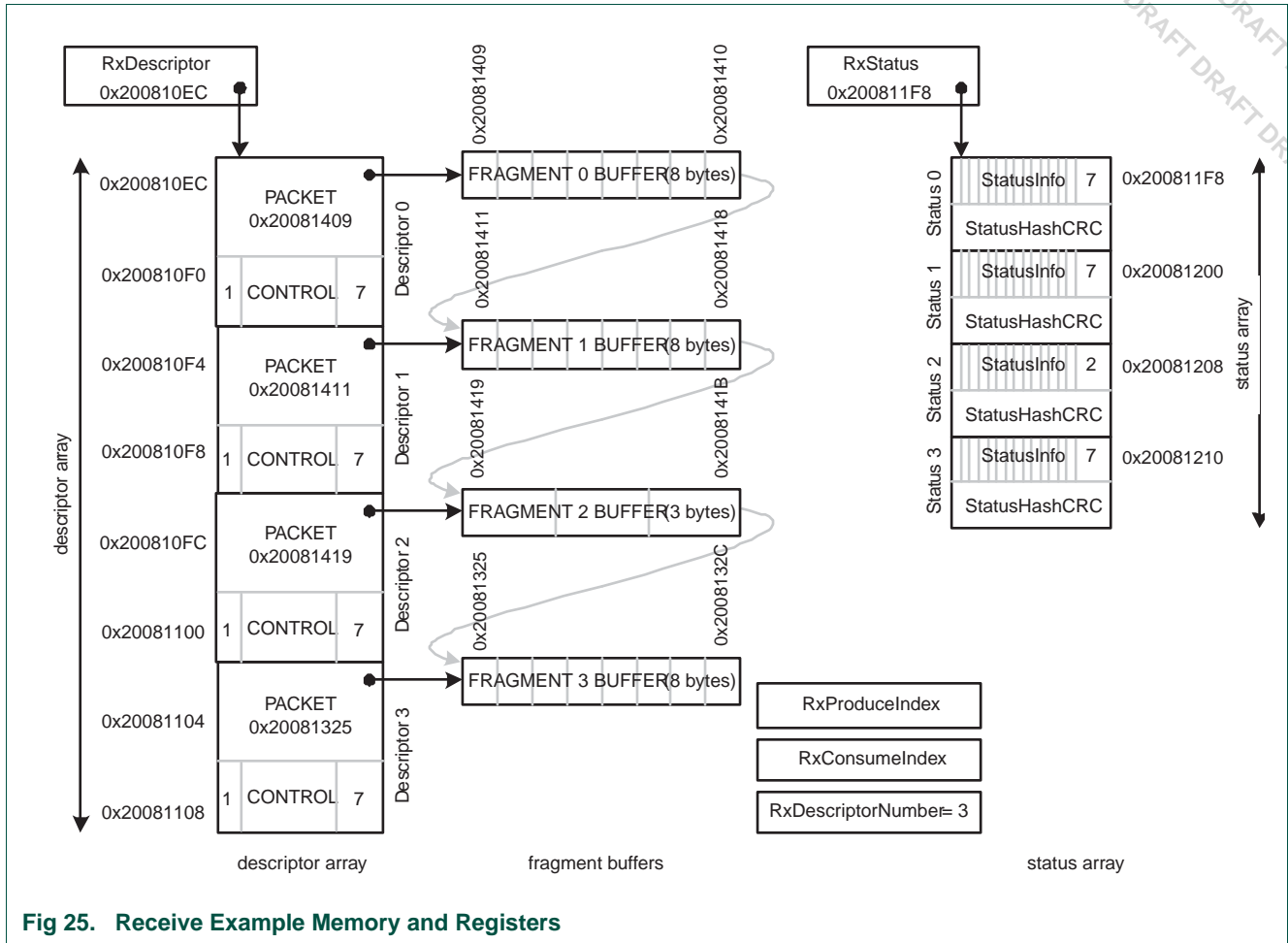


Fig 25. Receive Example Memory and Registers

After reset, the values of the DMA registers will be zero. During initialization, the device driver will allocate the descriptor and status array in memory. In this example, an array of four descriptors is allocated; the array is 4x2x4 bytes and aligned on a 4 byte address boundary. Since the number of descriptors matches the number of statuses, the status array consists of four elements; the array is 4x2x4 bytes and aligned on a 8 byte address boundary. The device driver writes the base address of the descriptor array (0x2008 10EC) in the RxDescriptor register, and the base address of the status array (0x2008 11F8) in the RxStatus register. The device driver writes the number of descriptors and statuses minus 1 (3) in the RxDescriptorNumber register. The descriptors and statuses in the arrays need not be initialized yet.

After allocating the descriptors, a fragment buffer needs to be allocated for each of the descriptors. Each fragment buffer can be between 1 byte and 2 k bytes. The base address of the fragment buffer is stored in the Packet field of the descriptors. The number of bytes in the fragment buffer is stored in the Size field of the descriptor Control word. The Interrupt field in the Control word of the descriptor can be set to generate an interrupt as soon as the descriptor has been filled by the receive process. In this example the fragment buffers are 8 bytes, so the value of the Size field in the Control word of the descriptor is set to 7. Note that in this example, the fragment buffers are actually a

continuous memory space; even when a frame is distributed over multiple fragments it will typically be in a linear, continuous memory space; when the descriptors wrap at the end of the descriptor array the frame will not be in a continuous memory space.

The device driver should enable the receive process by writing a 1 to the RxEnable bit of the Command register, after which the MAC needs to be enabled by writing a 1 to the 'RECEIVE ENABLE' bit of the MAC1 configuration register. The Ethernet block will now start receiving Ethernet frames. To reduce the processor interrupt load, some interrupts can be disabled by setting the relevant bits in the IntEnable register.

After the Rx DMA manager is enabled, it will start issuing descriptor read commands. In this example the number of descriptors is 4. Initially the RxProduceIndex and RxConsumeIndex are 0. Since the descriptor array is considered full if  $RxProduceIndex == RxConsumeIndex - 1$ , the Rx DMA manager can only read  $(RxConsumeIndex - RxProduceIndex - 1) = 3$  descriptors; note the wrapping.

After enabling the receive function in the MAC, data reception will begin starting at the next frame i.e. if the receive function is enabled while the MII/RMII interface is halfway through receiving a frame, the frame will be discarded and reception will start at the next frame. The Ethernet block will strip the preamble and start of frame delimiter from the frame. If the frame passes the receive filtering, the Rx DMA manager will start writing the frame to the first fragment buffer.

Suppose the frame is 19 bytes long. Due to the buffer sizes specified in this example, the frame will be distributed over three fragment buffers. After writing the initial 8 bytes in the first fragment buffer, the status for the first fragment buffer will be written and the Rx DMA will continue filling the second fragment buffer. Since this is a multi-fragment receive, the status of the first fragment will have a 0 for the LastFrag bit in the StatusInfo word; the RxSize field will be set to 7 (8, -1 encoded). After writing the 8 bytes in the second fragment the Rx DMA will continue writing the third fragment. The status of the second fragment will be like the status of the first fragment: LastFrag = 0, RxSize = 7. After writing the three bytes in the third fragment buffer, the end of the frame has been reached and the status of the third fragment is written. The third fragment's status will have the LastFrag bit set to 1 and the RxSize equal to 2 (3, -1 encoded).

The next frame received from the MII/RMII interface will be written to the fourth fragment buffer i.e. five bytes of the third buffer will be unused.

The Rx DMA manager uses an internal tag protocol in the memory interface to check that the receive data and status have been committed to memory. After the status of the fragments are committed to memory, an RxDoneInt interrupt will be triggered, which activates the device driver to inspect the status information. In this example, all descriptors have the Interrupt bit set in the Control word i.e. all descriptors will generate an interrupt after committing data and status to memory.

In this example the receive function cannot read new descriptors as long as the device driver does not increment the RxConsumeIndex, because the descriptor array is full (even though one descriptor is not programmed yet). Only after the device driver has forwarded the receive data to application software, and after the device driver has updated the RxConsumeIndex by incrementing it, will the Ethernet block can continue reading descriptors and receive data. The device driver will probably increment the RxConsumeIndex by 3, since the driver will forward the complete frame consisting of three fragments to the application, and hence free up three descriptors at the same time.

Each pair of nibbles transferred on the MII interface (or four pairs of bits for RMII) are transferred as a byte on the data write interface after being delayed by 128 or 136 cycles for filtering by the receive filter and buffer modules. The Ethernet block removes preamble, frame start delimiter, and CRC from the data and checks the CRC. To limit the buffer NoDescriptor error probability, three descriptors are buffered. The value of the RxProduceIndex is only updated after status information has been committed to memory, which is checked by an internal tag protocol in the memory interface. The software device driver will process the receive data, after which the device driver will update the RxConsumeIndex.

For an RMII PHY the data between the Ethernet block and the PHY is communicated at half the data-width and twice the clock frequency (50 MHz).

### 11.17.5 Transmission retry

If a collision on the Ethernet occurs, it usually takes place during the collision window spanning the first 64 bytes of a frame. If collision is detected, the Ethernet block will retry the transmission. For this purpose, the first 64 bytes of a frame are buffered, so that this data can be used during the retry. A transmission retry within the first 64 bytes in a frame is fully transparent to the application and device driver software.

When a collision occurs outside of the 64 byte collision window, a LateCollision error is triggered, and the transmission is aborted. After a LateCollision error, the remaining data in the transmit frame will be discarded. The Ethernet block will set the Error and LateCollision bits in the frame's status fields. The TxError bit in the IntStatus register will be set. If the corresponding bit in the IntEnable register is set, the TxError bit in the IntStatus register will be propagated to the CPU (via the NVIC). The device driver software should catch the interrupt and take appropriate actions.

The 'RETRANSMISSION MAXIMUM' field of the CLRT register can be used to configure the maximum number of retries before aborting the transmission.

### 11.17.6 Status hash CRC calculations

For each received frame, the Ethernet block is able to detect the destination address and source address and from them calculate the corresponding hash CRCs. To perform the computation, the Ethernet block features two internal blocks: one is a controller synchronized with the beginning and the end of each frame, the second block is the CRC calculator.

When a new frame is detected, internal signaling notifies the controller. The controller starts counting the incoming bytes of the frame, which correspond to the destination address bytes. When the sixth (and last) byte is counted, the controller notifies the calculator to store the corresponding 32-bit CRC into a first inner register. Then the controller repeats counting the next incoming bytes, in order to get synchronized with the source address. When the last byte of the source address is encountered, the controller again notifies the CRC calculator, which freezes until the next new frame. When the calculator receives this second notification, it stores the present 32-bit CRC into a second inner register. Then the CRCs remain frozen in their own registers until new notifications arise.

The destination address and source address hash CRCs being written in the StatusHashCRC word are the nine most significant bits of the 32-bit CRCs as calculated by the CRC calculator.

### 11.17.7 Duplex modes

The Ethernet block can operate in full duplex and half duplex mode. Half or full duplex mode needs to be configured by the device driver software during initialization.

For a full duplex connection the FullDuplex bit of the Command register needs to be set to 1 and the FULL-DUPLEX bit of the MAC2 configuration register needs to be set to 1; for half duplex the same bits need to be set to 0.

### 11.17.8 IEE 802.3/Clause 31 flow control

#### Overview

For full duplex connections, the Ethernet block supports IEEE 802.3/clause 31 flow control using pause frames. This type of flow control may be used in full-duplex point-to-point connections. Flow control allows a receiver to stall a transmitter e.g. when the receive buffers are (almost) full. For this purpose, the receiving side sends a pause frame to the transmitting side.

Pause frames use units of 512 bit times corresponding to 128 rx\_clk/tx\_clk cycles.

#### Receive flow control

In full-duplex mode, the Ethernet block will suspend its transmissions when the it receives a pause frame. Rx flow control is initiated by the receiving side of the transmission. It is enabled by setting the 'RX FLOW CONTROL' bit in the MAC1 configuration register. If the 'RX FLOW CONTROL' bit is zero, then the Ethernet block ignores received pause control frames. When a pause frame is received on the Rx side of the Ethernet block, transmission on the Tx side will be interrupted after the currently transmitting frame has completed, for an amount of time as indicated in the received pause frame. The transmit data path will stop transmitting data for the number of 512 bit slot times encoded in the pause-timer field of the received pause control frame.

By default the received pause control frames are not forwarded to the device driver. To forward the receive flow control frames to the device driver, set the 'PASS ALL RECEIVE FRAMES' bit in the MAC1 configuration register.

#### Transmit flow control

If case device drivers need to stall the receive data e.g. because software buffers are full, the Ethernet block can transmit pause control frames. Transmit flow control needs to be initiated by the device driver software; there is no IEEE 802.3/31 flow control initiated by hardware, such as the DMA managers.

With software flow control, the device driver can detect a situation in which the process of receiving frames needs to be interrupted by sending out Tx pause frames. Note that due to Ethernet delays, a few frames can still be received before the flow control takes effect and the receive stream stops.

Transmit flow control is activated by writing 1 to the TxFlowControl bit of the Command register. When the Ethernet block operates in full duplex mode, this will result in transmission of IEEE 802.3/31 pause frames. The flow control continues until a 0 is written to TxFlowControl bit of the Command register.

If the MAC is operating in full-duplex mode, then setting the TxFlowControl bit of the Command register will start a pause frame transmission. The value inserted into the pause-timer value field of transmitted pause frames is programmed via the PauseTimer[15:0] bits in the FlowControlCounter register. When the TxFlowControl bit is de-asserted, another pause frame having a pause-timer value of 0x0000 is automatically sent to abort flow control and resume transmission.

When flow control be in force for an extended time, a sequence of pause frames must be transmitted. This is supported with a mirror counter mechanism. To enable mirror counting, a nonzero value is written to the MirrorCounter[15:0] bits in the FlowControlCounter register. When the TxFlowControl bit is asserted, a pause frame is transmitted. After sending the pause frame, an internal mirror counter is initialized to zero. The internal mirror counter starts incrementing one every 512 bit-slot times. When the internal mirror counter reaches the MirrorCounter value, another pause frame is transmitted with pause-timer value equal to the PauseTimer field from the FlowControlCounter register, the internal mirror counter is reset to zero and restarts counting. The register MirrorCounter[15:0] is usually set to a smaller value than register PauseTimer[15:0] to ensure an early expiration of the mirror counter, allowing time to send a new pause frame before the transmission on the other side can resume. By continuing to send pause frames before the transmitting side finishes counting the pause timer, the pause can be extended as long as TxFlowControl is asserted. This continues until TxFlowControl is de-asserted when a final pause frame having a pause-timer value of 0x0000 is automatically sent to abort flow control and resume transmission. To disable the mirror counter mechanism, write the value 0 to MirrorCounter field in the FlowControlCounter register. When using the mirror counter mechanism, account for time-of-flight delays, frame transmission time, queuing delays, crystal frequency tolerances, and response time delays by programming the MirrorCounter conservatively, typically about 80% of the PauseTimer value.

If the software device driver sets the MirrorCounter field of the FlowControlCounter register to zero, the Ethernet block will only send one pause control frame. After sending the pause frame an internal pause counter is initialized at zero; the internal pause counter is incremented by one every 512 bit-slot times. Once the internal pause counter reaches the value of the PauseTimer register, the TxFlowControl bit in the Command register will be reset. The software device driver can poll the TxFlowControl bit to detect when the pause completes.

The value of the internal counter in the flow control module can be read out via the FlowControlStatus register. If the MirrorCounter is nonzero, the FlowControlStatus register will return the value of the internal mirror counter; if the MirrorCounter is zero the FlowControlStatus register will return the value of the internal pause counter value.

The device driver is allowed to dynamically modify the MirrorCounter register value and switch between zero MirrorCounter and nonzero MirrorCounter modes.

Transmit flow control is enabled via the 'TX FLOW CONTROL' bit in the MAC1 configuration register. If the 'TX FLOW CONTROL' bit is zero, then the MAC will not transmit pause control frames, software must not initiate pause frame transmissions, and the TxFlowControl bit in the Command register should be zero.

**Transmit flow control example**

Figure 26 illustrates the transmit flow control.

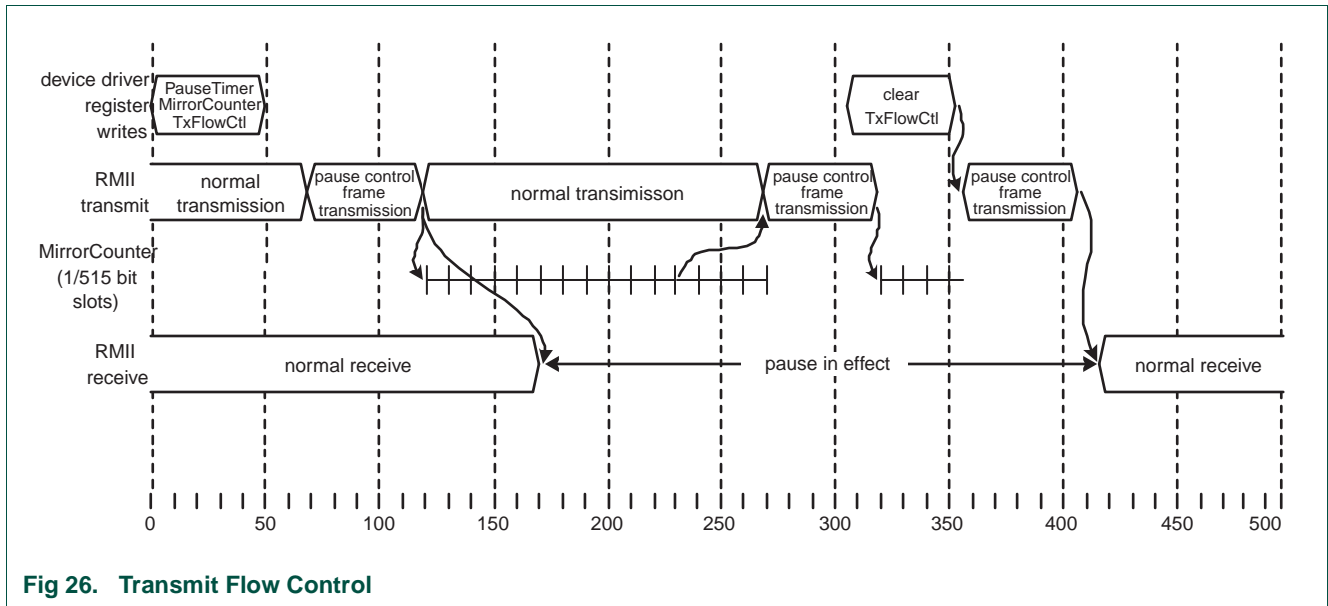


Fig 26. Transmit Flow Control

In this example, a frame is received while transmitting another frame (full duplex.) The device driver detects that some buffer might overrun and enables the transmit flow control by programming the PauseTimer and MirrorCounter fields of the FlowControlCounter register, after which it enables the transmit flow control by setting the TxFlowControl bit in the Command register.

As a response to the enabling of the flow control a pause control frame will be sent after the currently transmitting frame has been transmitted. When the pause frame transmission completes the internal mirror counter will start counting bit slots; as soon as the counter reaches the value in the MirrorCounter field another pause frame is transmitted. While counting the transmit data path will continue normal transmissions.

As soon as software disables transmit flow control a zero pause control frame is transmitted to resume the receive process.

**11.17.9 Half-Duplex mode backpressure**

When in half-duplex mode, backpressure can be generated to stall receive packets by sending continuous preamble that basically jams any other transmissions on the Ethernet medium. When the Ethernet block operates in half duplex mode, asserting the TxFlowControl bit in the Command register will result in applying continuous preamble on the Ethernet wire, effectively blocking traffic from any other Ethernet station on the same segment.

In half duplex mode, when the TxFlowControl bit goes high, continuous preamble is sent until TxFlowControl is de-asserted. If the medium is idle, the Ethernet block begins transmitting preamble, which raises carrier sense causing all other stations to defer. In the event the transmitting of preamble causes a collision, the backpressure 'rides through' the collision. The colliding station backs off and then defers to the backpressure. If during backpressure, the user wishes to send a frame, the backpressure is interrupted, the frame sent and then the backpressure resumed. If TxFlowControl is asserted for longer than 3.3 ms in 10 Mbps mode or 0.33 ms in 100 Mbps mode, backpressure will cease sending preamble for several byte times to avoid the jabber limit.

### 11.17.10 Receive filtering

#### Features of receive filtering

The Ethernet MAC has several receive packet filtering functions that can be configured from the software driver:

- Perfect address filter: allows packets with a perfectly matching station address to be identified and passed to the software driver.
- Hash table filter: allows imperfect filtering of packets based on the station address.
- Unicast/multicast/broadcast filtering: allows passing of all unicast, multicast, and/or broadcast packets.
- Magic packet filter: detection of magic packets to generate a Wake-on-LAN interrupt.

The filtering functions can be logically combined to create complex filtering functions. Furthermore, the Ethernet block can pass or reject runt packets smaller than 64 bytes; a promiscuous mode allows all packets to be passed to software.

#### Overview

The Ethernet block has the capability to filter out receive frames by analyzing the Ethernet destination address in the frame. This capability greatly reduces the load on the host system, because Ethernet frames that are addressed to other stations would otherwise need to be inspected and rejected by the device driver software, using up bandwidth, memory space, and host CPU time. Address filtering can be implemented using the perfect address filter or the (imperfect) hash filter. The latter produces a 6-bit hash code which can be used as an index into a 64 entry programmable hash table. [Figure 27](#) depicts a functional view of the receive filter.

At the top of the diagram the Ethernet receive frame enters the filters. Each filter is controlled by signals from control registers; each filter produces a 'Ready' output and a 'Match' output. If 'Ready' is 0 then the Match value is 'don't care'; if a filter finishes filtering then it will assert its Ready output; if the filter finds a matching frame it will assert the Match output along with the Ready output. The results of the filters are combined by logic functions into a single RxAbort output. If the RxAbort output is asserted, the frame does not need to be received.

In order to reduce memory traffic, the receive data path has a buffer of 68 bytes. The Ethernet MAC will only start writing a frame to memory after 68 byte delays. If the RxAbort signal is asserted during the initial 68 bytes of the frame, the frame can be discarded and removed from the buffer and not stored to memory at all, not using up receive descriptors, etc. If the RxAbort signal is asserted after the initial 68 bytes in a frame (probably due to reception of a Magic Packet), part of the frame is already written to memory and the



Ethernet MAC will stop writing further data in the frame to memory; the FailFilter bit in the status word of the frame will be set to indicate that the software device driver can discard the frame immediately.

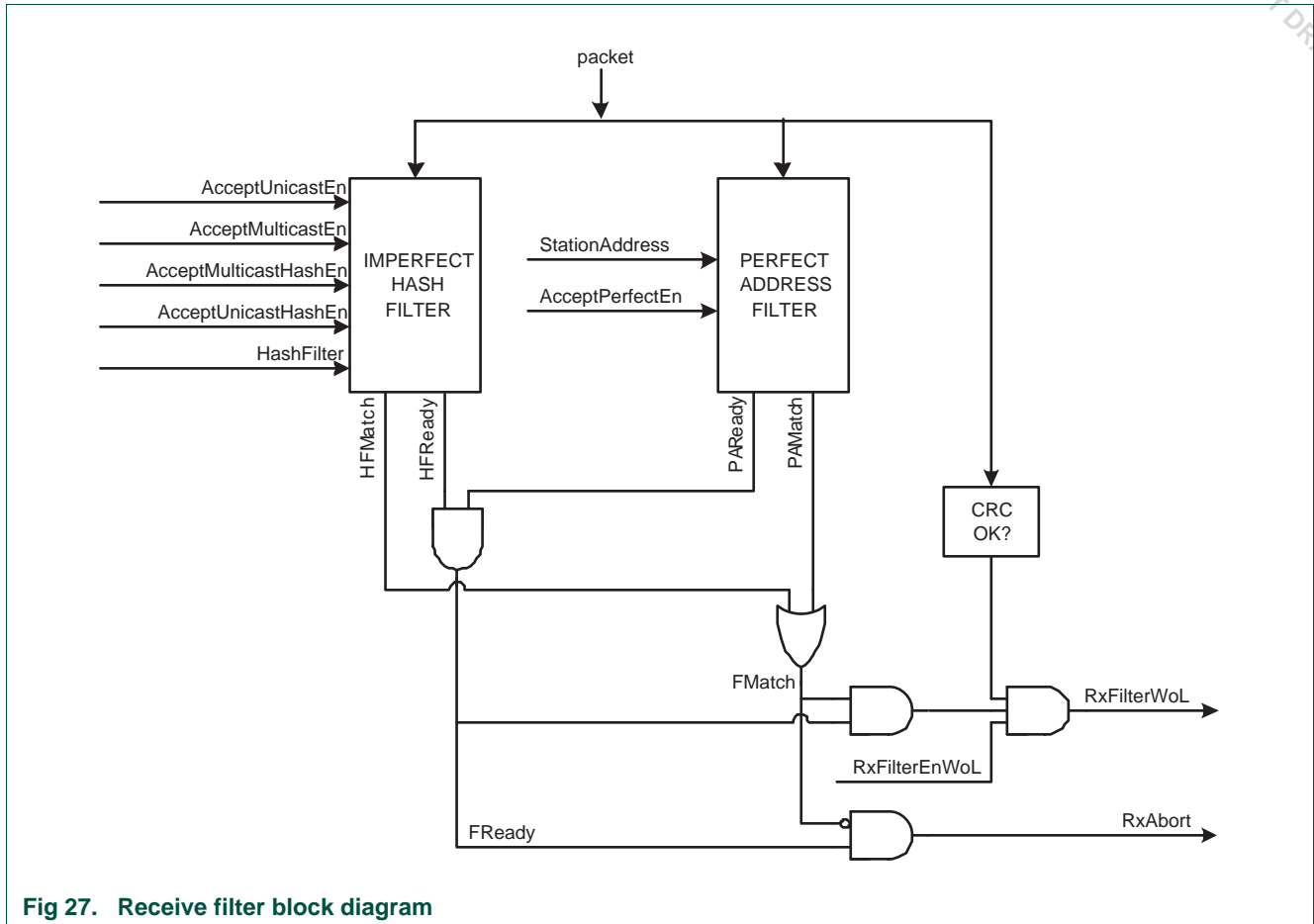


Fig 27. Receive filter block diagram

**Unicast, broadcast and multicast**

Generic filtering based on the type of frame (unicast, multicast or broadcast) can be programmed using the AcceptUnicastEn, AcceptMulticastEn, or AcceptBroadcastEn bits of the RxFilterCtrl register. Setting the AcceptUnicast, AcceptMulticast, and AcceptBroadcast bits causes all frames of types unicast, multicast and broadcast, respectively, to be accepted, ignoring the Ethernet destination address in the frame. To program promiscuous mode, i.e. to accept all frames, set all 3 bits to 1.

**Perfect address match**

When a frame with a unicast destination address is received, a perfect filter compares the destination address with the 6 byte station address programmed in the station address registers SA0, SA1, SA2. If the AcceptPerfectEn bit in the RxFilterCtrl register is set to 1, and the address matches, the frame is accepted.

**Imperfect hash filtering**

An imperfect filter is available, based on a hash mechanism. This filter applies a hash function to the destination address and uses the hash to access a table that indicates if the frame should be accepted. The advantage of this type of filter is that a small table can cover any possible address. The disadvantage is that the filtering is imperfect, i.e. sometimes frames are accepted that should have been discarded.

- Hash function:
  - The standard Ethernet cyclic redundancy check (CRC) function is calculated from the 6 byte destination address in the Ethernet frame (this CRC is calculated anyway as part of calculating the CRC of the whole frame), then bits [28:23] out of the 32-bit CRC result are taken to form the hash. The 6-bit hash is used to access the hash table: it is used as an index in the 64-bit HashFilter register that has been programmed with accept values. If the selected accept value is 1, the frame is accepted.
  - The device driver can initialize the hash filter table by writing to the registers HashFilterL and HashfilterH. HashFilterL contains bits 0 through 31 of the table and HashFilterH contains bit 32 through 63 of the table. So, hash value 0 corresponds to bit 0 of the HashfilterL register and hash value 63 corresponds to bit 31 of the HashFilterH register.
- Multicast and unicast
  - The imperfect hash filter can be applied to multicast addresses, by setting the AcceptMulticastHashEn bit in the RxFilter register to 1.
  - The same imperfect hash filter that is available for multicast addresses can also be used for unicast addresses. This is useful to be able to respond to a multitude of unicast addresses without enabling all unicast addresses. The hash filter can be applied to unicast addresses by setting the AcceptUnicastHashEn bit in the RxFilter register to 1.

### Enabling and disabling filtering

The filters as defined in the sections above can be bypassed by setting the PassRxFilter bit in the Command register. When the PassRxFilter bit is set, all receive frames will be passed to memory. In this case the device driver software has to implement all filtering functionality in software. Setting the PassRxFilter bit does not affect the runt frame filtering as defined in the next section.

### Runt frames

A frame with less than 64 bytes (or 68 bytes for VLAN frames) is shorter than the minimum Ethernet frame size and therefore considered erroneous; they might be collision fragments. The receive data path automatically filters and discards these runt frames without writing them to memory and using a receive descriptor.

When a runt frame has a correct CRC there is a possibility that it is intended to be useful. The device driver can receive the runt frames with correct CRC by setting the PassRuntFrame bit of the Command register to 1.

#### 11.17.11 Power management

The Ethernet block supports power management by means of clock switching. All clocks in the Ethernet core can be switched off. If Wake-up on LAN is needed, the rx\_clk should not be switched off.

### 11.17.12 Wake-up on LAN

#### Overview

The Ethernet block supports power management with remote wake-up over LAN. The host system can be powered down, even including part of the Ethernet block itself, while the Ethernet block continues to listen to packets on the LAN. Appropriately formed packets can be received and recognized by the Ethernet block and used to trigger the host system to wake up from its power-down state.

Wake-up of the system takes effect through an interrupt. When a wake-up event is detected, the WakeupInt bit in the IntStatus register is set. The interrupt status will trigger an interrupt if the corresponding WakeupIntEn bit in the IntEnable register is set. This interrupt should be used by system power management logic to wake up the system.

While in a power-down state the packet that generates a Wake-up on LAN event is lost.

There are two ways in which Ethernet packets can trigger wake-up events: generic Wake-up on LAN and Magic Packet. Magic Packet filtering uses an additional filter for Magic Packet detection. In both cases a Wake-up on LAN event is only triggered if the triggering packet has a valid CRC. [Figure 27](#) shows the generation of the wake-up signal.

The RxFilterWoLStatus register can be read by the software to inspect the reason for a Wake-up event. Before going to power-down the power management software should clear the register by writing the RxFilterWoLClear register.

**NOTE:** when entering in power-down mode, a receive frame might be not entirely stored into the Rx buffer. In this situation, after turning exiting power-down mode, the next receive frame is corrupted due to the data of the previous frame being added in front of the last received frame. Software drivers have to reset the receive data path just after exiting power-down mode.

The following subsections describe the two Wake-up on LAN mechanisms.

#### Filtering for WoL

The receive filter functionality can be used to generate Wake-up on LAN events. If the RxFilterEnWoL bit of the RxFilterCtrl register is set, the receive filter will set the WakeupInt bit of the IntStatus register if a frame is received that passes the filter. The interrupt will only be generated if the CRC of the frame is correct.

#### Magic Packet WoL

The Ethernet block supports wake-up using Magic Packet technology (see 'Magic Packet technology', Advanced Micro Devices). A Magic Packet is a specially formed packet solely intended for wake-up purposes. This packet can be received, analyzed and recognized by the Ethernet block and used to trigger a wake-up event.

A Magic Packet is a packet that contains in its data portion the station address repeated 16 times with no breaks or interruptions, preceded by 6 Magic Packet synchronization bytes with the value 0xFF. Other data may be surrounding the Magic Packet pattern in the data portion of the packet. The whole packet must be a well-formed Ethernet frame.

The magic packet detection unit analyzes the Ethernet packets, extracts the packet address and checks the payload for the Magic Packet pattern. The address from the packet is used for matching the pattern (not the address in the SA0/1/2 registers.) A magic packet only sets the wake-up interrupt status bit if the packet passes the receive filter as illustrated in [Figure 27](#): the result of the receive filter is ANDed with the magic packet filter result to produce the result.

Magic Packet filtering is enabled by setting the MagicPacketEnWoL bit of the RxFilterCtrl register. Note that when doing Magic Packet WoL, the RxFilterEnWoL bit in the RxFilterCtrl register should be 0. Setting the RxFilterEnWoL bit to 1 would accept all packets for a matching address, not just the Magic Packets i.e. WoL using Magic Packets is more strict.

When a magic packet is detected, apart from the WakeupInt bit in the IntStatus register, the MagicPacketWoL bit is set in the RxFilterWoLStatus register. Software can reset the bit writing a 1 to the corresponding bit of the RxFilterWoLClear register.

**Example:** An example of a Magic Packet with station address 0x11 0x22 0x33 0x44 0x55 0x66 is the following (MISC indicates miscellaneous additional data bytes in the packet):

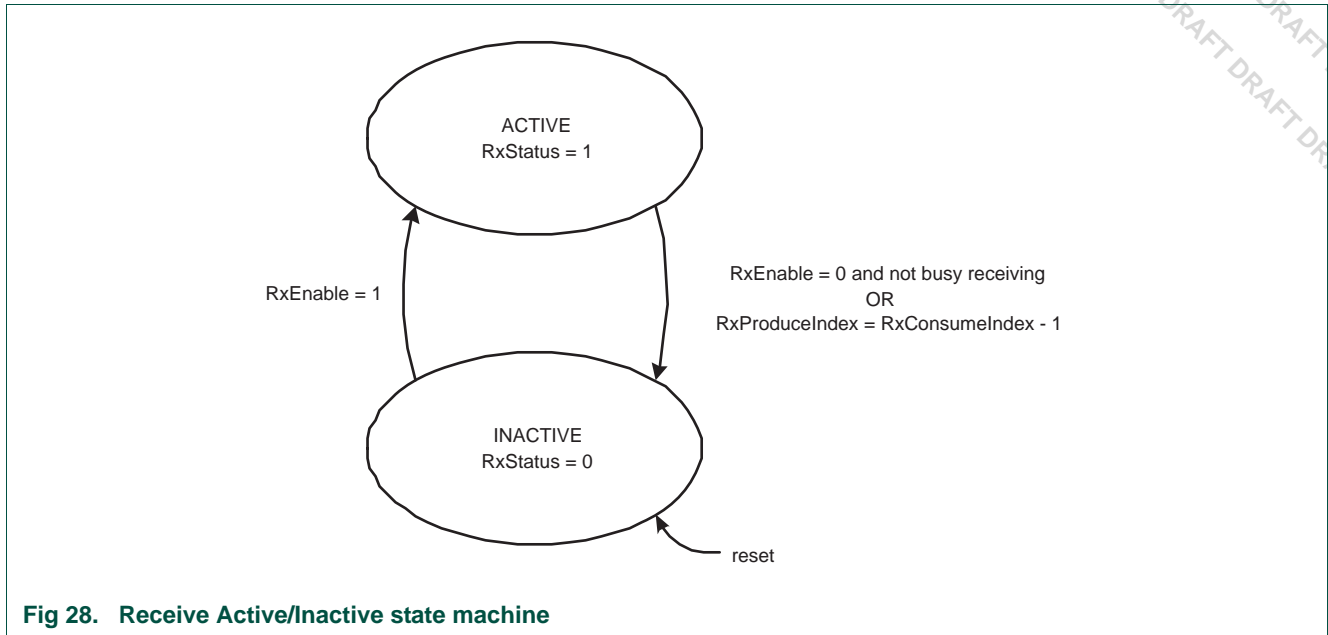
```
<DESTINATION> <SOURCE> <MISC>
FF FF FF FF FF FF
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
11 22 33 44 55 66 11 22 33 44 55 66
<MISC> <CRC>
```

### 11.17.13 Enabling and disabling receive and transmit

#### Enabling and disabling reception

After reset, the receive function of the Ethernet block is disabled. The receive function can be enabled by the device driver setting the RxEnable bit in the Command register and the "RECEIVE ENABLE" bit in the MAC1 configuration register (in that order).

The status of the receive data path can be monitored by the device driver by reading the RxStatus bit of the Status register. [Figure 28](#) illustrates the state machine for the generation of the RxStatus bit.



**Fig 28. Receive Active/Inactive state machine**

After a reset, the state machine is in the INACTIVE state. As soon as the RxEnable bit is set in the Command register, the state machine transitions to the ACTIVE state. As soon as the RxEnable bit is cleared, the state machine returns to the INACTIVE state. If the receive data path is busy receiving a packet while the receive data path gets disabled, the packet will be received completely, stored to memory along with its status before returning to the INACTIVE state. Also if the Receive descriptor array is full, the state machine will return to the INACTIVE state.

For the state machine in [Figure 28](#), a soft reset is like a hardware reset assertion, i.e. after a soft reset the receive data path is inactive until the data path is re-enabled.

**Enabling and disabling transmission**

After reset, the transmit function of the Ethernet block is disabled. The Tx transmit data path can be enabled by the device driver setting the TxEnable bit in the Command register to 1.

The status of the transmit data paths can be monitored by the device driver reading the TxStatus bit of the Status register. [Figure 29](#) illustrates the state machine for the generation of the TxStatus bit.

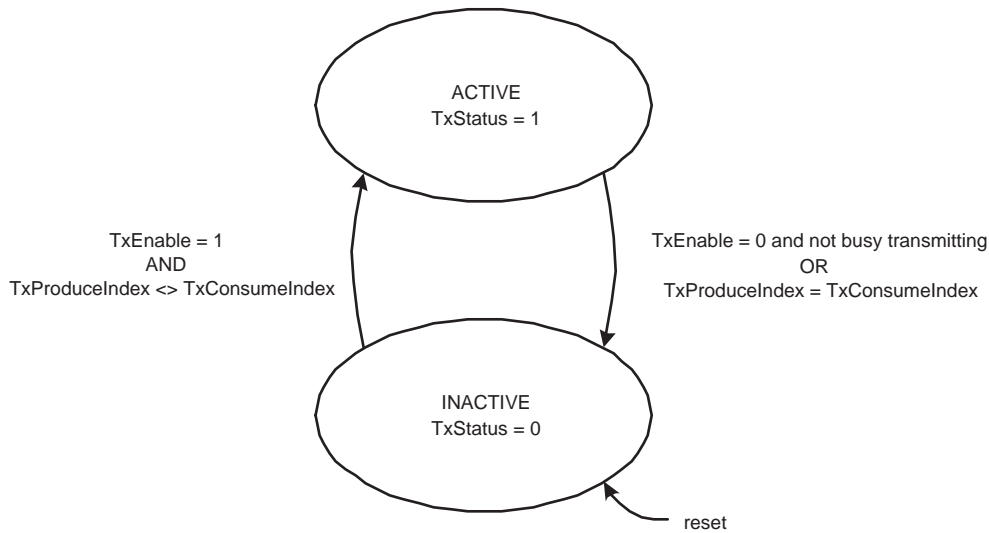


Fig 29. Transmit Active/Inactive state machine

After reset, the state machine is in the INACTIVE state. As soon as the TxEnable bit is set in the Command register and the Produce and Consume indices are not equal, the state machine transitions to the ACTIVE state. As soon as the TxEnable bit is cleared and the transmit data path has completed all pending transmissions, including committing the transmission status to memory, the state machine returns to the INACTIVE state. The state machine will also return to the INACTIVE state if the Produce and Consume indices are equal again i.e. all frames have been transmitted.

For the state machine in [Figure 29](#), a soft reset is like a hardware reset assertion, i.e. after a soft reset the transmit data path is inactive until the data path is re-enabled.

### 11.17.14 Transmission padding and CRC

In the case of a frame of less than 60 bytes (or 64 bytes for VLAN frames), the Ethernet block can pad the frame to 64 or 68 bytes including a 4 bytes CRC Frame Check Sequence (FCS). Padding is affected by the value of the 'AUTO DETECT PAD ENABLE' (ADPEN), 'VLAN PAD ENABLE' (VLPEN) and 'PAD/CRC ENABLE' (PADEN) bits of the MAC2 configuration register, as well as the Override and Pad bits from the transmit descriptor Control word. CRC generation is affected by the 'CRC ENABLE' (CRCE) and 'DELAYED CRC' (DCRC) bits of the MAC2 configuration register, and the Override and CRC bits from the transmit descriptor Control word.

The effective pad enable (EPADEN) is equal to the 'PAD/CRC ENABLE' bit from the MAC2 register if the Override bit in the descriptor is 0. If the Override bit is 1, then EPADEN will be taken from the descriptor Pad bit. Likewise the effective CRC enable (ECRCE) equals CRCE if the Override bit is 0, otherwise it equal the CRC bit from the descriptor.

If padding is required and enabled, a CRC will always be appended to the padded frames. A CRC will only be appended to the non-padded frames if ECRCE is set.

If EPADEN is 0, the frame will not be padded and no CRC will be added unless ECRCE is set.

If EPADEN is 1, then small frames will be padded and a CRC will always be added to the padded frames. In this case if ADPEN and VLPEN are both 0, then the frames will be padded to 60 bytes and a CRC will be added creating 64 bytes frames; if VLPEN is 1, the frames will be padded to 64 bytes and a CRC will be added creating 68 bytes frames; if ADPEN is 1, while VLPEN is 0 VLAN frames will be padded to 64 bytes, non VLAN frames will be padded to 60 bytes, and a CRC will be added to padded frames, creating 64 or 68 bytes padded frames.

If CRC generation is enabled, CRC generation can be delayed by four bytes by setting the DELAYED CRC bit in the MAC2 register, in order to skip proprietary header information.

### 11.17.15 Huge frames and frame length checking

The 'HUGE FRAME ENABLE' bit in the MAC2 configuration register can be set to 1 to enable transmission and reception of frames of any length. Huge frame transmission can be enabled on a per frame basis by setting the Override and Huge bits in the transmit descriptor Control word.

When enabling huge frames, the Ethernet block will not check frame lengths and report frame length errors (RangeError and LengthError). If huge frames are enabled, the received byte count in the RSV register may be invalid because the frame may exceed the maximum size; the RxSize fields from the receive status arrays will be valid.

Frame lengths are checked by comparing the length/type field of the frame to the actual number of bytes in the frame. A LengthError is reported by setting the corresponding bit in the receive StatusInfo word.

The MAXF register allows the device driver to specify the maximum number of bytes in a frame. The Ethernet block will compare the actual receive frame to the MAXF value and report a RangeError in the receive StatusInfo word if the frame is larger.

### 11.17.16 Statistics counters

Generally, Ethernet applications maintain many counters that track Ethernet traffic statistics. There are a number of standards specifying such counters, such as IEEE std 802.3 / clause 30. Other standards are RFC 2665 and RFC 2233.

The approach taken here is that by default all counters are implemented in software. With the help of the StatusInfo field in frame statuses, many of the important statistics events listed in the standards can be counted by software.

### 11.17.17 MAC status vectors

Transmit and receive status information as detected by the MAC are available in registers TSV0, TSV1 and RSV so that software can poll them. These registers are normally of limited use because the communication between driver software and the Ethernet block takes place primarily through frame descriptors. Statistical events can be counted by software in the device driver. However, for debug purposes the transmit and receive status vectors are made visible. They are valid as long as the internal status of the MAC is valid and should typically only be read when the transmit and receive processes are halted.

### 11.17.18 Reset

The Ethernet block has a hard reset input which is connected to the chip reset, as well as several soft resets which can be activated by setting the appropriate bit(s) in registers. All registers in the Ethernet block have a value of 0 after a hard reset, unless otherwise specified.

#### Hard reset

After a hard reset, all registers will be set to their default value.

#### Soft reset

Parts of the Ethernet block can be soft reset by setting bits in the Command register and the MAC1 configuration register. The MAC1 register has six different reset bits:

- **SOFT RESET:** Setting this bit will put all modules in the MAC in reset, except for the MAC registers (at addresses 0x000 to 0x0FC). The value of the soft reset after a hardware reset assertion is 1, i.e. the soft reset needs to be cleared after a hardware reset.
- **SIMULATION RESET:** Resets the random number generator in the Transmit Function. The value after a hardware reset assertion is 0.
- **RESET MCS/Rx:** Setting this bit will reset the MAC Control Sublayer (pause frame logic) and the receive function in the MAC. The value after a hardware reset assertion is 0.
- **RESET Rx:** Setting this bit will reset the receive function in the MAC. The value after a hardware reset assertion is 0.
- **RESET MCS/Tx:** Setting this bit will reset the MAC Control Sublayer (pause frame logic) and the transmit function in the MAC. The value after a hardware reset assertion is 0.
- **RESET Tx:** Setting this bit will reset the transmit function of the MAC. The value after a hardware reset assertion is 0.

The above reset bits must be cleared by software.

The Command register has three different reset bits:

- **TxReset:** Writing a '1' to the TxReset bit will reset the transmit data path, excluding the MAC portions, including all (read-only) registers in the transmit data path, as well as the TxProduceIndex register in the host registers module. A soft reset of the transmit data path will abort all AHB transactions of the transmit data path. The reset bit will be cleared autonomously by the Ethernet block. A soft reset of the Tx data path will clear the TxStatus bit in the Status register.
- **RxReset:** Writing a '1' to the RxReset bit will reset the receive data path, excluding the MAC portions, including all (read-only) registers in the receive data path, as well as the RxConsumeIndex register in the host registers module. A soft reset of the receive data path will abort all AHB transactions of the receive data path. The reset bit will be cleared autonomously by the Ethernet block. A soft reset of the Rx data path will clear the RxStatus bit in the Status register.



- RegReset: Resets all of the data paths and registers in the host registers module, excluding the registers in the MAC. A soft reset of the registers will also abort all AHB transactions of the transmit and receive data path. The reset bit will be cleared autonomously by the Ethernet block.

To do a full soft reset of the Ethernet block, device driver software must:

- Set the 'SOFT RESET' bit in the MAC1 register to 1.
- Set the RegReset bit in the Command register, this bit clears automatically.
- Re-initialize the MAC registers (0x000 to 0x0FC).
- Reset the 'SOFT RESET' bit in the MAC1 register to 0.

To reset just the transmit data path, the device driver software has to:

- Set the 'RESET MCS/Tx' bit in the MAC1 register to 1.
- Disable the Tx DMA managers by setting the TxEnable bits in the Command register to 0.
- Set the TxReset bit in the Command register, this bit clears automatically.
- Reset the 'RESET MCS/Tx' bit in the MAC1 register to 0.

To reset just the receive data path, the device driver software has to:

- Disable the receive function by resetting the 'RECEIVE ENABLE' bit in the MAC1 configuration register and resetting of the RxEnable bit of the Command register.
- Set the 'RESET MCS/Rx' bit in the MAC1 register to 1.
- Set the RxReset bit in the Command register, this bit clears automatically.
- Reset the 'RESET MCS/Rx' bit in the MAC1 register to 0.

### 11.17.19 Ethernet errors

The Ethernet block generates errors for the following conditions:

- A reception can cause an error: AlignmentError, RangeError, LengthError, SymbolError, CRCError, NoDescriptor, or Overrun. These are reported back in the receive StatusInfo and in the interrupt status register (IntStatus).
- A transmission can cause an error: LateCollision, ExcessiveCollision, ExcessiveDefer, NoDescriptor, or Underrun. These are reported back in the transmission StatusInfo and in the interrupt status register (IntStatus).

## 11.18 AHB bandwidth

The Ethernet block is connected to an AHB bus which must carry all of the data and control information associated with all Ethernet traffic in addition to the CPU accesses required to operate the Ethernet block and deal with message contents.

### 11.18.1 DMA access

#### Assumptions

By making some assumptions, the bandwidth needed for each type of AHB transfer can be calculated and added in order to find the overall bandwidth requirement.

The flexibility of the descriptors used in the Ethernet block allows the possibility of defining memory buffers in a range of sizes. In order to analyze bus bandwidth requirements, some assumptions must be made about these buffers. The "worst case" is not addressed since that would involve all descriptors pointing to single byte buffers, with most of the memory occupied in holding descriptors and very little data. It can easily be shown that the AHB cannot handle the huge amount of bus traffic that would be caused by such a degenerate (and illogical) case.

For this analysis, an Ethernet packet is assumed to consist of a 64 byte frame. Continuous traffic is assumed on both the transmit and receive channels.

This analysis does not reflect the flow of Ethernet traffic over time, which would include inter-packet gaps in both the transmit and receive channels that reduce the bandwidth requirements over a larger time frame.

#### Types of DMA access and their bandwidth requirements

The interface to an external Ethernet PHY is via RMII. RMII operates at 50 MHz, transferring a byte in 4 clock cycles. The data transfer rate is 12.5 Mbps.

The interface to an external Ethernet PHY is via either MII or RMII. An interface MII operates at 25 MHz, transferring a byte in 2 clock cycles. An RMII interface operates at 50 MHz, transferring a byte in 4 clock cycles. The data transfer rate is the same in both cases: 12.5 Mbps.

The Ethernet block initiates DMA accesses for the following cases:

- Tx descriptor read:
  - Transmit descriptors occupy 2 words (8 bytes) of memory and are read once for each use of a descriptor.
  - Two word read happens once every 64 bytes (16 words) of transmitted data.
  - This gives 1/8th of the data rate, which = 1.5625 Mbps.
- Rx descriptor read:
  - Receive descriptors occupy 2 words (8 bytes) of memory and are read once for each use of a descriptor.
  - Two word read happens once every 64 bytes (16 words) of received data.
  - This gives 1/8th of the data rate, which = 1.5625 Mbps.
- Tx status write:

- Transmit status occupies 1 word (4 bytes) of memory and is written once for each use of a descriptor.
- One word write happens once every 64 bytes (16 words) of transmitted data.
- This gives 1/16th of the data rate, which = 0.7813 Mbps.
- Rx status write:
  - Receive status occupies 2 words (8 bytes) of memory and is written once for each use of a descriptor.
  - Two word write happens once every 64 bytes (16 words) of received data.
  - This gives 1/8 of the data rate, which = 1.5625 Mbps.
- Tx data read:
  - Data transmitted in an Ethernet frame, the size is variable.
  - Basic Ethernet rate = 12.5 Mbps.
- Rx data write:
  - Data to be received in an Ethernet frame, the size is variable.
  - Basic Ethernet rate = 12.5 Mbps.

This gives a total rate of 30.5 Mbps for the traffic generated by the Ethernet DMA function.

### 11.18.2 Types of CPU access

- Accesses that mirror each of the DMA access types:
  - All or part of status values must be read, and all or part of descriptors need to be written after each use, transmitted data must be stored in the memory by the CPU, and eventually received data must be retrieved from the memory by the CPU.
  - This gives roughly the same or slightly lower rate as the combined DMA functions, which = 30.5 Mbps.
- Access to registers in the Ethernet block:
  - The CPU must read the RxProduceIndex, TxConsumeIndex, and IntStatus registers, and both read and write the RxConsumeIndex and TxProduceIndex registers.
  - 7 word read/writes once every 64 bytes (16 words) of transmitted and received data.
  - This gives 7/16 of the data rate, which = 5.4688 Mbps.

This gives a total rate of 36 Mbps for the traffic generated by the Ethernet DMA function.

### 11.18.3 Overall bandwidth

Overall traffic on the AHB is the sum of DMA access rates and CPU access rates, which comes to approximately 66.5 MB/s.

The peak bandwidth requirement can be somewhat higher due to the use of small memory buffers, in order to hold often used addresses (e.g. the station address) for example. Driver software can determine how to build frames in an efficient manner that does not overutilize the AHB.

The bandwidth available on the AHB bus depends on the system clock frequency. As an example, assume that the system clock is set at 60 MHz. All or nearly all of bus accesses related to the Ethernet will be word transfers. The raw AHB bandwidth can be approximated as 4 bytes per two system clocks, which equals 2 times the system clock rate. With a 60 MHz system clock, the bandwidth is 120 MB/s, giving about 55% utilization for Ethernet traffic during simultaneous transmit and receive operations. This shows that it is not necessary to use the maximum CPU frequency for the Ethernet to work with plenty of bandwidth headroom.

## 11.19 CRC calculation

The calculation is used for several purposes:

- Generation the FCS at the end of the Ethernet frame.
- Generation of the hash table index for the hash table filtering.
- Generation of the destination and source address hash CRCs.

The C pseudocode function below calculates the CRC on a frame taking the frame (without FCS) and the number of bytes in the frame as arguments. The function returns the CRC as a 32-bit integer.

```
int crc_calc(char frame_no_fcs[], int frame_len) {
    int i; // iterator
    int j; // another iterator
    char byte; // current byte
    int crc; // CRC result
    int q0, q1, q2, q3; // temporary variables
    crc = 0xFFFFFFFF;
    for (i = 0; i < frame_len; i++) {
        byte = *frame_no_fcs++;
        for (j = 0; j < 2; j++) {
            if (((crc >> 28) ^ (byte >> 3)) & 0x00000001) {
                q3 = 0x04C11DB7;
            } else {
                q3 = 0x00000000;
            }
            if (((crc >> 29) ^ (byte >> 2)) & 0x00000001) {
                q2 = 0x09823B6E;
            } else {
                q2 = 0x00000000;
            }
            if (((crc >> 30) ^ (byte >> 1)) & 0x00000001) {
                q1 = 0x130476DC;
            } else {
                q1 = 0x00000000;
            }
            if (((crc >> 31) ^ (byte >> 0)) & 0x00000001) {
                q0 = 0x2608EDB8;
            } else {
                q0 = 0x00000000;
            }
            crc = (crc << 4) ^ q3 ^ q2 ^ q1 ^ q0;
            byte >>= 4;
        }
    }
    return crc;
}
```

For FCS calculation, this function is passed a pointer to the first byte of the frame and the length of the frame without the FCS.

For hash filtering, this function is passed a pointer to the destination address part of the frame and the CRC is only calculated on the 6 address bytes. The hash filter uses bits [28:23] for indexing the 64-bits { HashFilterH, HashFilterL } vector. If the corresponding bit is set the packet is passed, otherwise it is rejected by the hash filter.

For obtaining the destination and source address hash CRCs, this function calculates first both the 32-bit CRCs, then the nine most significant bits from each 32-bit CRC are extracted, concatenated, and written in every StatusHashCRC word of every fragment status.

### 12.1 How to read this chapter

---

The LCD controller is available on some LPC178x/177x devices, see [Section 1.4](#) for details.

### 12.2 Basic configuration

---

The LCD controller is configured using the following registers:

1. Power: In the PCONP register ([Section 4.7.9](#)), set bit PCLCD.  
**Remark:** The LCD is disabled on reset (PCLCD = 0).  
Also see [Section 12.6.12](#) for power-up procedure.
2. Clock: See [Table 224](#), and [Table 221](#).
3. Pins: Select LCD pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 12.3 Introduction

---

The LCD controller provides all of the necessary control signals to interface directly to a variety of color and monochrome LCD panels.

## 12.4 Features

- AHB bus master interface to access frame buffer.
- Setup and control via a separate AHB slave interface.
- Dual 16-deep programmable 64-bit wide FIFOs for buffering incoming display data.
- Supports single and dual-panel monochrome Super Twisted Nematic (STN) displays with 4 or 8-bit interfaces.
- Supports single and dual-panel color STN displays.
- Supports Thin Film Transistor (TFT) color displays.
- Programmable display resolution including, but not limited to: 320x200, 320x240, 640x200, 640x240, 640x480, 800x600, and 1024x768.
- Hardware cursor support for single-panel displays.
- 15 gray-level monochrome, 3375 color STN, and 32K color palettized TFT support.
- 1, 2, or 4 bits-per-pixel (bpp) palettized displays for monochrome STN.
- 1, 2, 4, or 8 bpp palettized color displays for color STN and TFT.
- 16 bpp true-color non-palettized, for color STN and TFT.
- 24 bpp true-color non-palettized, for color TFT.
- Programmable timing for different display panels.
- 256 entry, 16-bit palette RAM, arranged as a 128x32-bit RAM.
- Frame, line, and pixel clock signals.
- AC bias signal for STN, data enable signal for TFT panels.
- Supports little and big-endian, and Windows CE data formats.
- LCD panel clock may be generated from the peripheral clock, or from a clock input pin.

### 12.4.1 Programmable parameters

The following key display and controller parameters can be programmed:

- Horizontal front and back porch
- Horizontal synchronization pulse width
- Number of pixels per line
- Vertical front and back porch
- Vertical synchronization pulse width
- Number of lines per panel
- Number of pixel clocks per line
- Hardware cursor control.
- Signal polarity, active HIGH or LOW
- AC panel bias
- Panel clock frequency
- Bits-per-pixel
- Display type: STN monochrome, STN color, or TFT



- STN 4 or 8-bit interface mode
- STN dual or single panel mode
- Little-endian, big-endian, or Windows CE mode
- Interrupt generation event

### 12.4.2 Hardware cursor support

The hardware cursor feature reduces software overhead associated with maintaining a cursor image in the LCD frame buffer.

Without this feature, software needed to:

- Save an image of the area under the next cursor position.
- Update the area with the cursor image.
- Repair the last cursor position with a previously saved image.

In addition, the LCD driver had to check whether the graphics operation had overwritten the cursor, and correct it. With a cursor size of 64x64 and 24-bit color, each cursor move involved reading and writing approximately 75KB of data.

The hardware cursor removes the requirement for this management by providing a completely separate image buffer for the cursor, and superimposing the cursor image on the LCD output stream at the current cursor (X,Y) coordinate.

To move the hardware cursor, the software driver supplies a new cursor coordinate. The frame buffer requires no modification. This significantly reduces software overhead.

The cursor image is held in the LCD controller in an internal 256x32-bit buffer memory.

### 12.4.3 Types of LCD panels supported

The LCD controller supports the following types of LCD panel:

- Active matrix TFT panels with up to 24-bit bus interface.
- Single-panel monochrome STN panels (4-bit and 8-bit bus interface).
- Dual-panel monochrome STN panels (4-bit and 8-bit bus interface per panel).
- Single-panel color STN panels, 8-bit bus interface.
- Dual-panel color STN panels, 8-bit bus interface per panel.

### 12.4.4 TFT panels

TFT panels support one or more of the following color modes:

- 1 bpp, palettized, 2 colors selected from available colors.
- 2 bpp, palettized, 4 colors selected from available colors.
- 4 bpp, palettized, 16 colors selected from available colors.
- 8 bpp, palettized, 256 colors selected from available colors.
- 12 bpp, direct 4:4:4 RGB.

- 16 bpp, direct 5:5:5 RGB, with 1 bpp not normally used. This pixel is still output, and can be used as a brightness bit to connect to the Least Significant Bit (LSB) of RGB components of a 6:6:6 TFT panel.
- 16 bpp, direct 5:6:5 RGB.
- 24 bpp, direct 8:8:8 RGB, providing over 16 million colors.

Each 16-bit palette entry is composed of 5 bpp (RGB), plus a common intensity bit. This provides better memory utilization and performance compared with a full 6 bpp structure. The total number of colors supported can be doubled from 32K to 64K if the intensity bit is used and applied to all three color components simultaneously.

Alternatively, the 16 signals can be used to drive a 5:6:5 panel with the extra bit only applied to the green channel.

#### 12.4.5 Color STN panels

Color STN panels support one or more of the following color modes:

- 1 bpp, palettized, 2 colors selected from 3375.
- 2 bpp, palettized, 4 colors selected from 3375.
- 4 bpp, palettized, 16 colors selected from 3375.
- 8 bpp, palettized, 256 colors selected from 3375.
- 16 bpp, direct 4:4:4 RGB, with 4 bpp not being used.

#### 12.4.6 Monochrome STN panels

Monochrome STN panels support one or more of the following modes:

- 1 bpp, palettized, 2 gray scales selected from 15.
- 2 bpp, palettized, 4 gray scales selected from 15.
- 4 bpp, palettized, 16 gray scales selected from 15.

More than 4 bpp for monochrome panels can be programmed, but using these modes has no benefit because the maximum number of gray scales supported on the display is 15.

## 12.5 Pin description

The largest configuration for the LCD controller uses 31 pins. There are many variants using as few as 10 pins for a monochrome STN panel. Pins are allocated in groups based on the selected configuration. All LCD functions are shared with other chip functions. In [Table 203](#), only the LCD related portion of the pin name is shown.

**Remark:** To connect the LCD controller to necessary pins, see [Section 8.3](#).

**Table 203. LCD controller pins**

Pin name	Type	Function
LCD_PWR	output	LCD panel power enable.
LCD_DCLK	output	LCD panel clock.
LCD_ENAB_M	output	STN AC bias drive or TFT data enable output.
LCD_FP	output	Frame pulse (STN). Vertical synchronization pulse (TFT)
LCD_LE	output	Line end signal
LCD_LP	output	Line synchronization pulse (STN). Horizontal synchronization pulse (TFT)
LCD_VD[23:0]	output	LCD panel data. Bits used depend on the panel configuration.
LCD_CLKIN	input	Optional clock input.

### 12.5.1 Signal usage

The signals that are used for various display types are identified in the following sections.

#### 12.5.1.1 Signals used for single panel STN displays

The signals used for single panel STN displays are shown in [Table 204](#). UD refers to upper panel data.

**Table 204. Pins used for single panel STN displays**

Pin name	4-bit Monochrome (10 pins)	8-bit Monochrome (14 pins)	Color (14 pins)
LCD_PWR	Y	Y	Y
LCD_DCLK	Y	Y	Y
LCD_ENAB_M	Y	Y	Y
LCD_FP	Y	Y	Y
LCD_LE	Y	Y	Y
LCD_LP	Y	Y	Y
LCD_VD[3:0]	UD[3:0]	UD[3:0]	UD[3:0]
LCD_VD[7:4]	-	UD[7:4]	UD[7:4]
LCD_VD[23:8]	-	-	-

#### 12.5.1.2 Signals used for dual panel STN displays

The signals used for dual panel STN displays are shown in [Table 205](#). UD refers to upper panel data, and LD refers to lower panel data.

**Table 205. Pins used for dual panel STN displays**

Pin name	4-bit Monochrome (14 pins)	8-bit Monochrome (22 pins)	Color (22 pins)
LCD_PWR	Y	Y	Y
LCD_DCLK	Y	Y	Y
LCD_ENAB_M	Y	Y	Y
LCD_FP	Y	Y	Y
LCD_LE	Y	Y	Y
LCD_LP	Y	Y	Y
LCD_VD[3:0]	UD[3:0]	UD[3:0]	UD[3:0]
LCD_VD[7:4]	-	UD[7:4]	UD[7:4]
LCD_VD[11:8]	LD[3:0]	LD[3:0]	LD[3:0]
LCD_VD[15:12]	-	LD[7:4]	LD[7:4]
LCD_VD[23:16]	-	-	-

### 12.5.1.3 Signals used for TFT displays

The signals used for TFT displays are shown in [Table 206](#).

**Table 206. Pins used for TFT displays**

Pin name	12-bit, 4:4:4 mode (18 pins)	16-bit, 5:6:5 mode (22 pins)	16-bit, 1:5:5:5 mode (24 pins)	24-bit (30 pins)
LCD_PWR	Y	Y	Y	Y
LCD_DCLK	Y	Y	Y	Y
LCD_ENAB_M	Y	Y	Y	Y
LCD_FP	Y	Y	Y	Y
LCD_LE	Y	Y	Y	Y
LCD_LP	Y	Y	Y	Y
LCD_VD[1:0]	-	-	-	RED[1:0]
LCD_VD[2]	-	-	Intensity	RED[2]
LCD_VD[3]	-	RED[0]	RED[0]	RED[3]
LCD_VD[7:4]	RED[3:0]	RED[4:1]	RED[4:1]	RED[7:4]
LCD_VD[9:8]	-	-	-	GREEN[1:0]
LCD_VD[10]	-	GREEN[0]	Intensity	GREEN[2]
LCD_VD[11]	-	GREEN[1]	GREEN[0]	GREEN[3]
LCD_VD[15:12]	GREEN[3:0]	GREEN[5:2]	GREEN[4:1]	GREEN[7:4]
LCD_VD[17:16]	-	-	-	BLUE[1:0]
LCD_VD[18]	-	-	Intensity	BLUE[2]
LCD_VD[19]	-	BLUE[0]	BLUE[0]	BLUE[3]
LCD_VD[23:20]	BLUE[3:0]	BLUE[4:1]	BLUE[4:1]	BLUE[7:4]

## 12.6 LCD controller functional description

The LCD controller performs translation of pixel-coded data into the required formats and timings to drive a variety of single or dual panel monochrome and color LCDs.

Packets of pixel coded data are fed using the AHB interface, to two independent, programmable, 32-bit wide, DMA FIFOs that act as input data flow buffers.

The buffered pixel coded data is then unpacked using a pixel serializer.

Depending on the LCD type and mode, the unpacked data can represent:

- An actual true display gray or color value.
- An address to a 256x16 bit wide palette RAM gray or color value.

In the case of STN displays, either a value obtained from the addressed palette location, or the true value is passed to the gray scaling generators. The hardware-coded gray scale algorithm logic sequences the activity of the addressed pixels over a programmed number of frames to provide the effective display appearance.

For TFT displays, either an addressed palette value or true color value is passed directly to the output display drivers, bypassing the gray scaling algorithmic logic.

In addition to data formatting, the LCD controller provides a set of programmable display control signals, including:

- LCD panel power enable.
- Pixel clock.
- Horizontal and vertical synchronization pulses.
- Display bias.

The LCD controller generates individual interrupts for:

- Upper or lower panel DMA FIFO underflow.
- Base address update signification.
- Vertical compare.
- Bus error.

There is also a single combined interrupt that is asserted when any of the individual interrupts become active.

[Figure 30](#) shows a simplified block diagram of the LCD controller.

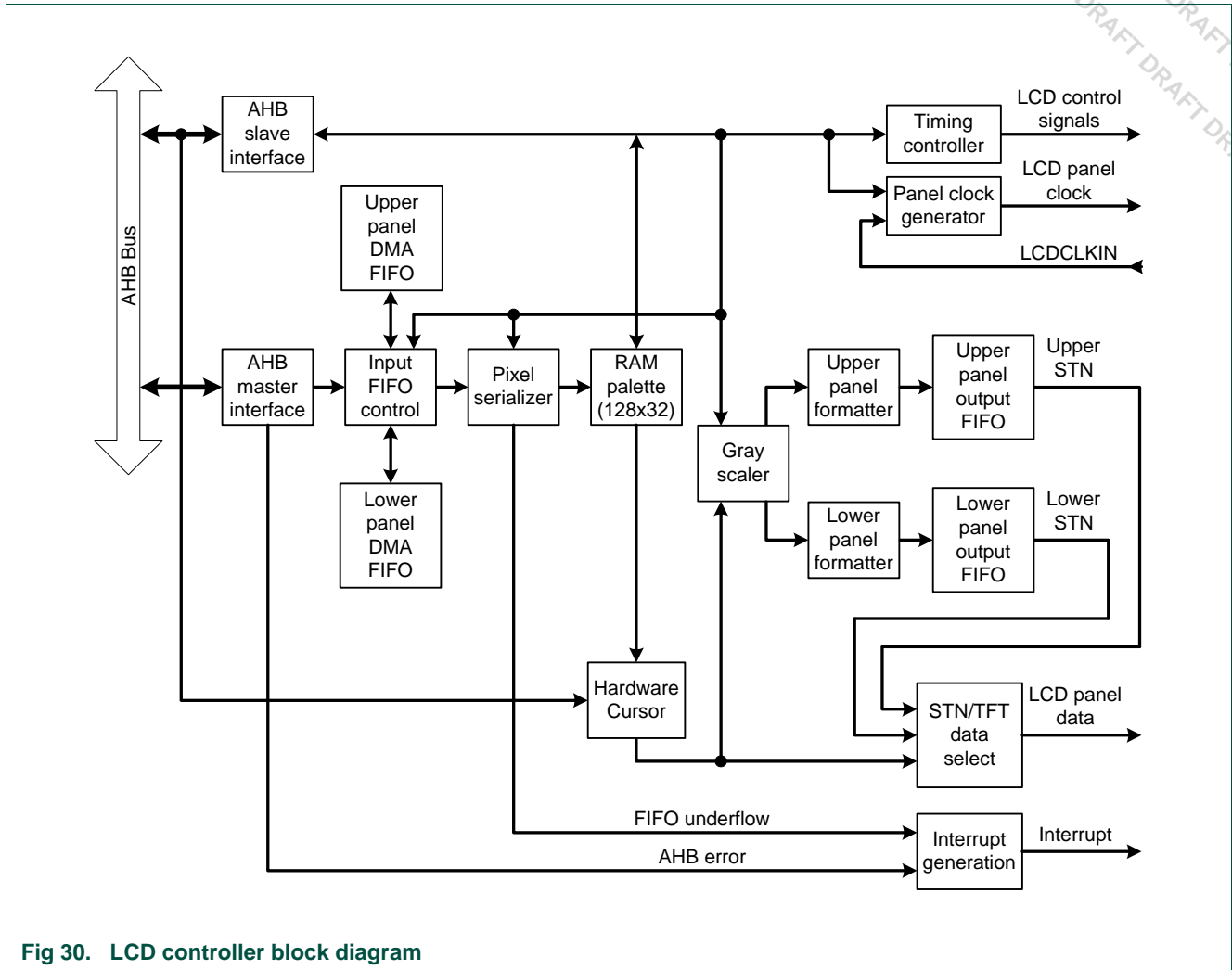


Fig 30. LCD controller block diagram

### 12.6.1 AHB interfaces

The LCD controller includes two separate AHB interfaces. The first, an AHB slave interface, is used primarily by the CPU to access control and data registers within the LCD controller. The second, an AHB master interface, is used by the LCD controller for DMA access to display data stored in memory elsewhere in the system. The LCD DMA controller can only access the 10 kB SRAM on AHB1 and the external memory.

#### 12.6.1.1 AMBA AHB slave interface

The AHB slave interface connects the LCD controller to the AHB bus and provides CPU accesses to the registers and palette RAM.

#### 12.6.1.2 AMBA AHB master interface

The AHB master interface transfers display data from a selected slave (memory) to the LCD controller DMA FIFOs. It can be configured to obtain data from the 16 kB, on-chip SRAM on AHB1, various types of off-chip static memory, or off-chip SDRAM.

In dual panel mode, the DMA FIFOs are filled up in an alternating fashion via a single DMA request. In single panel mode, the DMA FIFOs are filled up in a sequential fashion from a single DMA request.

The inherent AHB master interface state machine performs the following functions:

- Loads the upper panel base address into the AHB address incremter on recognition of a new frame.
- Monitors both the upper and lower DMA FIFO levels and asserts a DMA request to request display data from memory, filling them to above the programmed watermark. the DMA request is reasserted when there are at least four locations available in either FIFO (dual panel mode).
- Checks for 1KB boundaries during fixed-length bursts, appropriately adjusting the address in such occurrences.
- Generates the address sequences for fixed-length and undefined bursts.
- Controls the handshaking between the memory and DMA FIFOs. It inserts busy cycles if the FIFOs have not completed their synchronization and updating sequence.
- Fills up the DMA FIFOs, in dual panel mode, in an alternating fashion from a single DMA request.
- Asserts the a bus error interrupt if an error occurs during an active burst.
- Responds to retry commands by restarting the failed access. This introduces some busy cycles while it re-synchronizes.

### 12.6.2 Dual DMA FIFOs and associated control logic

The pixel data accessed from memory is buffered by two DMA FIFOs that can be independently controlled to cover single and dual-panel LCD types. Each FIFO is 16 words deep by 64 bits wide and can be cascaded to form an effective 32-Dword deep FIFO in single panel mode.

Synchronization logic transfers the pixel data from the AHB clock domain to the LCD controller clock domain. The water level marks in each FIFO are set such that each FIFO requests data when at least four locations become available.

An interrupt signal is asserted if an attempt is made to read either of the two DMA FIFOs when they are empty (an underflow condition has occurred).

### 12.6.3 Pixel serializer

This block reads the 32-bit wide LCD data from the output port of the DMA FIFO and extracts 24, 16, 8, 4, 2, or 1 bpp data, depending on the current mode of operation. The LCD controller supports big-endian, little-endian, and Windows CE data formats.

Depending on the mode of operation, the extracted data can be used to point to a color or gray scale value in the palette RAM or can actually be a true color value that can be directly applied to an LCD panel input.

[Table 207](#) through [Table 209](#) show the structure of the data in each DMA FIFO word corresponding to the endianness and bpp combinations. For each of the three supported data formats, the required data for each panel display pixel must be extracted from the data word.

Table 207. FIFO bits for Little-endian Byte, Little-endian Pixel order

FIFO bit	1 bpp	2 bpp	4 bpp	8 bpp	16 bpp	24 bpp
0	p0	p0	p0	p0	p0	p0
1	p1					
2	p2	p1				
3	p3					
4	p4	p2	p1			
5	p5					
6	p6	p3	p1			
7	p7					
8	p8	p4	p2	p1		
9	p9					
10	p10	p5	p3	p1		
11	p11					
12	p12	p6	p3	p1		
13	p13					
14	p14	p7	p4	p2		
15	p15					
16	p16	p8	p4	p2		
17	p17					
18	p18	p9	p5	p1		
19	p19					
20	p20	p10	p5	p1		
21	p21					
22	p22	p11	p6	p3		
23	p23					
24	p24	p12	p6	p3		
25	p25					
26	p26	p13	p7	p1		
27	p27					
28	p28	p14	p7	p1		
29	p29					
30	p30	p15	p7	p1		
31	p31					



Table 208. FIFO bits for Big-endian Byte, Big-endian Pixel order

FIFO bit	1 bpp	2 bpp	4 bpp	8 bpp	16 bpp	24 bpp	
0	p31	p15	p7	p3	p1	p0	
1	p30						
2	p29	p14					
3	p28						
4	p27	p13	p6				
5	p26						
6	p25	p12					
7	p24						
8	p23	p11	p5	p2			
9	p22						
10	p21	p10					
11	p20						
12	p19	p9					p4
13	p18						
14	p17	p8					
15	p16						
16	p15	p7	p3		p1		
17	p14						
18	p13	p6					
19	p12						
20	p11	p5					p2
21	p10						
22	p9	p4					
23	p8						
24	p7	p3	p1	p0			
25	p6						
26	p5	p2					
27	p4						
28	p3	p1				p0	
29	p2						
30	p1	p0					
31	p0						

Table 209. FIFO bits for Little-endian Byte, Big-endian Pixel order

FIFO bit	1 bpp	2 bpp	4 bpp	8 bpp	16 bpp	24 bpp	
0	p7	p3	p1	p0	p0	p0	
1	p6						
2	p5	p2					
3	p4						
4	p3	p1	p0				
5	p2						
6	p1	p0					
7	p0						
8	p15	p7	p3	p1			
9	p14						
10	p13	p6					
11	p12						
12	p11	p5					p2
13	p10						
14	p9	p4					
15	p8						
16	p23	p11	p5		p2		
17	p22						
18	p21	p10					
19	p20						
20	p19	p9					p4
21	p18						
22	p17	p8					
23	p16						
24	p31	p15	p7	p1			
25	p30						
26	p29	p14					
27	p28						
28	p27	p13					p6
29	p26						
30	p25	p12					
31	p24						

Table 210 shows the structure of the data in each DMA FIFO word in RGB mode.

Table 210. RGB mode data formats

FIFO data	24-bit RGB	16-bit (1:5:5:5 RGB)	16-bit (5:6:5 RGB)	16-bit (4:4:4 RGB)
0	p0, Red 0	p0, Red 0	p0, Red 0	p0, Red 0
1	p0, Red 1	p0, Red 1	p0, Red 1	p0, Red 1
2	p0, Red 2	p0, Red 2	p0, Red 2	p0, Red 2
3	p0, Red 3	p0, Red 3	p0, Red 3	p0, Red 3
4	p0, Red 4	p0, Red 4	p0, Red 4	p0, Green 0
5	p0, Red 5	p0, Green 0	p0, Green 0	p0, Green 1
6	p0, Red 6	p0, Green 1	p0, Green 1	p0, Green 2
7	p0, Red 7	p0, Green 2	p0, Green 2	p0, Green 3
8	p0, Green 0	p0, Green 3	p0, Green 3	p0, Blue 0
9	p0, Green 1	p0, Green 4	p0, Green 4	p0, Blue 1
10	p0, Green 2	p0, Blue 0	p0, Green 5	p0, Blue 2
11	p0, Green 3	p0, Blue 1	p0, Blue 0	p0, Blue 3
12	p0, Green 4	p0, Blue 2	p0, Blue 1	-
13	p0, Green 5	p0, Blue 3	p0, Blue 2	-
14	p0, Green 6	p0, Blue 4	p0, Blue 3	-
15	p0, Green 7	p0 intensity bit	p0, Blue 4	-
16	p0, Blue 0	p1, Red 0	p1, Red 0	p1, Red 0
17	p0, Blue 1	p1, Red 1	p1, Red 1	p1, Red 1
18	p0, Blue 2	p1, Red 2	p1, Red 2	p1, Red 2
19	p0, Blue 3	p1, Red 3	p1, Red 3	p1, Red 3
20	p0, Blue 4	p1, Red 4	p1, Red 4	p1, Green 0
21	p0, Blue 5	p1, Green 0	p1, Green 0	p1, Green 1
22	p0, Blue 6	p1, Green 1	p1, Green 1	p1, Green 2
23	p0, Blue 7	p1, Green 2	p1, Green 2	p1, Green 3
24	-	p1, Green 3	p1, Green 3	p1, Blue 0
25	-	p1, Green 4	p1, Green 4	p1, Blue 1
26	-	p1, Blue 0	p1, Green 5	p1, Blue 2
27	-	p1, Blue 1	p1, Blue 0	p1, Blue 3
28	-	p1, Blue 2	p1, Blue 1	-
29	-	p1, Blue 3	p1, Blue 2	-
30	-	p1, Blue 4	p1, Blue 3	-
31	-	p1 intensity bit	p1, Blue 4	-

### 12.6.4 RAM palette

The RAM-based palette is a 256 x 16 bit dual-port RAM physically structured as 128 x 32 bits. Two entries can be written into the palette from a single word write access. The Least Significant Bit (LSB) of the serialized pixel data selects between upper and lower halves of the palette RAM. The half that is selected depends on the byte ordering mode. In little-endian mode, setting the LSB selects the upper half, but in big-endian mode, the lower half of the palette is selected.

Pixel data values can be written and verified through the AHB slave interface. For information on the supported colors, refer to the section on the related panel type earlier in this chapter.

The palette RAM is a dual port RAM with independent controls and addresses for each port. Port1 is used as a read/write port and is connected to the AHB slave interface. The palette entries can be written and verified through this port. Port2 is used as a read-only port and is connected to the unpacker and gray scaler. For color modes of less than 16 bpp, the palette enables each pixel value to be mapped to a 16-bit color:

- For TFT displays, the 16-bit value is passed directly to the pixel serializer.
- For STN displays, the 16-bit value is first converted by the gray scaler.

[Table 211](#) shows the bit representation of the palette data. The palette 16-bit output uses the TFT 1:5:5:5 data format. In 16 and 24 bpp TFT mode, the palette is bypassed and the output of the pixel serializer is used as the TFT panel data.

**Table 211. Palette data storage for TFT modes.**

Bit(s)	Name (RGB format)	Description (RGB format)	Name (BGR format)	Description (BGR format)
4:0	R[4:0]	Red palette data	B[4:0]	Blue palette data
9:5	G[4:0]	Green palette data	G[4:0]	Green palette data
14:10	B[4:0]	Blue palette data	R[4:0]	Red palette data
15	I	Intensity / unused	I	Intensity / unused
20:16	R[4:0]	Red palette data	B[4:0]	Blue palette data
25:21	G[4:0]	Green palette data	G[4:0]	Green palette data
30:26	B[4:0]	Blue palette data	R[4:0]	Red palette data
31	I	Intensity / unused	I	Intensity / unused

The red and blue pixel data can be swapped to support BGR data format using a control register bit (bit 8 = BGR). See the LCD\_CTRL register description for more information. [Table 212](#) shows the bit representation of the palette data for the STN color modes.

**Table 212. Palette data storage for STN color modes.**

Bit(s)	Name (RGB format)	Description (RGB format)	Name (BGR format)	Description (BGR format)
0	R[0]	Unused	B[0]	Unused
4:1	R[4:1]	Red palette data	B[4:1]	Blue palette data
5	G[0]	Unused	G[0]	Unused
9:6	G[4:1]	Green palette data	G[4:1]	Green palette data
10	B[0]	Unused	R[0]	Unused
14:11	B[4:1]	Blue palette data	R[4:1]	Red palette data
15	I	Unused	I	Unused
16	-	Unused	-	Unused
20:17	R[3:0]	Red palette data	B[3:0]	Blue palette data
21	-	Unused	-	Unused
25:22	G[3:0]	Green palette data	G[3:0]	Green palette data

**Table 212. Palette data storage for STN color modes.**

Bit(s)	Name (RGB format)	Description (RGB format)	Name (BGR format)	Description (BGR format)
26	-	Unused	-	Unused
30:27	B[3:0]	Blue palette data	R[3:0]	Red palette data
31	-	Unused	-	Unused

For monochrome STN mode, only the red palette field bits [4:1] are used. However, in STN color mode the green and blue [4:1] are also used. Only 4 bits per color are used, because the gray scaler only supports 16 different shades per color.

[Table 213](#) shows the bit representation of the palette data for the STN monochrome mode.

**Table 213. Palette data storage for STN monochrome mode.**

Bit(s)	Name	Description
0	-	Unused
4:1	Y[3:0]	Intensity data
16:5	-	Unused
20:17	Y[3:0]	Intensity data
31:21	-	Unused

## 12.6.5 Hardware cursor

The hardware cursor is an integral part of the LCD controller. It uses the LCD timing module to provide an indication of the current scan position coordinate, and intercepts the pixel stream between the palette logic and the gray scale/output multiplexer.

All cursor programming registers are accessed through the LCD slave interface. This also provides a read/write port to the cursor image RAM.

### 12.6.5.1 Cursor operation

The hardware cursor is contained in a dual port RAM. It is programmed by software through the AHB slave interface. The AHB slave interface also provides access to the hardware cursor control registers. These registers enable you to modify the cursor position and perform various other functions.

When enabled, the hardware cursor uses the horizontal and vertical synchronization signals, along with a pixel clock enable and various display parameters to calculate the current scan coordinate.

When the display point is inside the bounds of the cursor image, the cursor replaces frame buffer pixels with cursor pixels.

When the last cursor pixel is displayed, an interrupt is generated that software can use as an indication that it is safe to modify the cursor image. This enables software controlled animations to be performed without flickering for frame synchronized cursors.

### 12.6.5.2 Cursor sizes

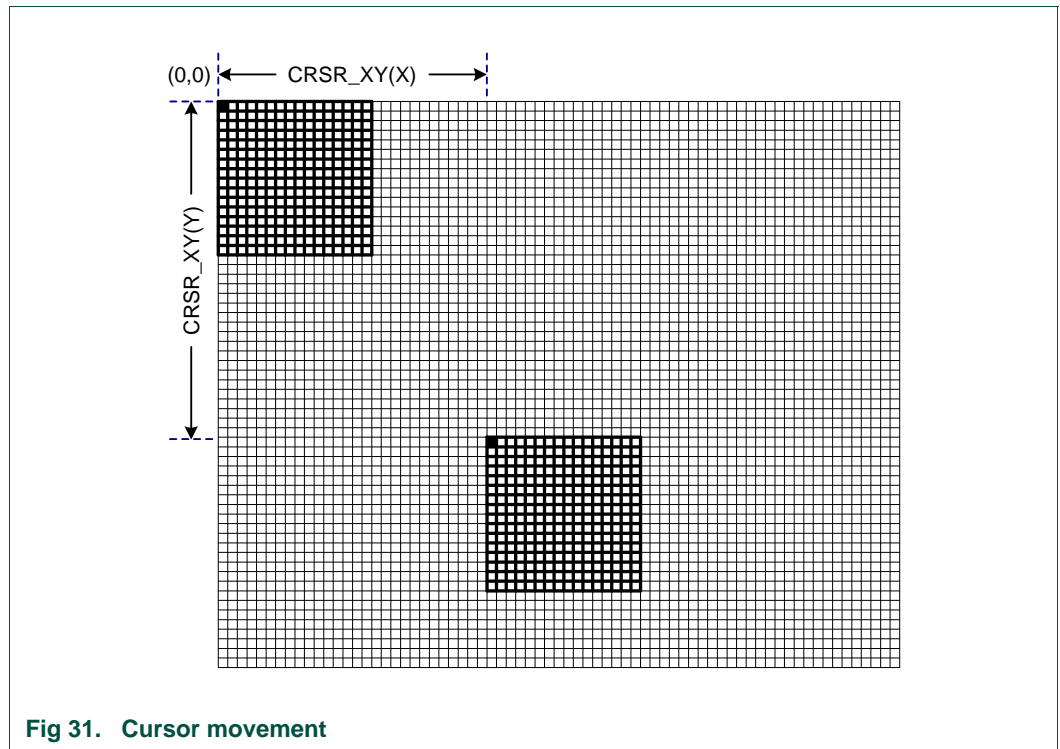
Two cursor sizes are supported, as shown in [Table 214](#).

**Table 214. Palette data storage for STN monochrome mode.**

X Pixels	Y Pixels	Bits per pixel	Words per line	Words in cursor image
32	32	2	2	64
64	64	2	4	256

### 12.6.5.3 Cursor movement

The following descriptions assume that both the screen and cursor origins are at the top left of the visible screen (the first visible pixel scanned each frame). [Figure 31](#) shows how each pixel coordinate is assumed to be the top left corner of the pixel.



**Fig 31. Cursor movement**

### 12.6.5.4 Cursor XY positioning

The  $CRSR\_XY$  register controls the cursor position on the cursor overlay (see Cursor XY Position register). This provides separate fields for X and Y ordinates.

The  $CRSR\_CFG$  register (see Cursor Configuration register) provides a FrameSync bit controlling the visible behavior of the cursor.

With FrameSync inactive, the cursor responds immediately to any change in the programmed  $CRSR\_XY$  value. Some transient smearing effects may be visible if the cursor is moved across the LCD scan line.

With FrameSync active, the cursor only updates its position after a vertical synchronization has occurred. This provides clean cursor movement, but the cursor position only updates once a frame.

12.6.5.5 Cursor clipping

The CRSR\_XY register (see Cursor XY Position register) is programmed with positive binary values that enable the cursor image to be located anywhere on the visible screen image. The cursor image is clipped automatically at the screen limits when it extends beyond the screen image to the right or bottom (see X1,Y1 in Figure 32). The checked pattern shows the visible portion of the cursor.

Because the CRSR\_XY register values are positive integers, to emulate cursor clipping on the left and top of screen, a Clip Position register, CRSR\_CLIP, is provided. This controls which point of the cursor image is positioned at the CRSR\_CLIP coordinate. For clipping functions on the Y axis, CRSR\_XY(X) is zero, and Clip(X) is programmed to provide the offset into the cursor image (X2 and X3). The equivalent function is provided to clip on the X axis at the top of the display (Y2).

For cursors that are not clipped at the X=0 or Y=0 lines, program the Clip Position register X and Y fields with zero to display the cursor correctly. See Clip(X4,Y4) for the effect of incorrect programming.

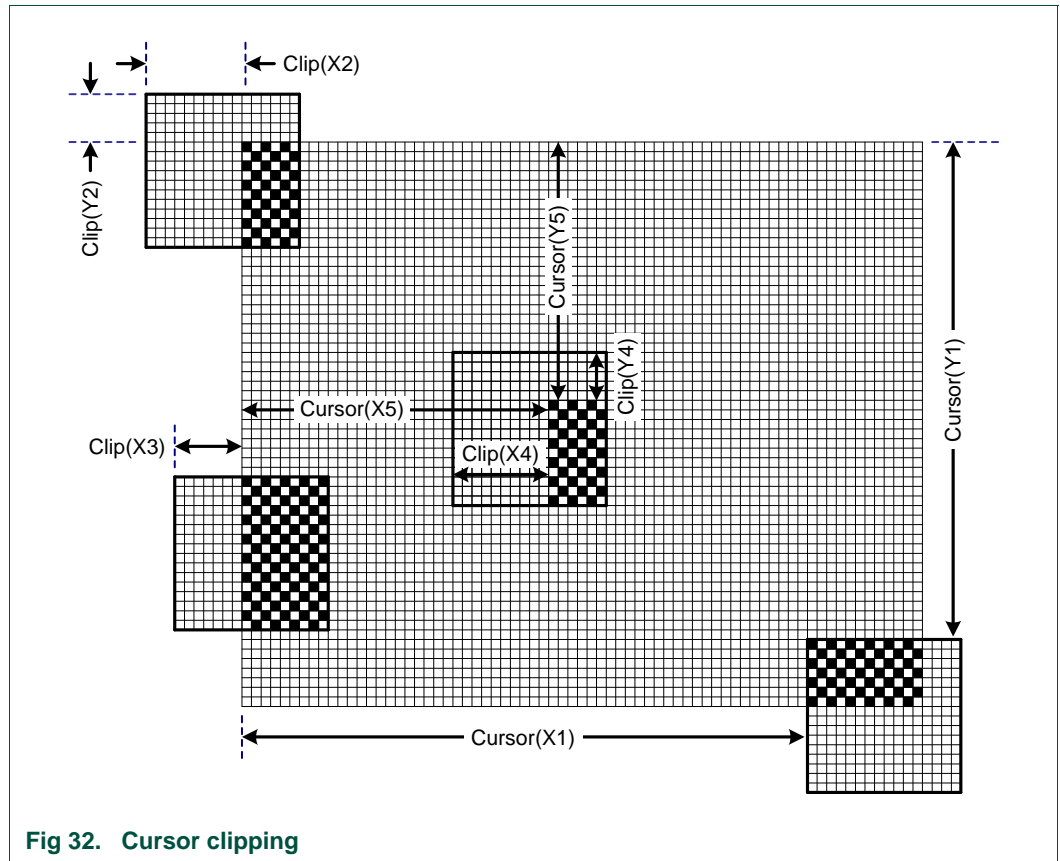


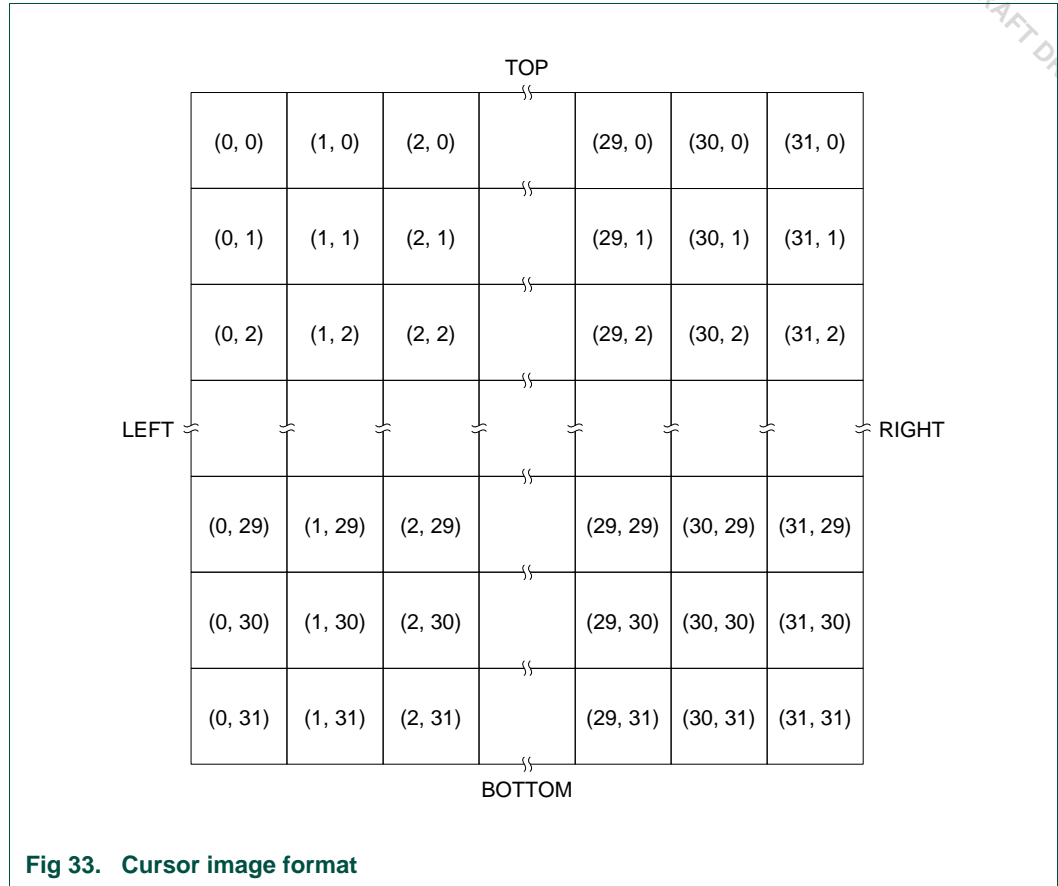
Fig 32. Cursor clipping

12.6.5.6 Cursor image format

The LCD frame buffer supports three packing formats, but the hardware cursor image requirement has been simplified to support only LBBP. This is little-endian byte, big-endian pixel for Windows CE mode.

The Image RAM start address is offset by 0x800 from the LCD base address, as shown in the register description in this chapter.

The displayed cursor coordinate system is expressed in terms of (X,Y). 64 x 64 is an extension of the 32 x 32 format shown in [Figure 33](#).



**32 by 32 pixel format**

Four cursors are held in memory, each with the same pixel format. [Table 215](#) lists the base addresses for the four cursors.

**Table 215. Addresses for 32 x 32 cursors**

Address	Description
0x2008 8800	Cursor 0 start address.
0x2008 8900	Cursor 1 start address.
0x2008 8A00	Cursor 2 start address.
0x2008 8B00	Cursor 3 start address.

[Table 216](#) shows the buffer to pixel mapping for Cursor 0.



**Table 216. Buffer to pixel mapping for 32 x 32 pixel cursor format**

Data bits	Offset into cursor memory					
	0	4	(8 * y)	(8 * y) +4	F8	FC
1:0	(3, 0)	(19, 0)	(3, y)	(19, y)	(3, 31)	(19, 31)
3:2	(2, 0)	(18, 0)	(2, y)	(18, y)	(2, 31)	(18, 31)
5:4	(1, 0)	(17, 0)	(1, y)	(17, y)	(1, 31)	(17, 31)
7:6	(0, 0)	(16, 0)	(0, y)	(16, y)	(0, 31)	(16, 31)
9:8	(7, 0)	(23, 0)	(7, y)	(23, y)	(7, 31)	(23, 31)
11:10	(6, 0)	(22, 0)	(6, y)	(22, y)	(6, 31)	(22, 31)
13:12	(5, 0)	(21, 0)	(5, y)	(21, y)	(5, 31)	(21, 31)
15:14	(4, 0)	(20, 0)	(4, y)	(20, y)	(4, 31)	(20, 31)
17:16	(11, 0)	(27, 0)	(11, y)	(27, y)	(11, 31)	(27, 31)
19:18	(10, 0)	(26, 0)	(10, y)	(26, y)	(10, 31)	(26, 31)
21:20	(9, 0)	(25, 0)	(9, y)	(25, y)	(9, 31)	(25, 31)
23:22	(8, 0)	(24, 0)	(8, y)	(24, y)	(8, 31)	(24, 31)
25:24	(15, 0)	(31, 0)	(15, y)	(31, y)	(15, 31)	(31, 31)
27:26	(14, 0)	(30, 0)	(14, y)	(30, y)	(14, 31)	(30, 31)
29:28	(13, 0)	(29, 0)	(13, y)	(29, y)	(13, 31)	(29, 31)
31:30	(12, 0)	(28, 0)	(12, y)	(28, y)	(12, 31)	(28, 31)

**64 by 64 pixel format**

Only one cursor fits in the memory space in 64 x 64 mode, as detailed in [Table 217](#).

**Table 217. Buffer to pixel mapping for 64 x 64 pixel cursor format**

Data bits	Offset into cursor memory								
	0	4	8	12	(16 * y)	(16 * y) +4	(16 * y) + 8	(16 * y) + 12	FC
1:0	(3, 0)	(19, 0)	(35, 0)	(51, 0)	(3, y)	(19, y)	(35, y)	(51, y)	(51, 63)
3:2	(2, 0)	(18, 0)	(34, 0)	(50, 0)	(2, y)	(18, y)	(34, y)	(50, y)	(50, 63)
5:4	(1, 0)	(17, 0)	(33, 0)	(49, 0)	(1, y)	(17, y)	(33, y)	(49, y)	(49, 63)
7:6	(0, 0)	(16, 0)	(32, 0)	(48, 0)	(0, y)	(16, y)	(32, y)	(48, y)	(48, 63)
9:8	(7, 0)	(23, 0)	(39, 0)	(55, 0)	(7, y)	(23, y)	(39, y)	(55, y)	(55, 63)
11:10	(6, 0)	(22, 0)	(38, 0)	(54, 0)	(6, y)	(22, y)	(38, y)	(54, y)	(54, 63)
13:12	(5, 0)	(21, 0)	(37, 0)	(53, 0)	(5, y)	(21, y)	(37, y)	(53, y)	(53, 63)
15:14	(4, 0)	(20, 0)	(36, 0)	(52, 0)	(4, y)	(20, y)	(36, y)	(52, y)	(52, 63)
17:16	(11, 0)	(27, 0)	(43, 0)	(59, 0)	(11, y)	(27, y)	(43, y)	(59, y)	(59, 63)
19:18	(10, 0)	(26, 0)	(42, 0)	(58, 0)	(10, y)	(26, y)	(42, y)	(58, y)	(58, 63)
21:20	(9, 0)	(25, 0)	(41, 0)	(57, 0)	(9, y)	(25, y)	(41, y)	(57, y)	(57, 63)
23:22	(8, 0)	(24, 0)	(40, 0)	(56, 0)	(8, y)	(24, y)	(40, y)	(56, y)	(56, 63)
25:24	(15, 0)	(31, 0)	(47, 0)	(63, 0)	(15, y)	(31, y)	(47, y)	(63, y)	(63, 63)
27:26	(14, 0)	(30, 0)	(46, 0)	(62, 0)	(14, y)	(30, y)	(46, y)	(62, y)	(62, 63)
29:28	(13, 0)	(29, 0)	(45, 0)	(61, 0)	(13, y)	(29, y)	(45, y)	(61, y)	(61, 63)
31:30	(12, 0)	(28, 0)	(44, 0)	(60, 0)	(12, y)	(28, y)	(44, y)	(60, y)	(60, 63)

**Cursor pixel encoding**

Each pixel of the cursor requires two bits of information. These are interpreted as Color0, Color1, Transparent, and Transparent inverted.

In the coding scheme, bit 1 selects between color and transparent (AND mask) and bit 0 selects variant (XOR mask).

[Table 218](#) shows the pixel encoding bit assignments.

**Table 218. Pixel encoding**

Value	Description
00	Color0. The cursor color is displayed according to the Red-Green-Blue (RGB) value programmed into the CRSR_PAL0 register.
01	Color1. The cursor color is displayed according to the RGB value programmed into the CRSR_PAL1 register.
10	Transparent. The cursor pixel is transparent, so is displayed unchanged. This enables the visible cursor to assume shapes that are not square.
11	Transparent inverted. The cursor pixel assumes the complementary color of the frame pixel that is displayed. This can be used to ensure that the cursor is visible regardless of the color of the frame buffer image.

**12.6.6 Gray scaler**

A patented gray scale algorithm drives monochrome and color STN panels. This provides 15 gray scales for monochrome displays. For STN color displays, the three color components (RGB) are gray scaled simultaneously. This results in 3375 (15x15x15) colors being available. The gray scaler transforms each 4-bit gray value into a sequence of activity-per-pixel over several frames, relying to some degree on the display characteristics, to give the representation of gray scales and color.

**12.6.7 Upper and lower panel formatters**

Formatters are used in STN mode to convert the gray scaler output to a parallel format as required by the display. For monochrome displays, this is either 4 or 8 bits wide, and for color displays, it is 8 bits wide. [Table 219](#) shows a color display driven with 2 2/3 pixels worth of data in a repeating sequence.

**Table 219. Color display driven with 2 2/3 pixel data**

Byte	CLD[7]	CLD[6]	CLD[5]	CLD[4]	CLD[3]	CLD[2]	CLD[1]	CLD[0]
0	P2[Green]	P2[Red]	P1[Blue]	P1[Green]	P1[Red]	P0[Blue]	P0[Green]	P0[Red]
1	P5[Red]	P4q[Blue]	P4[Green]	P4[Red]	P3[Blue]	P3[Green]	P3[Red]	P2[Blue]
2	P7[Blue]	P7[Green]	P7[Red]	P6[Blue]	P6[Green]	P6[Red]	P5[Blue]	P5[Green]

Each formatter consists of three 3-bit (RGB) shift left registers. RGB pixel data bit values from the gray scaler are concurrently shifted into the respective registers. When enough data is available, a byte is constructed by multiplexing the registered data to the correct bit position to satisfy the RGB data pattern of LCD panel. The byte is transferred to the 3-byte FIFO, which has enough space to store eight color pixels.

### 12.6.8 Panel clock generator

The output of the panel clock generator block is the panel clock, pin LCD\_DCLK. The panel clock can be based on either the peripheral clock for the LCD block or the external clock input for the LCD, pin LCD\_CLKIN. Whichever source is selected can be divided down in order to produce the internal LCD clock, LCDCLK.

The panel clock generator can be programmed to output the LCD panel clock in the range of LCDCLK/2 to LCDCLK/1025 to match the bpp data rate of the LCD panel being used.

The CLKSEL bit in the LCD\_POL register determines whether the base clock used is CCLK or the LCD\_CLKIN pin.

### 12.6.9 Timing controller

The primary function of the timing controller block is to generate the horizontal and vertical timing panel signals. It also provides the panel bias and enable signals. These timings are all register-programmable.

### 12.6.10 STN and TFT data select

Support is provided for passive Super Twisted Nematic (STN) and active Thin Film Transistor (TFT) LCD display types:

#### 12.6.10.1 STN displays

STN display panels require algorithmic pixel pattern generation to provide pseudo gray scaling on monochrome displays, or color creation on color displays.

#### 12.6.10.2 TFT displays

TFT display panels require the digital color value of each pixel to be applied to the display data inputs.

### 12.6.11 Interrupt generation

Four interrupts are generated by the LCD controller, and a single combined interrupt. The four interrupts are:

- Master bus error interrupt.
- Vertical compare interrupt.
- Next base address update interrupt.
- FIFO underflow interrupt.

Each of the four individual maskable interrupts is enabled or disabled by changing the mask bits in the LCD\_INT\_MSK register. These interrupts are also combined into a single overall interrupt, which is asserted if any of the individual interrupts are both asserted and unmasked. Provision of individual outputs in addition to a combined interrupt output enables use of either a global interrupt service routine, or modular device drivers to handle interrupts.

The status of the individual interrupt sources can be read from the LCD\_INTRAW register.

### 12.6.11.1 Master bus error interrupt

The master bus error interrupt is asserted when an ERROR response is received by the master interface during a transaction with a slave. When such an error is encountered, the master interface enters an error state and remains in this state until clearance of the error has been signaled to it. When the respective interrupt service routine is complete, the master bus error interrupt may be cleared by writing a 1 to the BERIC bit in the LCD\_INTCLR register. This action releases the master interface from its ERROR state to the start of FRAME state, and enables fresh frame of data display to be initiated.

### 12.6.11.2 Vertical compare interrupt

The vertical compare interrupt asserts when one of four vertical display regions, selected using the LCD\_CTRL register, is reached. The interrupt can be made to occur at the start of:

- Vertical synchronization.
- Back porch.
- Active video.
- Front porch.

The interrupt may be cleared by writing a 1 to the VcompIC bit in the LCD\_INTCLR register.

#### 12.6.11.2.1 Next base address update interrupt

The LCD next base address update interrupt asserts when either the LCDUPBASE or LCDLPBASE values have been transferred to the LCDUPCURR or LCDLPCURR incrementers respectively. This signals to the system that it is safe to update the LCDUPBASE or the LCDLPBASE registers with new frame base addresses if required.

The interrupt can be cleared by writing a 1 to the LNBUIC bit in the LCD\_INTCLR register.

#### 12.6.11.2.2 FIFO underflow interrupt

The FIFO underflow interrupt asserts when internal data is requested from an empty DMA FIFO. Internally, upper and lower panel DMA FIFO underflow interrupt signals are generated.

The interrupt can be cleared by writing a 1 to the FUFIC bit in the LCD\_INTCLR register.

## 12.6.12 LCD power-up and power-down sequence

The LCD controller requires the following power-up sequence to be performed:

1. When power is applied, the following signals are held LOW:

- LCD\_LP
- LCD\_DCLK
- LCD\_FP
- LCD\_ENAB\_M
- LCD\_VD[23:0]
- LCD\_LE

2. When LCD power is stabilized, a 1 is written to the LcdEn bit in the LCD\_CTRL register. This enables the following signals into their active states:

- LCD\_LP
- LCD\_DCLK
- LCD\_FP
- LCD\_ENAB\_M
- LCD\_LE

The LCD\_VD[23:0] signals remain in an inactive state.

3. When the signals in step 2 have stabilized, the contrast voltage (not controlled or supplied by the LCD controller) is applied to the LCD panel.

4. If required, a software or hardware timer can be used to provide the minimum display specific delay time between application of the control signals and power to the panel display. On completion of the time interval, power is applied to the panel by writing a 1 to the LcdPwr bit within the LCD\_CTRL register that, in turn, sets the LCD\_PWR signal high and enables the LCD\_VD[23:0] signals into their active states. The LCD\_PWR signal is intended to be used to gate the power to the LCD panel.

The power-down sequence is the reverse of the above four steps and must be strictly followed, this time, writing the respective register bits with 0.

Figure 34 shows the power-up and power-down sequences.

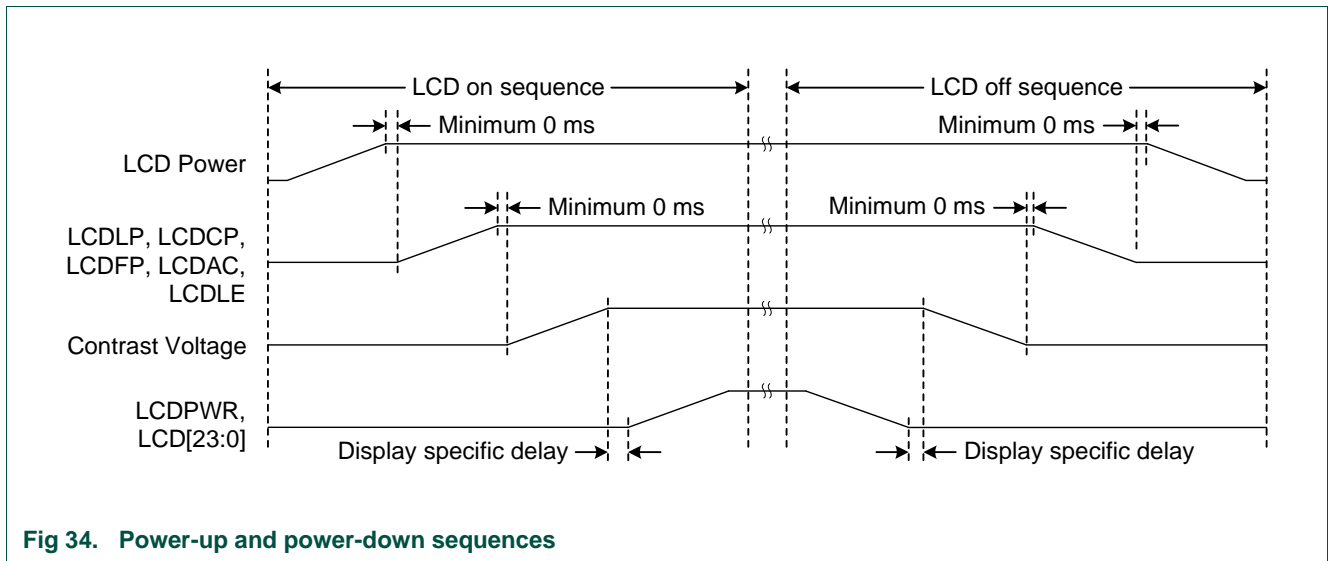


Fig 34. Power-up and power-down sequences

## 12.7 Register description

[Table 220](#) shows the registers associated with the LCD controller and a summary of their functions. Following the table are details for each register.

**Table 220. Summary of LCD controller registers**

Address	Name	Description	Reset value <a href="#">[1]</a>	Access	Table
0x400F C1B8	LCD_CFG	LCD Configuration and clocking control	0	R/W	<a href="#">221</a>
0x2008 8000	LCD_TIMH	Horizontal Timing Control register	0	R/W	<a href="#">222</a>
0x2008 8004	LCD_TIMV	Vertical Timing Control register	0	R/W	<a href="#">223</a>
0x2008 8008	LCD_POL	Clock and Signal Polarity Control register	0	R/W	<a href="#">224</a>
0x2008 800C	LCD_LE	Line End Control register	0	R/W	<a href="#">225</a>
0x2008 8010	LCD_UPBASE	Upper Panel Frame Base Address register	0	R/W	<a href="#">226</a>
0x2008 8014	LCD_LPBASE	Lower Panel Frame Base Address register	0	R/W	<a href="#">227</a>
0x2008 8018	LCD_CTRL	LCD Control register	0	R/W	<a href="#">228</a>
0x2008 801C	LCD_INTMSK	Interrupt Mask register	0	R/W	<a href="#">229</a>
0x2008 8020	LCD_INTRAW	Raw Interrupt Status register	0	RO	<a href="#">230</a>
0x2008 8024	LCD_INTSTAT	Masked Interrupt Status register	0	RO	<a href="#">231</a>
0x2008 8028	LCD_INTCLR	Interrupt Clear register	0	WO	<a href="#">232</a>
0x2008 802C	LCD_UPCURR	Upper Panel Current Address Value register	0	RO	<a href="#">233</a>
0x2008 8030	LCD_LPCURR	Lower Panel Current Address Value register	0	RO	<a href="#">234</a>
0x2008 8200 - 0x2008 83FC	LCD_PAL	256x16-bit Color Palette registers	0	R/W	<a href="#">235</a>
0x2008 8800 - 0x2008 8BFC	CRSR_IMG	Cursor Image registers	0	R/W	<a href="#">236</a>
0x2008 8C00	CRSR_CTRL	Cursor Control register	0	R/W	<a href="#">237</a>
0x2008 8C04	CRSR_CFG	Cursor Configuration register	0	R/W	<a href="#">238</a>
0x2008 8C08	CRSR_PAL0	Cursor Palette register 0	0	R/W	<a href="#">239</a>
0x2008 8C0C	CRSR_PAL1	Cursor Palette register 1	0	R/W	<a href="#">240</a>
0x2008 8C10	CRSR_XY	Cursor XY Position register	0	R/W	<a href="#">241</a>
0x2008 8C14	CRSR_CLIP	Cursor Clip Position register	0	R/W	<a href="#">242</a>
0x2008 8C20	CRSR_INTMSK	Cursor Interrupt Mask register	0	R/W	<a href="#">243</a>
0x2008 8C24	CRSR_INTCLR	Cursor Interrupt Clear register	0	WO	<a href="#">244</a>
0x2008 8C28	CRSR_INTRAW	Cursor Raw Interrupt Status register	0	RO	<a href="#">245</a>
0x2008 8C2C	CRSR_INTSTAT	Cursor Masked Interrupt Status register	0	RO	<a href="#">246</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 12.7.1 LCD Configuration register (LCD\_CFG, RW - 0x400F C1B8)

The LCD\_CFG register controls the prescaling of the clock used for LCD data generation.

The contents of the LCD\_CFG register are described in [Table 221](#).

**Table 221. LCD Configuration register (LCD\_CFG, RW - 0x400F C1B8)**

Bits	Function	Description	Reset value
4:0	CLKDIV	LCD panel clock prescaler selection. The value in the this register plus 1 is used to divide the selected input clock (see the CLKSEL bit in the LCD_POL register), to produce the panel clock.	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.2 Horizontal Timing register (LCD\_TIMH, RW - 0x2008 8000)

The LCD\_TIMH register controls the Horizontal Synchronization pulse Width (HSW), the Horizontal Front Porch (HFP) period, the Horizontal Back Porch (HBP) period, and the Pixels-Per-Line (PPL).

The contents of the LCD\_TIMH register are described in [Table 222](#).

**Table 222. Horizontal Timing register (LCD\_TIMH, RW - 0x2008 8000)**

Bits	Function	Description	Reset value
1:0	-	Reserved. Read value is undefined, only zero should be written.	-
7:2	PPL	Pixels-per-line. The PPL bit field specifies the number of pixels in each line or row of the screen. PPL is a 6-bit value that represents between 16 and 1024 pixels per line. PPL counts the number of pixel clocks that occur before the HFP is applied. Program the value required divided by 16, minus 1. Actual pixels-per-line = 16 * (PPL + 1). For example, to obtain 320 pixels per line, program PPL as (320/16) - 1 = 19.	0
15:8	HSW	Horizontal synchronization pulse width. The 8-bit HSW field specifies the pulse width of the line clock in passive mode, or the horizontal synchronization pulse in active mode. Program with desired value minus 1.	0
23:16	HFP	Horizontal front porch. The 8-bit HFP field sets the number of pixel clock intervals at the end of each line or row of pixels, before the LCD line clock is pulsed. When a complete line of pixels is transmitted to the LCD driver, the value in HFP counts the number of pixel clocks to wait before asserting the line clock. HFP can generate a period of 1-256 pixel clock cycles. Program with desired value minus 1.	0
31:24	HBP	Horizontal back porch. The 8-bit HBP field is used to specify the number of pixel clock periods inserted at the beginning of each line or row of pixels. After the line clock for the previous line has been deasserted, the value in HBP counts the number of pixel clocks to wait before starting the next display line. HBP can generate a delay of 1-256 pixel clock cycles. Program with desired value minus 1.	0

#### 12.7.2.1 Horizontal timing restrictions

DMA requests new data at the start of a horizontal display line. Some time must be allowed for the DMA transfer and for data to propagate down the FIFO path in the LCD interface. The data path latency forces some restrictions on the usable minimum values for horizontal porch width in STN mode. The minimum values are HSW = 2 and HBP = 2.

Single panel mode:

- HSW = 3 pixel clock cycles
- HBP = 5 pixel clock cycles
- HFP = 5 pixel clock cycles
- Panel Clock Divisor (PCD) = 1 (LCDCLK / 3)

Dual panel mode:

- HSW = 3 pixel clock cycles
- HBP = 5 pixel clock cycles
- HFP = 5 pixel clock cycles
- PCD = 5 (LCDCLK / 7)

If enough time is given at the start of the line, for example, setting HSW = 6, HBP = 10, data does not corrupt for PCD = 4, the minimum value.

### 12.7.3 Vertical Timing register (LCD\_TIMV, RW - 0x2008 8004)

The LCD\_TIMV register controls the Vertical Synchronization pulse Width (VSW), the Vertical Front Porch (VFP) period, the Vertical Back Porch (VBP) period, and the Lines-Per-Panel (LPP).

The contents of the LCD\_TIMV register are described in [Table 223](#).

**Table 223. Vertical Timing register (LCD\_TIMV, RW - 0x2008 8004)**

Bits	Function	Description	Reset value
9:0	LPP	<p>Lines per panel.</p> <p>This is the number of active lines per screen. The LPP field specifies the total number of lines or rows on the LCD panel being controlled. LPP is a 10-bit value allowing between 1 and 1024 lines. Program the register with the number of lines per LCD panel, minus 1. For dual panel displays, program the register with the number of lines on each of the upper and lower panels.</p>	0
15:10	VSW	<p>Vertical synchronization pulse width.</p> <p>This is the number of horizontal synchronization lines. The 6-bit VSW field specifies the pulse width of the vertical synchronization pulse. Program the register with the number of lines required, minus one.</p> <p>The number of horizontal synchronization lines must be small (for example, program to zero) for passive STN LCDs. The higher the value the worse the contrast on STN LCDs.</p>	0
23:16	VFP	<p>Vertical front porch.</p> <p>This is the number of inactive lines at the end of a frame, before the vertical synchronization period. The 8-bit VFP field specifies the number of line clocks to insert at the end of each frame. When a complete frame of pixels is transmitted to the LCD display, the value in VFP is used to count the number of line clock periods to wait.</p> <p>After the count has elapsed, the vertical synchronization signal, LCD_FP, is asserted in active mode, or extra line clocks are inserted as specified by the VSW bit-field in passive mode. VFP generates 0–255 line clock cycles. Program to zero on passive displays for improved contrast.</p>	0



**Table 223. Vertical Timing register (LCD\_TIMV, RW - 0x2008 8004)**

Bits	Function	Description	Reset value
31:24	VBP	Vertical back porch.  This is the number of inactive lines at the start of a frame, after the vertical synchronization period. The 8-bit VBP field specifies the number of line clocks inserted at the beginning of each frame. The VBP count starts immediately after the vertical synchronization signal for the previous frame has been negated for active mode, or the extra line clocks have been inserted as specified by the VSW bit field in passive mode. After this has occurred, the count value in VBP sets the number of line clock periods inserted before the next frame. VBP generates 0–255 extra line clock cycles. Program to zero on passive displays for improved contrast.	0

### 12.7.4 Clock and Signal Polarity register (LCD\_POL, RW - 0x2008 8008)

The LCD\_POL register controls various details of clock timing and signal polarity.

The contents of the LCD\_POL register are described in [Table 224](#).

**Table 224. Clock and Signal Polarity register (LCD\_POL, RW - 0x2008 8008)**

Bits	Function	Description	Reset value
4:0	PCD_LO	Lower five bits of panel clock divisor.  The ten-bit PCD field, comprising PCD_HI (bits 31:27 of this register) and PCD_LO, is used to derive the LCD panel clock frequency LCD_DCLK from the input clock, $LCD\_DCLK = LCDCLK / (PCD + 2)$ .  For monochrome STN displays with a 4 or 8-bit interface, the panel clock is a factor of four and eight down from the actual individual pixel clock rate. For color STN displays, 22/3 pixels are output per LCD_DCLK cycle, so the panel clock is 0.375 times the pixel rate.  For TFT displays, the pixel clock divider can be bypassed by setting the BCD bit in this register.  <b>Note:</b> data path latency forces some restrictions on the usable minimum values for the panel clock divider in STN modes: Single panel color mode, $PCD = 1$ ( $LCD\_DCLK = LCDCLK / 3$ ). Dual panel color mode, $PCD = 4$ ( $LCD\_DCLK = LCDCLK / 6$ ). Single panel monochrome 4-bit interface mode, $PCD = 2$ ( $LCD\_DCLK = LCDCLK / 4$ ). Dual panel monochrome 4-bit interface mode and single panel monochrome 8-bit interface mode, $PCD = 6$ ( $LCD\_DCLK = LCDCLK / 8$ ). Dual panel monochrome 8-bit interface mode, $PCD = 14$ ( $LCD\_DCLK = LCDCLK / 16$ ).	0
5	CLKSEL	Clock Select.  This bit controls the selection of the source for LCDCLK. 0 = the clock source for the LCD block is CCLK. 1 = the clock source for the LCD block is LCD_CLKIN (external clock input for the LVD).	0
10:6	ACB	AC bias pin frequency.  The AC bias pin frequency is only applicable to STN displays. These require the pixel voltage polarity to periodically reverse to prevent damage caused by DC charge accumulation. Program this field with the required value minus one to apply the number of line clocks between each toggle of the AC bias pin, LCD_ENAB_M. This field has no effect if the LCD is operating in TFT mode, when the LCD_ENAB_M pin is used as a data enable signal.	0

**Table 224. Clock and Signal Polarity register (LCD\_POL, RW - 0x2008 8008)**

Bits	Function	Description	Reset value
11	IVS	Invert vertical synchronization. The IVS bit inverts the polarity of the LCD_FP signal. 0 = LCD_FP pin is active HIGH and inactive LOW. 1 = LCD_FP pin is active LOW and inactive HIGH.	0
12	IHS	Invert horizontal synchronization. The IHS bit inverts the polarity of the LCD_LP signal. 0 = LCD_LP pin is active HIGH and inactive LOW. 1 = LCD_LP pin is active LOW and inactive HIGH.	0
13	IPC	Invert panel clock. The IPC bit selects the edge of the panel clock on which pixel data is driven out onto the LCD data lines. 0 = Data is driven on the LCD data lines on the rising edge of LCD_DCLK. 1 = Data is driven on the LCD data lines on the falling edge of LCD_DCLK.	0
14	IOE	Invert output enable. This bit selects the active polarity of the output enable signal in TFT mode. In this mode, the LCD_ENAB_M pin is used as an enable that indicates to the LCD panel when valid display data is available. In active display mode, data is driven onto the LCD data lines at the programmed edge of LCD_DCLK when LCD_ENAB_M is in its active state. 0 = LCD_ENAB_M output pin is active HIGH in TFT mode. 1 = LCD_ENAB_M output pin is active LOW in TFT mode.	0
15	-	Reserved. Read value is undefined, only zero should be written.	-
25:16	CPL	Clocks per line. This field specifies the number of actual LCD_DCLK clocks to the LCD panel on each line. This is the number of PPL divided by either 1 (for TFT), 4 or 8 (for monochrome passive), 2 2/3 (for color passive), minus one. This must be correctly programmed in addition to the PPL bit in the LCD_TIMH register for the LCD display to work correctly.	0
26	BCD	Bypass pixel clock divider. Setting this to 1 bypasses the pixel clock divider logic. This is mainly used for TFT displays.	0
31:27	PCD_HI	Upper five bits of panel clock divisor. See description for PCD_LO, in bits [4:0] of this register.	0

### 12.7.5 Line End Control register (LCD\_LE, RW - 0x2008 800C)

The LCD\_LE register controls the enabling of line-end signal LCD\_LE. When enabled, a positive pulse, four LCDCLK periods wide, is output on LCD\_LE after a programmable delay, LED, from the last pixel of each display line. If the line-end signal is disabled it is held permanently LOW.

The contents of the LCD\_LE register are described in [Table 225](#).

**Table 225. Line End Control register (LCD\_LE, RW - 0x2008 800C)**

Bits	Function	Description	Reset value
6:0	LED	Line-end delay. Controls Line-end signal delay from the rising-edge of the last panel clock, LCD_DCLK. Program with number of LCDCLK clock periods minus 1.	0
15:7	-	Reserved. Read value is undefined, only zero should be written.	-
16	LEE	LCD Line end enable. 0 = LCD_LE disabled (held LOW). 1 = LCD_LE signal active.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.6 Upper Panel Frame Base Address register (LCD\_UPBASE, RW - 0x2008 8010)

The LCD\_UPBASE register is the color LCD upper panel DMA base address register, and is used to program the base address of the frame buffer for the upper panel. LCDUPBase (and LCDLPBase for dual panels) must be initialized before enabling the LCD controller. The base address must be doubleword aligned.

Optionally, the value may be changed mid-frame to create double-buffered video displays. These registers are copied to the corresponding current registers at each LCD vertical synchronization. This event causes the LNBU bit and an optional interrupt to be generated. The interrupt can be used to reprogram the base address when generating double-buffered video.

The contents of the LCD\_UPBASE register are described in [Table 226](#).

**Table 226. Upper Panel Frame Base register (LCD\_UPBASE, RW - 0x2008 8010)**

Bits	Function	Description	Reset value
2:0	-	Reserved. Read value is undefined, only zero should be written.	-
31:3	LCDUPBASE	LCD upper panel base address. This is the start address of the upper panel frame data in memory and is doubleword aligned.	0

### 12.7.7 Lower Panel Frame Base Address register (LCD\_LPBASE, RW - 0x2008 8014)

The LCD\_LPBASE register is the color LCD lower panel DMA base address register, and is used to program the base address of the frame buffer for the lower panel. LCDLPBase must be initialized before enabling the LCD controller. The base address must be doubleword aligned.

Optionally, the value may be changed mid-frame to create double-buffered video displays. These registers are copied to the corresponding current registers at each LCD vertical synchronization. This event causes the LNBU bit and an optional interrupt to be generated. The interrupt can be used to reprogram the base address when generating double-buffered video.

The contents of the LCD\_LPBASE register are described in [Table 227](#).

**Table 227. Lower Panel Frame Base register (LCD\_LPBASE, RW - 0x2008 8014)**

Bits	Function	Description	Reset value
2:0	-	Reserved. Read value is undefined, only zero should be written.	-
31:3	LCDLPBASE	LCD lower panel base address. This is the start address of the lower panel frame data in memory and is doubleword aligned.	0

### 12.7.8 LCD Control register (LCD\_CTRL, RW - 0x2008 8018)

The LCD\_CTRL register controls the LCD operating mode and the panel pixel parameters.

The contents of the LCD\_CTRL register are described in [Table 228](#).

**Table 228. LCD Control register (LCD\_CTRL, RW - 0x2008 8018)**

Bits	Function	Description	Reset value
0	LcdEn	LCD enable control bit. 0 = LCD disabled. Signals LCD_LP, LCD_DCLK, LCD_FP, LCD_ENAB_M, and LCD_LE are low. 1 = LCD enabled. Signals LCD_LP, LCD_DCLK, LCD_FP, LCD_ENAB_M, and LCD_LE are high. See LCD power-up and power-down sequence for details on LCD power sequencing.	0
3:1	LcdBpp	LCD bits per pixel: Selects the number of bits per LCD pixel: 000 = 1 bpp. 001 = 2 bpp. 010 = 4 bpp. 011 = 8 bpp. 100 = 16 bpp. 101 = 24 bpp (TFT panel only). 110 = 16 bpp, 5:6:5 mode. 111 = 12 bpp, 4:4:4 mode.	0
4	LcdBW	STN LCD monochrome/color selection. 0 = STN LCD is color. 1 = STN LCD is monochrome. This bit has no meaning in TFT mode.	0
5	LcdTFT	LCD panel TFT type selection. 0 = LCD is an STN display. Use gray scaler. 1 = LCD is a TFT display. Do not use gray scaler.	0
6	LcdMono8	Monochrome LCD interface width. This bit controls whether a monochrome STN LCD uses a 4 or 8-bit parallel interface. It has no meaning in other modes and must be programmed to zero. 0 = monochrome LCD uses a 4-bit interface. 1 = monochrome LCD uses a 8-bit interface.	0
7	LcdDual	Single or Dual LCD panel selection. STN LCD interface is: 0 = single-panel. 1 = dual-panel.	0

**Table 228. LCD Control register (LCD\_CTRL, RW - 0x2008 8018)**

Bits	Function	Description	Reset value
8	BGR	Color format selection. 0 = RGB: normal output. 1 = BGR: red and blue swapped.	0
9	BEBO	Big-endian Byte Order. Controls byte ordering in memory: 0 = little-endian byte order. 1 = big-endian byte order.	0
10	BEPO	Big-Endian Pixel Ordering. Controls pixel ordering within a byte: 0 = little-endian ordering within a byte. 1 = big-endian pixel ordering within a byte. The BEPO bit selects between little and big-endian pixel packing for 1, 2, and 4 bpp display modes, it has no effect on 8 or 16 bpp pixel formats. See Pixel serializer for more information on the data format.	0
11	LcdPwr	LCD power enable. 0 = power not gated through to LCD panel and LCD_VD[23:0] signals disabled, (held LOW). 1 = power gated through to LCD panel and LCD_VD[23:0] signals enabled, (active). See LCD power-up and power-down sequence for details on LCD power sequencing.	0
13:12	LcdVComp	LCD Vertical Compare Interrupt. Generate VComp interrupt at: 00 = start of vertical synchronization. 01 = start of back porch. 10 = start of active video. 11 = start of front porch.	0
15:14	-	Reserved. Read value is undefined, only zero should be written.	-
16	WATERMARK	LCD DMA FIFO watermark level. Controls when DMA requests are generated: 0 = An LCD DMA request is generated when either of the DMA FIFOs have four or more empty locations. 1 = An LCD DMA request is generated when either of the DMA FIFOs have eight or more empty locations.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.9 Interrupt Mask register (LCD\_INTMSK, RW - 0x2008 801C)

The LCD\_INTMSK register controls whether various LCD interrupts occur. Setting bits in this register enables the corresponding raw interrupt LCD\_INTRAW status bit values to be passed to the LCD\_INTSTAT register for processing as interrupts.

The contents of the LCD\_INTMSK register are described in [Table 229](#).

**Table 229. Interrupt Mask register (LCD\_INTMSK, RW - 0x2008 801C)**

Bits	Function	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	-
1	FUFIM	FIFO underflow interrupt enable. 0: The FIFO underflow interrupt is disabled. 1: Interrupt will be generated when the FIFO underflows.	0
2	LNBUIM	LCD next base address update interrupt enable. 0: The base address update interrupt is disabled. 1: Interrupt will be generated when the LCD base address registers have been updated from the next address registers.	0
3	VCompIM	Vertical compare interrupt enable. 0: The vertical compare time interrupt is disabled. 1: Interrupt will be generated when the vertical compare time (as defined by LcdVComp field in the LCD_CTRL register) is reached.	0
4	BERIM	AHB master error interrupt enable. 0: The AHB Master error interrupt is disabled. 1: Interrupt will be generated when an AHB Master error occurs.	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.10 Raw Interrupt Status register (LCD\_INTRAW, RW - 0x2008 8020)

The LCD\_INTRAW register contains status flags for various LCD controller events. These flags can generate an interrupts if enabled by mask bits in the LCD\_INTMSK register.

The contents of LCD\_INTRAW register are described in [Table 230](#).

**Table 230. Raw Interrupt Status register (LCD\_INTRAW, RW - 0x2008 8020)**

Bits	Function	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	-
1	FUF RIS	FIFO underflow raw interrupt status. Set when either the upper or lower DMA FIFOs have been read accessed when empty causing an underflow condition to occur. Generates an interrupt if the FUFIM bit in the LCD_INTMSK register is set.	0
2	LNBU RIS	LCD next address base update raw interrupt status. Mode dependent. Set when the current base address registers have been successfully updated by the next address registers. Signifies that a new next address can be loaded if double buffering is in use. Generates an interrupt if the LNBUIM bit in the LCD_INTMSK register is set.	0
3	VComp RIS	Vertical compare raw interrupt status. Set when one of the four vertical regions is reached, as selected by the LcdVComp bits in the LCD_CTRL register. Generates an interrupt if the VCompIM bit in the LCD_INTMSK register is set.	0
4	BERRAW	AHB master bus error raw interrupt status. Set when the AHB master interface receives a bus error response from a slave. Generates an interrupt if the BERIM bit in the LCD_INTMSK register is set.	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.11 Masked Interrupt Status register (LCD\_INTSTAT, RW - 0x2008 8024)

The LCD\_INTSTAT register is Read-Only, and contains a bit-by-bit logical AND of the LCD\_INTRAW register and the LCD\_INTMSK register. A logical OR of all interrupts is provided to the system interrupt controller.

The contents of LCD\_INTSTAT register are described in [Table 231](#).

**Table 231. Masked Interrupt Status register (LCD\_INTSTAT, RW - 0x2008 8024)**

Bits	Function	Description	Reset value
0	-	Reserved. The value read from a reserved bit is not defined.	-
1	FUFMIS	FIFO underflow masked interrupt status. Set when the both the FUFMIS bit in the LCD_INTRAW register and the FUFMIS bit in the LCD_INTMSK register are set.	0
2	LNBUMIS	LCD next address base update masked interrupt status. Set when the both the LNBUIS bit in the LCD_INTRAW register and the LNBUIS bit in the LCD_INTMSK register are set.	0
3	VCompMIS	Vertical compare masked interrupt status. Set when the both the VCompMIS bit in the LCD_INTRAW register and the VCompMIS bit in the LCD_INTMSK register are set.	0
4	BERMIS	AHB master bus error masked interrupt status. Set when the both the BERRAW bit in the LCD_INTRAW register and the BERIM bit in the LCD_INTMSK register are set.	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.12 Interrupt Clear register (LCD\_INTCLR, RW - 0x2008 8028)

The LCD\_INTCLR register is Write-Only. Writing a logic 1 to the relevant bit clears the corresponding interrupt.

The contents of the LCD\_INTCLR register are described in [Table 232](#).

**Table 232. Interrupt Clear register (LCD\_INTCLR, RW - 0x2008 8028)**

Bits	Function	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	-
1	FUFIC	FIFO underflow interrupt clear. Writing a 1 to this bit clears the FIFO underflow interrupt.	0
2	LNBUIC	LCD next address base update interrupt clear. Writing a 1 to this bit clears the LCD next address base update interrupt.	0
3	VCompIC	Vertical compare interrupt clear. Writing a 1 to this bit clears the vertical compare interrupt.	0
4	BERIC	AHB master error interrupt clear. Writing a 1 to this bit clears the AHB master error interrupt.	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.13 Upper Panel Current Address register (LCD\_UPCURR, RW - 0x2008 802C)

The LCD\_UPCURR register is Read-Only, and contains an approximate value of the upper panel data DMA address when read.

**Note:** This register can change at any time and therefore can only be used as a rough indication of display position.

The contents of the LCD\_UPCURR register are described in [Table 233](#).

**Table 233. Upper Panel Current Address register (LCD\_UPCURR, RW - 0x2008 802C)**

Bits	Function	Description	Reset value
31:0	LCDUPCURR	LCD Upper Panel Current Address. Contains the current LCD upper panel data DMA address.	0

### 12.7.14 Lower Panel Current Address register (LCD\_LPCURR, RW - 0x2008 8030)

The LCD\_LPCURR register is Read-Only, and contains an approximate value of the lower panel data DMA address when read.

**Note:** This register can change at any time and therefore can only be used as a rough indication of display position.

The contents of the LCD\_LPCURR are described in [Table 234](#).

**Table 234. Lower Panel Current Address register (LCD\_LPCURR, RW - 0x2008 8030)**

Bits	Function	Description	Reset value
31:0	LCDLPCURR	LCD Lower Panel Current Address. Contains the current LCD lower panel data DMA address.	0

### 12.7.15 Color Palette registers (LCD\_PAL, RW - 0x2008 8200 to 0x2008 83FC)

The LCD\_PAL register contain 256 palette entries organized as 128 locations of two entries per word.

Each word location contains two palette entries. This means that 128 word locations are used for the palette. When configured for little-endian byte ordering, bits [15:0] are the lower numbered palette entry and [31:16] are the higher numbered palette entry. When configured for big-endian byte ordering this is reversed, because bits [31:16] are the low numbered palette entry and [15:0] are the high numbered entry.

**Note:** Only TFT displays use all of the palette entry bits.

The contents of the LCD\_PAL register are described in [Table 235](#).

**Table 235. Color Palette registers (LCD\_PAL, RW - 0x2008 8200 to 0x2008 83FC)**

Bits	Function	Description	Reset value
4:0	R[4:0]	Red palette data. For STN displays, only the four MSBs, bits [4:1], are used. For monochrome displays only the red palette data is used. All of the palette registers have the same bit fields.	0
9:5	G[4:0]	Green palette data.	0
14:10	B[4:0]	Blue palette data.	0
15	I	Intensity / unused bit. Can be used as the LSB of the R, G, and B inputs to a 6:6:6 TFT display, doubling the number of colors to 64K, where each color has two different intensities.	0



**Table 235. Color Palette registers (LCD\_PAL, RW - 0x2008 8200 to 0x2008 83FC)**

Bits	Function	Description	Reset value
20:16	R[4:0]	Red palette data. For STN displays, only the four MSBs, bits [4:1], are used. For monochrome displays only the red palette data is used. All of the palette registers have the same bit fields.	0
25:21	G[4:0]	Green palette data.	0
30:26	B[4:0]	Blue palette data.	0
31	I	Intensity / unused bit. Can be used as the LSB of the R, G, and B inputs to a 6:6:6 TFT display, doubling the number of colors to 64K, where each color has two different intensities.	0

### 12.7.16 Cursor Image registers (CRSR\_IMG, RW - 0x2008 8800 to 0x2008 8BFC)

The CRSR\_IMG register area contains 256-word wide values which are used to define the image or images overlaid on the display by the hardware cursor mechanism. The image must always be stored in LBBP mode (little-endian byte, big-endian pixel) mode, as described in [Section 12.6.5.6](#). Two bits are used to encode color and transparency for each pixel in the cursor.

Depending on the state of bit 0 in the CRSR\_CFG register (see Cursor Configuration register description), the cursor image RAM contains either four 32x32 cursor images, or a single 64x64 cursor image.

The two colors defined for the cursor are mapped onto values from the CRSR\_PAL0 and CRSR\_PAL1 registers (see Cursor Palette register descriptions).

The contents of the CRSR\_IMG register are described in [Table 236](#).

**Table 236. Cursor Image registers (CRSR\_IMG, RW - 0x2008 8800 to 0x2008 8BFC)**

Bits	Function	Description	Reset value
31:0	CRSR_IMG	Cursor Image data. The 256 words of the cursor image registers define the appearance of either one 64x64 cursor, or 4 32x32 cursors.	0

### 12.7.17 Cursor Control register (CRSR\_CTRL, RW - 0x2008 8C00)

The CRSR\_CTRL register provides access to frequently used cursor functions, such as the display on/off control for the cursor, and the cursor number.

If a 32x32 cursor is selected, one of four 32x32 cursors can be enabled. The images each occupy one quarter of the image memory, with Cursor0 from location 0, followed by Cursor1 from address 0x100, Cursor2 from 0x200 and Cursor3 from 0x300. If a 64x64 cursor is selected only one cursor fits in the image buffer, and no selection is possible.

Similar frame synchronization rules apply to the cursor number as apply to the cursor coordinates. If CrsrFramesync is 1, the displayed cursor image is only changed during the vertical frame blanking period. If CrsrFrameSync is 0, the cursor image index is changed immediately, even if the cursor is currently being scanned.

The contents of the CRSR\_CTRL register are described in [Table 237](#).

**Table 237. Cursor Control register (CRSR\_CTRL, RW - 0x2008 8C00)**

Bits	Function	Description	Reset value
0	CrsrOn	Cursor enable. 0 = Cursor is not displayed. 1 = Cursor is displayed.	0
3:1	-	Reserved. Read value is undefined, only zero should be written.	0
5:4	CrsrNum[1:0]	Cursor image number. If the selected cursor size is 6x64, this field has no effect. If the selected cursor size is 32x32: 00 = Cursor0. 01 = Cursor1. 10 = Cursor2. 11 = Cursor3.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	0

### 12.7.18 Cursor Configuration register (CRSR\_CFG, RW - 0x2008 8C04)

The CRSR\_CFG register provides overall configuration information for the hardware cursor.

The contents of the CRSR\_CFG register are described in [Table 238](#).

**Table 238. Cursor Configuration register (CRSR\_CFG, RW - 0x2008 8C04)**

Bits	Function	Description	Reset value
0	CrsrSize	Cursor size selection. 0 = 32x32 pixel cursor. Allows for 4 defined cursors. 1 = 64x64 pixel cursor.	0
1	FrameSync	Cursor frame synchronization type. 0 = Cursor coordinates are asynchronous. 1 = Cursor coordinates are synchronized to the frame synchronization pulse.	0
31:2	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.19 Cursor Palette register 0 (CRSR\_PAL0, RW - 0x2008 8C08)

The cursor palette registers provide color palette information for the visible colors of the cursor. Color0 maps through CRSR\_PAL0.

The register provides 24-bit RGB values that are displayed according to the abilities of the LCD panel in the same way as the frame-buffers palette output is displayed.

In monochrome STN mode, only the upper 4 bits of the Red field are used. In STN color mode, the upper 4 bits of the Red, Blue, and Green fields are used. In 24 bits per pixel mode, all 24 bits of the palette registers are significant.

The contents of the CRSR\_PAL0 register are described in [Table 239](#).

**Table 239. Cursor Palette register 0 (CRSR\_PAL0, RW - 0x2008 8C08)**

Bits	Function	Description	Reset value
7:0	Red	Red color component	0
15:8	Green	Green color component	0
23:16	Blue	Blue color component.	0
31:24	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.20 Cursor Palette register 1 (CRSR\_PAL1, RW - 0x2008 8C0C)

The cursor palette registers provide color palette information for the visible colors of the cursor. Color1 maps through CRSR\_PAL1.

The register provides 24-bit RGB values that are displayed according to the abilities of the LCD panel in the same way as the frame-buffers palette output is displayed.

In monochrome STN mode, only the upper 4 bits of the Red field are used. In STN color mode, the upper 4 bits of the Red, Blue, and Green fields are used. In 24 bits per pixel mode, all 24 bits of the palette registers are significant.

The contents of the CRSR\_PAL1 register are described in [Table 240](#).

**Table 240. Cursor Palette register 1 (CRSR\_PAL1, RW - 0x2008 8C0C)**

Bits	Function	Description	Reset value
7:0	Red	Red color component	0
15:8	Green	Green color component	0
23:16	Blue	Blue color component.	0
31:24	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.21 Cursor XY Position register (CRSR\_XY, RW - 0x2008 8C10)

The CRSR\_XY register defines the distance of the top-left edge of the cursor from the top-left side of the cursor overlay. refer to the section on Cursor Clipping for more details.

If the FrameSync bit in the CRSR\_CFG register is 0, the cursor position changes immediately, even if the cursor is currently being scanned. If Framesync is 1, the cursor position is only changed during the next vertical frame blanking period.

The contents of the CRSR\_XY register are described in [Table 241](#).

**Table 241. Cursor XY Position register (CRSR\_XY, RW - 0x2008 8C10)**

Bits	Function	Description	Reset value
9:0	CrsrX	X ordinate of the cursor origin measured in pixels. When 0, the left edge of the cursor is at the left of the display.	0
15:10	-	Reserved. Read value is undefined, only zero should be written.	-
25:16	CrsrY	Y ordinate of the cursor origin measured in pixels. When 0, the top edge of the cursor is at the top of the display.	0
31:26	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.22 Cursor Clip Position register (CRSR\_CLIP, RW - 0x2008 8C14)

The CRSR\_CLIP register defines the distance from the top-left edge of the cursor image, to the first displayed pixel in the cursor image.

Different synchronization rules apply to the Cursor Clip registers than apply to the cursor coordinates. If the FrameSync bit in the CRSR\_CFG register is 0, the cursor clip point is changed immediately, even if the cursor is currently being scanned.

If the Framesync bit in the CRSR\_CFG register is 1, the displayed cursor image is only changed during the vertical frame blanking period, providing that the cursor position has been updated since the Clip register was programmed. When programming, the Clip register must be written before the Position register (ClcdCrsrXY) to ensure that in a given frame, the clip and position information is coherent.

The contents of the CRSR\_CLIP register are described in [Table 242](#).

**Table 242. Cursor Clip Position register (CRSR\_CLIP, RW - 0x2008 8C14)**

Bits	Function	Description	Reset value
5:0	CrsrClipX	Cursor clip position for X direction. Distance from the left edge of the cursor image to the first displayed pixel in the cursor. When 0, the first pixel of the cursor line is displayed.	0
7:6	-	Reserved. Read value is undefined, only zero should be written.	-
13:8	CrsrClipY	Cursor clip position for Y direction. Distance from the top of the cursor image to the first displayed pixel in the cursor. When 0, the first displayed pixel is from the top line of the cursor image.	0
31:14	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.23 Cursor Interrupt Mask register (CRSR\_INTMSK, RW - 0x2008 8C20)

The CRSR\_INTMSK register is used to enable or disable the cursor from interrupting the processor.

The contents of the CRSR\_INTMSK register are described in [Table 243](#).

**Table 243. Cursor Interrupt Mask register (CRSR\_INTMSK, RW - 0x2008 8C20)**

Bits	Function	Description	Reset value
0	CrsrIM	Cursor interrupt mask. When clear, the cursor never interrupts the processor. When set, the cursor interrupts the processor immediately after reading of the last word of cursor image.	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.24 Cursor Interrupt Clear register (CRSR\_INTCLR, RW - 0x2008 8C24)

The CRSR\_INTCLR register is used by software to clear the cursor interrupt status and the cursor interrupt signal to the processor.

The contents of the CRSR\_INTCLR register are described in [Table 244](#).

**Table 244. Cursor Interrupt Clear register (CRSR\_INTCLR, RW - 0x2008 8C24)**

Bits	Function	Description	Reset value
0	CrsrIC	Cursor interrupt clear. Writing a 0 to this bit has no effect. Writing a 1 to this bit causes the cursor interrupt status to be cleared.	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.25 Cursor Raw Interrupt Status register (CRSR\_INTRAW, RW - 0x2008 8C28)

The CRSR\_INTRAW register is set to indicate a cursor interrupt. When enabled via the CrsrIM bit in the CRSR\_INTMSK register, provides the interrupt to the system interrupt controller.

The contents of the CRSR\_INTRAW register are described in [Table 245](#).

**Table 245. Cursor Raw Interrupt Status register (CRSR\_INTRAW, RW - 0x2008 8C28)**

Bits	Function	Description	Reset value
0	CrsrRIS	Cursor raw interrupt status. The cursor interrupt status is set immediately after the last data is read from the cursor image for the current frame. This bit is cleared by writing to the CrsrIC bit in the CRSR_INTCLR register.	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	-

### 12.7.26 Cursor Masked Interrupt Status register (CRSR\_INTSTAT, RW - 0x2008 8C2C)

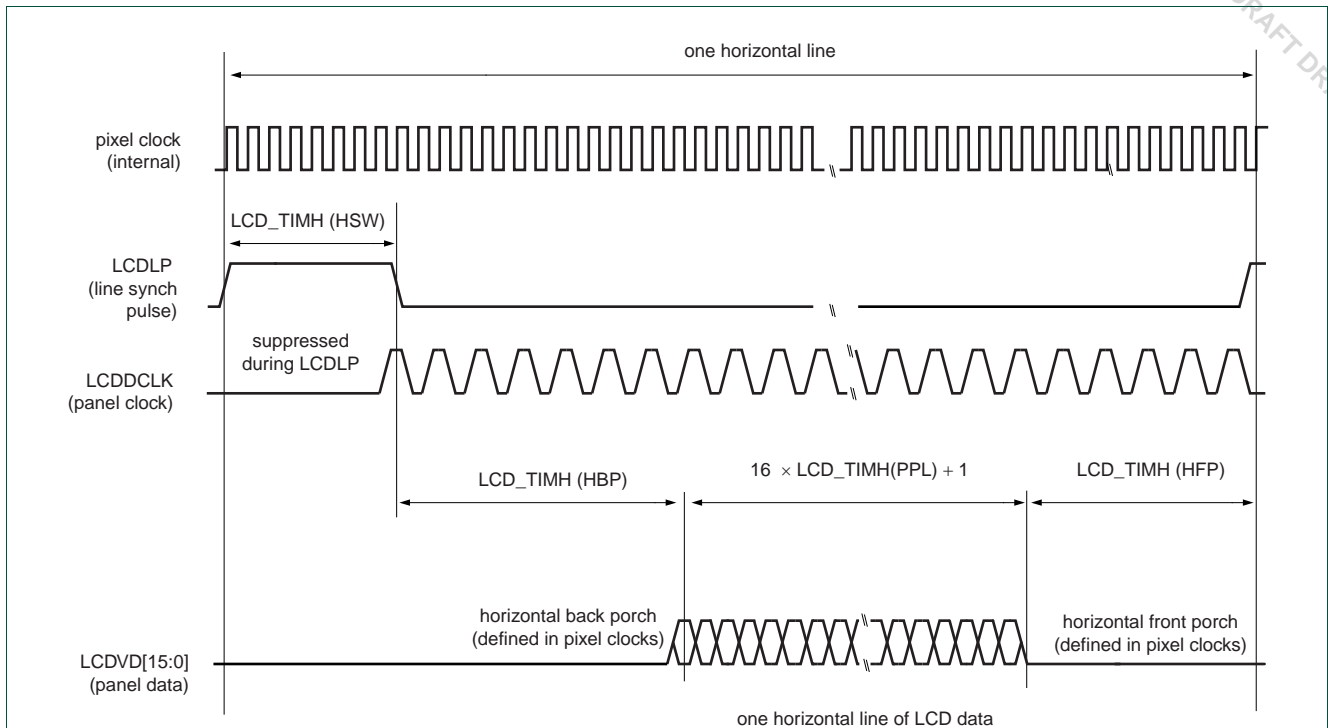
The CRSR\_INTSTAT register is set to indicate a cursor interrupt providing that the interrupt is not masked in the CRSR\_INTMSK register.

The contents of the CRSR\_INTSTAT register are described in [Table 246](#).

**Table 246. Cursor Masked Interrupt Status register (CRSR\_INTSTAT, RW - 0x2008 8C2C)**

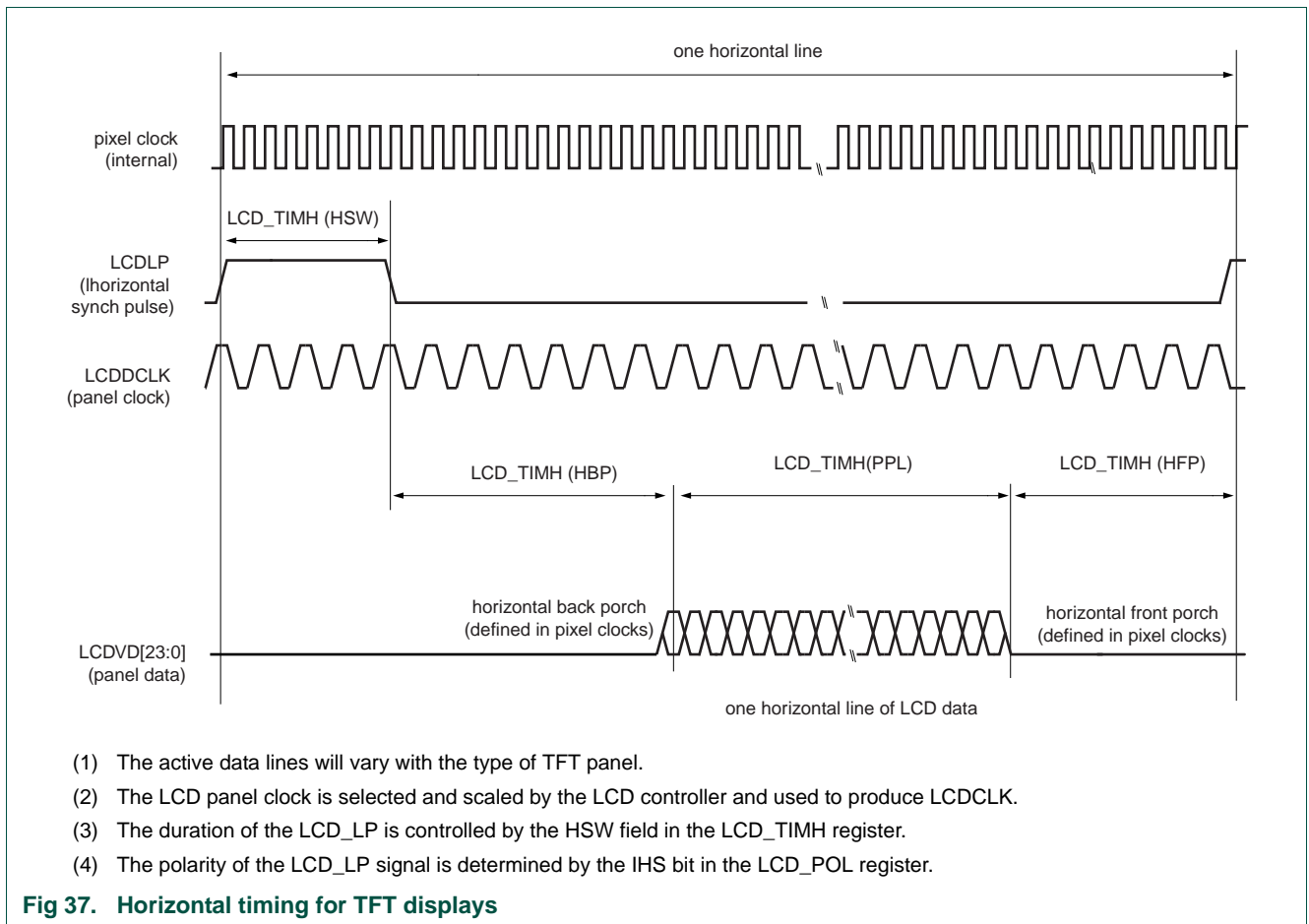
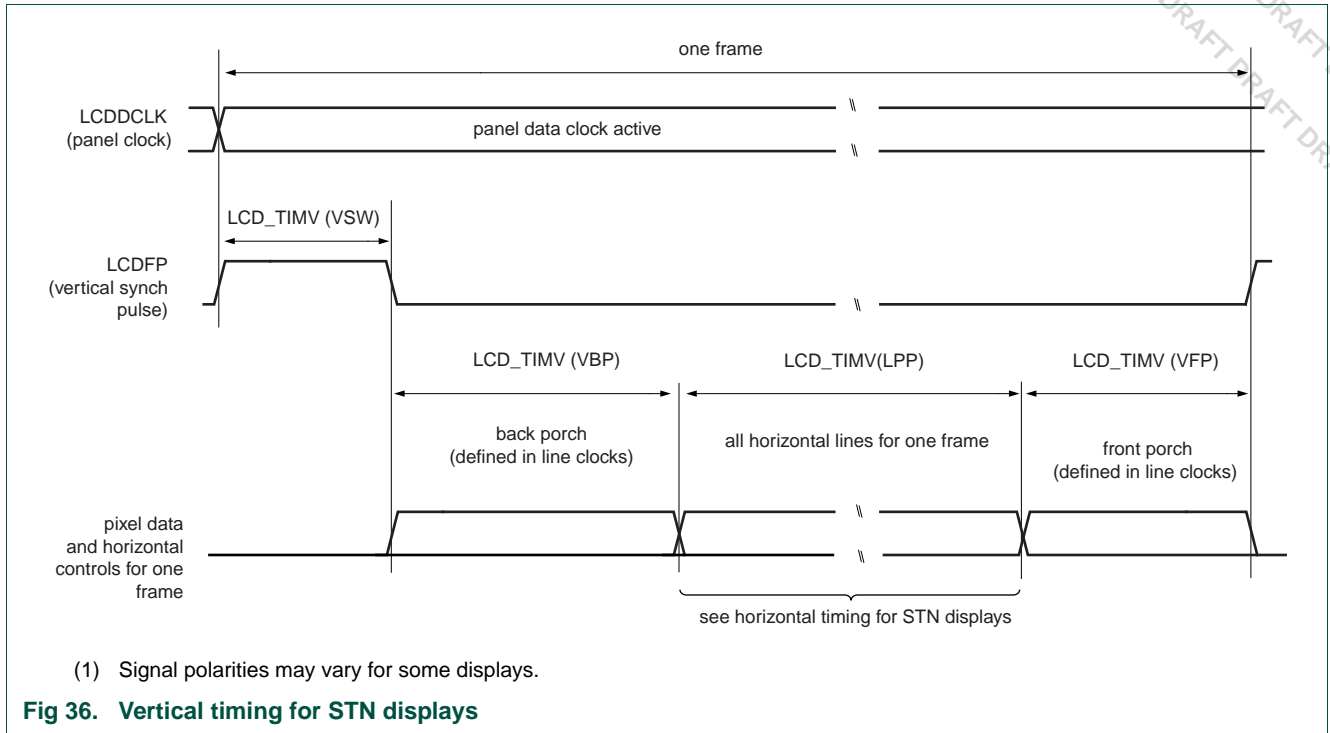
Bits	Function	Description	Reset value
0	CrsrMIS	Cursor masked interrupt status. The cursor interrupt status is set immediately after the last data read from the cursor image for the current frame, providing that the corresponding bit in the CRSR_INTMSK register is set. The bit remains clear if the CRSR_INTMSK register is clear. This bit is cleared by writing to the CRSR_INTCLR register.	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	-

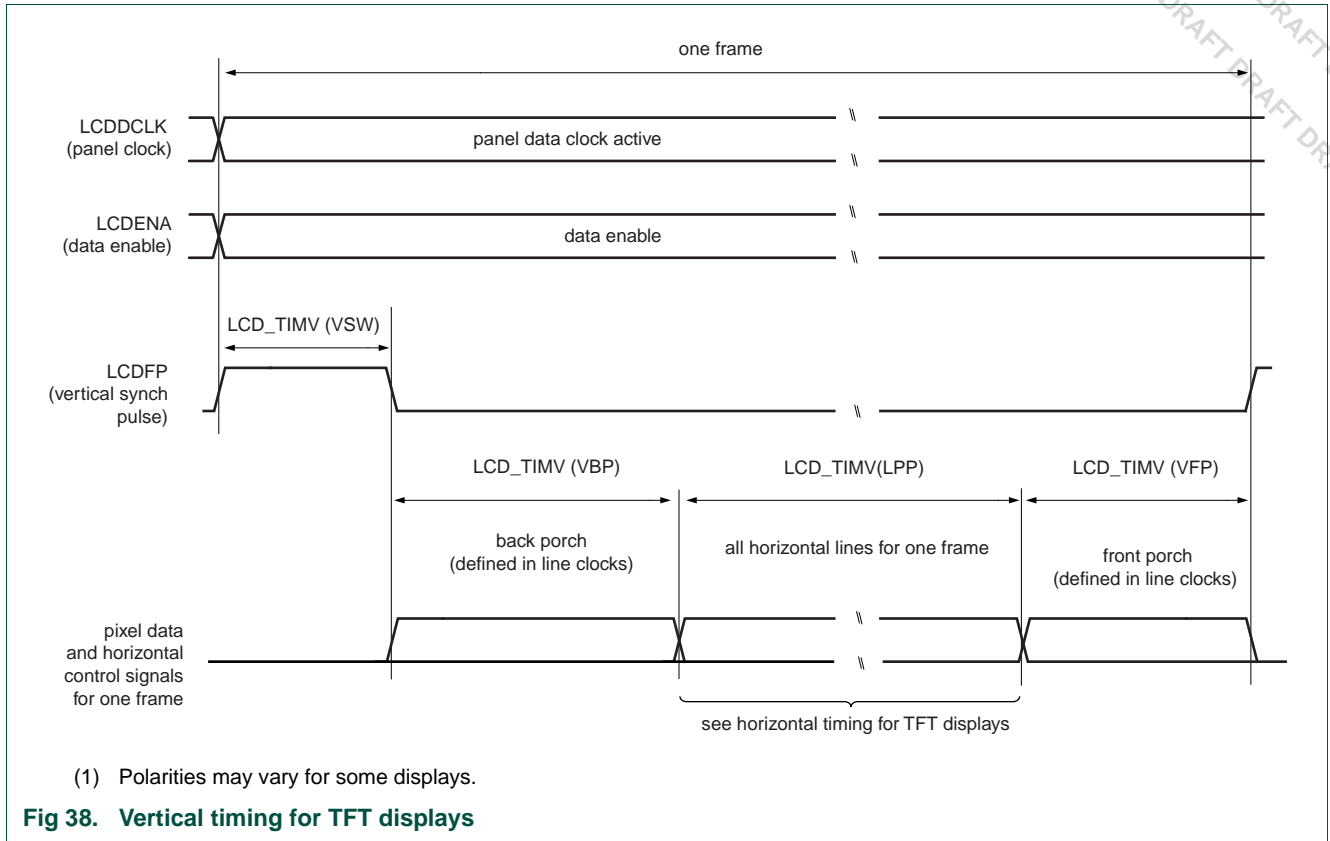
## 12.8 LCD timing diagrams



- (1) The active data lines will vary with the type of STN panel (4-bit, 8-bit, color, mono) and with single or dual frames.
- (2) The LCD panel clock is selected and scaled by the LCD controller and used to produce LCDCLK.
- (3) The duration of the LCD\_LP signal is controlled by the HSW field in the LCD\_TIMH register.
- (4) The Polarity of the LCD\_LP signal is determined by the IHS bit in the LCD\_POL register.

**Fig 35. Horizontal timing for STN displays**







## 12.9 LCD panel signal usage

Table 247. LCD panel connections for STN single panel mode

External pin	4-bit mono STN single panel		8-bit mono STN single panel		Color STN single panel	
	pin used	LCD function	pin used	LCD function	pin used	LCD function
LCD_VD[8] - LCD_VD[23]	-	-	-	-	-	-
LCD_VD[7]	-	-	P4[29]	UD[7]	P4[29]	UD[7]
LCD_VD[6]	-	-	P4[28]	UD[6]	P4[28]	UD[6]
LCD_VD[5]	-	-	P2[13]	UD[5]	P2[13]	UD[5]
LCD_VD[4]	-	-	P2[12]	UD[4]	P2[12]	UD[4]
LCD_VD[3]	P2[9]	UD[3]	P2[9]	UD[3]	P2[9]	UD[3]
LCD_VD[2]	P2[8]	UD[2]	P2[8]	UD[2]	P2[8]	UD[2]
LCD_VD[1]	P2[7]	UD[1]	P2[7]	UD[1]	P2[7]	UD[1]
LCD_VD[0]	P2[6]	UD[0]	P2[6]	UD[0]	P2[6]	UD[0]
LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP
LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M
LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP
LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK
LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE
LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR
LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN	P2[0]	LCD_PWR

Table 248. LCD panel connections for STN dual panel mode

External pin	4-bit mono STN dual panel		8-bit mono STN dual panel		Color STN dual panel	
	pin used	LCD function	pin used	LCD function	pin used	LCD function
LCD_VD[16] - LCD_VD[23]	-	-	-	-	-	-
LCD_VD[15]	-	-	P1[29]	LD[7]	P1[29]	LD[7]
LCD_VD[14]	-	-	P1[28]	LD[6]	P1[28]	LD[6]
LCD_VD[13]	-	-	P1[27]	LD[5]	P1[27]	LD[5]
LCD_VD[12]	-	-	P1[26]	LD[4]	P1[26]	LD[4]
LCD_VD[11]	P4[29]	LD[3]	P1[25]	LD[3]	P1[25]	LD[3]
LCD_VD[10]	P4[28]	LD[2]	P1[24]	LD[2]	P1[24]	LD[2]
LCD_VD[9]	P2[13]	LD[1]	P1[23]	LD[1]	P1[23]	LD[1]
LCD_VD[8]	P2[12]	LD[0]	P1[22]	LD[0]	P1[22]	LD[0]
LCD_VD[7]	-	-	P1[21]	UD[7]	P1[21]	UD[7]
LCD_VD[6]	-	-	P1[20]	UD[6]	P1[20]	UD[6]
LCD_VD[5]	-	-	P2[13]	UD[5]	P2[13]	UD[5]
LCD_VD[4]	-	-	P2[12]	UD[4]	P2[12]	UD[4]
LCD_VD[3]	P2[9]	UD[3]	P2[9]	UD[3]	P2[9]	UD[3]
LCD_VD[2]	P2[8]	UD[2]	P2[8]	UD[2]	P2[8]	UD[2]
LCD_VD[1]	P2[7]	UD[1]	P2[7]	UD[1]	P2[7]	UD[1]
LCD_VD[0]	P2[6]	UD[0]	P2[6]	UD[0]	P2[6]	UD[0]
LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP
LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M
LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP
LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK
LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE
LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR
LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN

Table 249. LCD panel connections for TFT panels

External pin	TFT 12 bit (4:4:4 mode)		TFT 16 bit (5:6:5 mode)		TFT 16 bit (1:5:5:5 mode)		TFT 24 bit	
	pin used	LCD function	pin used	LCD function	pin used	LCD function	pin used	LCD function
LCD_VD[23]	P1[29]	BLUE3	P1[29]	BLUE4	P1[29]	BLUE4	P1[29]	BLUE7
LCD_VD[22]	P1[28]	BLUE2	P1[28]	BLUE3	P1[28]	BLUE3	P1[28]	BLUE6
LCD_VD[21]	P1[27]	BLUE1	P1[27]	BLUE2	P1[27]	BLUE2	P1[27]	BLUE5
LCD_VD[20]	P1[26]	BLUE0	P1[26]	BLUE1	P1[26]	BLUE1	P1[26]	BLUE4
LCD_VD[19]	-	-	P2[13]	BLUE0	P2[13]	BLUE0	P2[13]	BLUE3
LCD_VD[18]	-	-	-	-	P2[12]	intensity	P2[12]	BLUE2
LCD_VD[17]	-	-	-	-	-	-	P0[9]	BLUE1
LCD_VD[16]	-	-	-	-	-	-	P0[8]	BLUE0
LCD_VD[15]	P1[25]	GREEN3	P1[25]	GREEN5	P1[25]	GREEN4	P1[25]	GREEN7
LCD_VD[14]	P1[24]	GREEN2	P1[24]	GREEN4	P1[24]	GREEN3	P1[24]	GREEN6
LCD_VD[13]	P1[23]	GREEN1	P1[23]	GREEN3	P1[23]	GREEN2	P1[23]	GREEN5
LCD_VD[12]	P1[22]	GREEN0	P1[22]	GREEN2	P1[22]	GREEN1	P1[22]	GREEN4
LCD_VD[11]	-	-	P1[21]	GREEN1	P1[21]	GREEN0	P1[21]	GREEN3
LCD_VD[10]	-	-	P1[20]	GREEN0	P1[20]	intensity	P1[20]	GREEN2
LCD_VD[9]	-	-	-	-	-	-	P0[7]	GREEN1
LCD_VD[8]	-	-	-	-	-	-	P0[6]	GREEN0
LCD_VD[7]	P2[9]	RED3	P2[9]	RED4	P2[9]	RED4	P2[9]	RED7
LCD_VD[6]	P2[8]	RED2	P2[8]	RED3	P2[8]	RED3	P2[8]	RED6
LCD_VD[5]	P2[7]	RED1	P2[7]	RED2	P2[7]	RED2	P2[7]	RED5
LCD_VD[4]	P2[6]	RED0	P2[6]	RED1	P2[6]	RED1	P2[6]	RED4
LCD_VD[3]	-	-	P2[12]	RED0	P4[29]	RED0	P4[29]	RED3
LCD_VD[2]	-	-	-	-	P4[28]	intensity	P4[28]	RED2
LCD_VD[1]	-	-	-	-	-	-	P0[5]	RED1
LCD_VD[0]	-	-	-	-	-	-	P0[4]	RED0
LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP	P2[5]	LCD_LP
LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M	P2[4]	LCD_ENAB_M
LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP	P2[3]	LCD_FP
LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK	P2[2]	LCD_DCLK
LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE	P2[1]	LCD_LE
LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR	P2[0]	LCD_PWR
LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN	P2[11]	LCD_CLKIN

### 13.1 How to read this chapter

---

This chapter describes the USB device controller which is present on some LPC178x/177x devices (see [Section 1.4](#) for details). On some LPC178x/177x family devices, the USB controller can also be configured for Host or OTG operation.

### 13.2 Basic configuration

---

The USB controller is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCUSB.  
**Remark:** On reset, the USB block is disabled (PCUSB = 0).
2. Clock: The USB block can be used with either the Main PLL (PLL0), or with the alternate PLL (PLL1) to obtain the USB clock. See [Section 4.5.2](#).
3. Pins: Select the required USB pins and their modes in the relevant IOCON registers ([Section 8.4.1](#)).
4. Wake-up: Activity on the USB bus port can wake up the microcontroller from Power-down mode, see [Section 4.7.8](#).
5. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
6. Initialization: See [Section 13.13](#).

### 13.3 Introduction

---

The Universal Serial Bus (USB) is a four-wire bus that supports communication between a host and one or more (up to 127) peripherals. The host controller allocates the USB bandwidth to attached devices through a token-based protocol. The bus supports hot plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The host schedules transactions in 1 ms frames. Each frame contains a Start-Of-Frame (SOF) marker and transactions that transfer data to or from device endpoints. Each device can have a maximum of 16 logical or 32 physical endpoints. There are four types of transfers defined for the endpoints. Control transfers are used to configure the device. Interrupt transfers are used for periodic data transfer. Bulk transfers are used when the rate of transfer is not critical. Isochronous transfers have guaranteed delivery time but no error correction.

For more information on the Universal Serial Bus, see the USB Implementers Forum website.

The USB device controller on the LPC178x/177x enables full-speed (12 Mb/s) data exchange with a USB host controller.

**Table 250. USB related acronyms, abbreviations, and definitions used in this chapter**

Acronym/abbreviation	Description
AHB	Advanced High-performance bus
ATLE	Auto Transfer Length Extraction
ATX	Analog Transceiver
DD	DMA Descriptor
DDP	DMA Description Pointer
DMA	Direct Memory Access
EOP	End-Of-Packet
EP	Endpoint
EP_RAM	Endpoint RAM
FS	Full Speed
LED	Light Emitting Diode
LS	Low Speed
MPS	Maximum Packet Size
NAK	Negative Acknowledge
PLL	Phase Locked Loop
RAM	Random Access Memory
SOF	Start-Of-Frame
SIE	Serial Interface Engine
SRAM	Synchronous RAM
UDCA	USB Device Communication Area
USB	Universal Serial Bus

## 13.4 Features

- Fully compliant with the USB 2.0 specification (full speed).
- Supports 32 physical (16 logical) endpoints.
- Supports Control, Bulk, Interrupt and Isochronous endpoints.
- Scalable realization of endpoints at run time.
- Endpoint maximum packet size selection (up to USB maximum specification) by software at run time.
- Supports SoftConnect and GoodLink features.
- Supports DMA transfers on all non-control endpoints.
- Allows dynamic switching between CPU controlled and DMA modes.
- Double buffer implementation for Bulk and Isochronous endpoints.

## 13.5 Fixed endpoint configuration

[Table 251](#) shows the supported endpoint configurations. Endpoints are realized and configured at run time using the Endpoint realization registers, documented in [Section 13.10.5 “Endpoint realization registers”](#).

Table 251. Fixed endpoint configuration

Logical endpoint	Physical endpoint	Endpoint type	Direction	Packet size (bytes)	Double buffer
0	0	Control	Out	8, 16, 32, 64	No
0	1	Control	In	8, 16, 32, 64	No
1	2	Interrupt	Out	1 to 64	No
1	3	Interrupt	In	1 to 64	No
2	4	Bulk	Out	8, 16, 32, 64	Yes
2	5	Bulk	In	8, 16, 32, 64	Yes
3	6	Isochronous	Out	1 to 1023	Yes
3	7	Isochronous	In	1 to 1023	Yes
4	8	Interrupt	Out	1 to 64	No
4	9	Interrupt	In	1 to 64	No
5	10	Bulk	Out	8, 16, 32, 64	Yes
5	11	Bulk	In	8, 16, 32, 64	Yes
6	12	Isochronous	Out	1 to 1023	Yes
6	13	Isochronous	In	1 to 1023	Yes
7	14	Interrupt	Out	1 to 64	No
7	15	Interrupt	In	1 to 64	No
8	16	Bulk	Out	8, 16, 32, 64	Yes
8	17	Bulk	In	8, 16, 32, 64	Yes
9	18	Isochronous	Out	1 to 1023	Yes
9	19	Isochronous	In	1 to 1023	Yes
10	20	Interrupt	Out	1 to 64	No
10	21	Interrupt	In	1 to 64	No
11	22	Bulk	Out	8, 16, 32, 64	Yes
11	23	Bulk	In	8, 16, 32, 64	Yes
12	24	Isochronous	Out	1 to 1023	Yes
12	25	Isochronous	In	1 to 1023	Yes
13	26	Interrupt	Out	1 to 64	No
13	27	Interrupt	In	1 to 64	No
14	28	Bulk	Out	8, 16, 32, 64	Yes
14	29	Bulk	In	8, 16, 32, 64	Yes
15	30	Bulk	Out	8, 16, 32, 64	Yes
15	31	Bulk	In	8, 16, 32, 64	Yes

### 13.6 Functional description

The architecture of the USB device controller is shown below in [Figure 39](#).

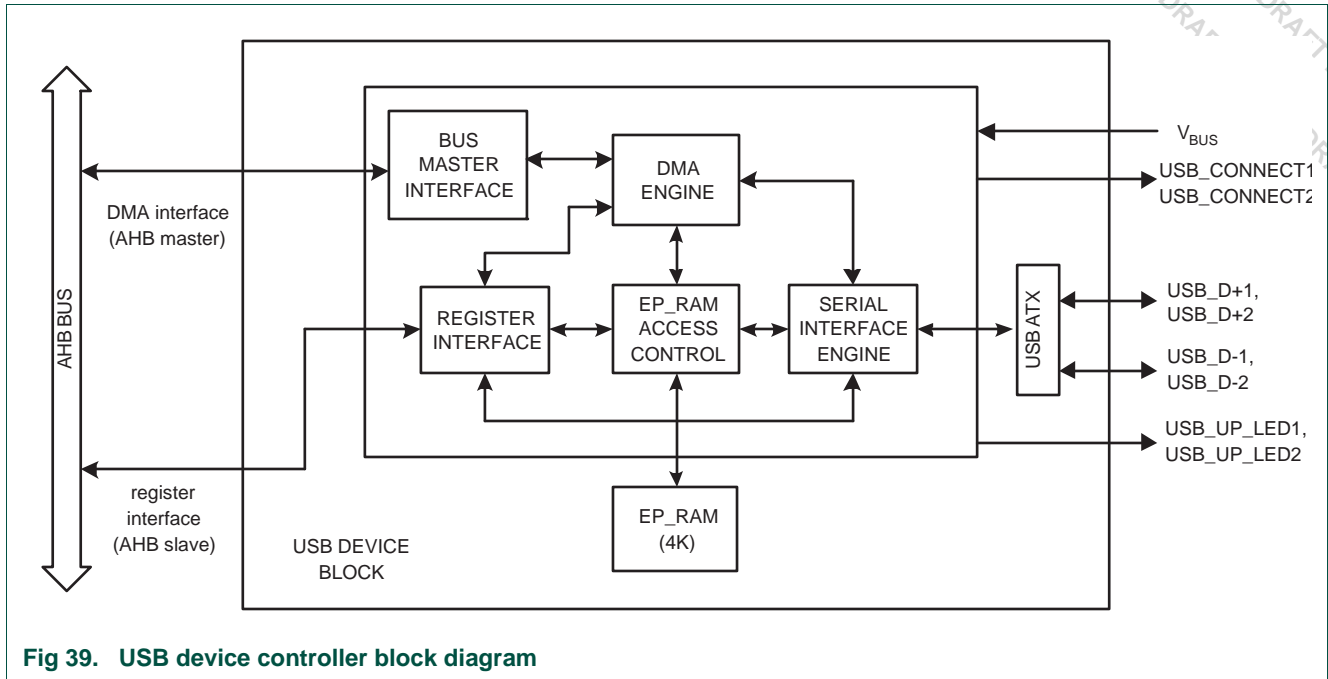


Fig 39. USB device controller block diagram

### 13.6.1 Analog transceiver

The USB Device Controller has a built-in analog transceiver (ATX). The USB ATX sends/receives the bidirectional D+ and D- signals of the USB bus.

### 13.6.2 Serial Interface Engine (SIE)

The SIE implements the full USB protocol layer. It is completely hardwired for speed and needs no firmware intervention. It handles transfer of data between the endpoint buffers in EP\_RAM and the USB bus. The functions of this block include: synchronization pattern recognition, parallel/serial conversion, bit stuffing/de-stuffing, CRC checking/generation, PID verification/generation, address recognition, and handshake evaluation/generation.

### 13.6.3 Endpoint RAM (EP\_RAM)

Each endpoint buffer is implemented as an SRAM based FIFO. The SRAM dedicated for this purpose is called the EP\_RAM. Each realized endpoint has a reserved space in the EP\_RAM. The total EP\_RAM space required depends on the number of realized endpoints, the maximum packet size of the endpoint, and whether the endpoint supports double buffering.

### 13.6.4 EP\_RAM access control

The EP\_RAM Access Control logic handles transfer of data from/to the EP\_RAM and the three sources that can access it: the CPU (via the Register Interface), the SIE, and the DMA Engine.

### 13.6.5 DMA engine and bus master interface

When enabled for an endpoint, the DMA Engine transfers data between RAM on the AHB bus and the endpoint's buffer in EP\_RAM. A single DMA channel is shared between all endpoints. When transferring data, the DMA Engine functions as a master on the AHB bus through the bus master interface.

### 13.6.6 Register interface

The Register Interface allows the CPU to control the operation of the USB Device Controller. It also provides a way to write transmit data to the controller and read receive data from the controller.

### 13.6.7 SoftConnect

The connection to the USB is accomplished by bringing D+ (for a full-speed device) HIGH through a 1.5 kOhm pull-up resistor. The SoftConnect feature can be used to allow software to finish its initialization sequence before deciding to establish connection to the USB. Re-initialization of the USB bus connection can also be performed without having to unplug the cable.

To use the SoftConnect feature, the CONNECT signal should control an external switch that connects the 1.5 kOhm resistor between D+ and +3.3V. Software can then control the CONNECT signal by writing to the CON bit using the SIE Set Device Status command.

### 13.6.8 GoodLink

Good USB connection indication is provided through GoodLink technology. When the device is successfully enumerated and configured, the LED indicator will be permanently ON. During suspend, the LED will be OFF.

This feature provides a user-friendly indicator on the status of the USB device. It is a useful field diagnostics tool to isolate faulty equipment.

To use the GoodLink feature the UP\_LED signal should control an LED. The UP\_LED signal is controlled using the SIE Configure Device command.

## 13.7 Operational overview

Transactions on the USB bus transfer data between device endpoints and the host. The direction of a transaction is defined with respect to the host. OUT transactions transfer data from the host to the device. IN transactions transfer data from the device to the host. All transactions are initiated by the host controller.

For an OUT transaction, the USB ATX receives the bidirectional D+ and D- signals of the USB bus. The Serial Interface Engine (SIE) receives the serial data from the ATX and converts it into a parallel data stream. The parallel data is written to the corresponding endpoint buffer in the EP\_RAM.

For IN transactions, the SIE reads the parallel data from the endpoint buffer in EP\_RAM, converts it into serial data, and transmits it onto the USB bus using the USB ATX.



Once data has been received or sent, the endpoint buffer can be read or written. How this is accomplished depends on the endpoint's type and operating mode. The two operating modes for each endpoint are Slave (CPU-controlled) mode, and DMA mode.

In Slave mode, the CPU transfers data between RAM and the endpoint buffer using the Register Interface. See [Section 13.14 "Slave mode operation"](#) for a detailed description of this mode.

In DMA mode, the DMA transfers data between RAM and the endpoint buffer. See [Section 13.15 "DMA operation"](#) for a detailed description of this mode.

## 13.8 Pin description

**Table 252. USB external interface**

Name	Direction	Description
V <sub>BUS</sub>	I	V <sub>BUS</sub> status input. When this function is not enabled via its corresponding IOCON register, it is driven HIGH internally.
USB_CONNECT1, USB_CONNECT2	O	SoftConnect control signal.
USB_UP_LED1, USB_UP_LED2	O	GoodLink LED control signal.
USB_D+1, USB_D+2	I/O	Positive differential data.
USB_D-1, USB_D-2	I/O	Negative differential data.

## 13.9 Clocking and power management

This section describes the clocking and power management features of the USB Device Controller.

### 13.9.1 Power requirements

The USB protocol insists on power management by the device. This becomes very critical if the device draws power from the bus (bus-powered device). The following constraints should be met by a bus-powered device:

1. A device in the non-configured state should draw a maximum of 100 mA from the bus.
2. A configured device can draw only up to what is specified in the Max Power field of the configuration descriptor. The maximum value is 500 mA.
3. A suspended device can draw a maximum of 500  $\mu$ A.

### 13.9.2 Clocks

The USB device controller clocks are shown in [Table 253](#)

**Table 253. USB device controller clock sources**

Clock source	Description
AHB master clock	Clock for the AHB master bus interface and DMA
AHB slave clock	Clock for the AHB slave interface
usbclk	48 MHz clock from the dedicated Alt PLL (PLL1) or the Main PLL (PLL0), used to recover the 12 MHz clock from the USB bus

### 13.9.3 Power management support

To help conserve power, the USB device controller automatically disables the AHB master clock and usbclk when not in use.

When the USB Device Controller goes into the suspend state (bus is idle for 3 ms), the usbclk input to the device controller is automatically disabled, helping to conserve power. However, if software wishes to access the device controller registers, usbclk must be active. To allow access to the device controller registers while in the suspend state, the USBClkCtrl and USBClkSt registers are provided.

When software wishes to access the device controller registers, it should first ensure usbclk is enabled by setting DEV\_CLK\_EN in the USBClkCtrl register, and then poll the corresponding DEV\_CLK\_ON bit in USBClkSt until set. Once set, usbclk will remain enabled until DEV\_CLK\_EN is cleared by software.

When a DMA transfer occurs, the device controller automatically turns on the AHB master clock. Once asserted, it remains active for a minimum of 2 ms (2 frames), to help ensure that DMA throughput is not affected by turning off the AHB master clock. 2 ms after the last DMA access, the AHB master clock is automatically disabled to help conserve power. If desired, software also has the capability of forcing this clock to remain enabled using the USBClkCtrl register.

Note that the AHB slave clock is always enabled as long as the PCUSB bit of PCONP is set. When the device controller is not in use, all of the device controller clocks may be disabled by clearing PCUSB.

The USB\_NEED\_CLK signal is used to facilitate going into and waking up from chip Power-down mode. USB\_NEED\_CLK is asserted if any of the bits of the USBClkSt register are asserted.

After entering the suspend state with DEV\_CLK\_EN and AHB\_CLK\_EN cleared, the DEV\_CLK\_ON and AHB\_CLK\_ON will be cleared when the corresponding clock turns off. When both bits are zero, USB\_NEED\_CLK will be low, indicating that the chip can be put into Power-down mode by writing to the PCON register. The status of USB\_NEED\_CLK can be read from the USBIntSt register.

Any bus activity in the suspend state will cause the USB\_NEED\_CLK signal to be asserted. When the chip is in Power-down mode and the USB interrupt is enabled, the assertion of USB\_NEED\_CLK causes the chip to wake up from Power-down mode.

### 13.9.4 Remote wake-up

The USB device controller supports software initiated remote wake-up. Remote wake-up involves resume signaling on the USB bus initiated from the device. This is done by clearing the SUS bit in the SIE Set Device Status register. Before writing into the register, all the clocks to the device controller have to be enabled using the USBClkCtrl register.

## 13.10 Register description

[Table 254](#) shows the USB Device Controller registers directly accessible by the CPU. The Serial Interface Engine (SIE) has other registers that are indirectly accessible via the SIE command registers. See [Section 13.12 “Serial interface engine command description”](#) for more info.

**Table 254. USB device register map**

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
<b>Port select register</b>					
USBPortSel <sup>[2]</sup>	USB Port Select	R/W	0	0x2008 C110	<a href="#">255</a>
<b>Clock control registers</b>					
USBClkCtrl	USB Clock Control	R/W	0	0x2008 CFF4	<a href="#">256</a>
USBClkSt	USB Clock Status	RO	0	0x2008 CFF8	<a href="#">257</a>
<b>Device interrupt registers</b>					
USBIntSt	USB Interrupt Status	R/W	0x8000 0000	0x400F C1C0	<a href="#">258</a>
USBDevIntSt	USB Device Interrupt Status	RO	0x10	0x2008 C200	<a href="#">259</a>
USBDevIntEn	USB Device Interrupt Enable	R/W	0	0x2008 C204	<a href="#">261</a>
USBDevIntClr	USB Device Interrupt Clear	WO	0	0x2008 C208	<a href="#">263</a>
USBDevIntSet	USB Device Interrupt Set	WO	0	0x2008 C20C	<a href="#">265</a>
USBDevIntPri	USB Device Interrupt Priority	WO	0	0x2008 C22C	<a href="#">267</a>
<b>Endpoint interrupt registers</b>					
USBEpIntSt	USB Endpoint Interrupt Status	RO	0	0x2008 C230	<a href="#">268</a>
USBEpIntEn	USB Endpoint Interrupt Enable	R/W	0	0x2008 C234	<a href="#">270</a>
USBEpIntClr	USB Endpoint Interrupt Clear	WO	0	0x2008 C238	<a href="#">272</a>
USBEpIntSet	USB Endpoint Interrupt Set	WO	0	0x2008 C23C	<a href="#">274</a>
USBEpIntPri	USB Endpoint Priority	WO <sup>[3]</sup>	0	0x2008 C240	<a href="#">276</a>
<b>Endpoint realization registers</b>					
USBRReEp	USB Realize Endpoint	R/W	0x3	0x2008 C244	<a href="#">278</a>
USBEpIn	USB Endpoint Index	WO <sup>[3]</sup>	0	0x2008 C248	<a href="#">280</a>
USBMaxPSize	USB MaxPacketSize	R/W	0x8	0x2008 C24C	<a href="#">281</a>
<b>USB transfer registers</b>					
USBRxData	USB Receive Data	RO	0	0x2008 C218	<a href="#">282</a>
USBRxPLen	USB Receive Packet Length	RO	0	0x2008 C220	<a href="#">283</a>
USBTxData	USB Transmit Data	WO <sup>[3]</sup>	0	0x2008 C21C	<a href="#">284</a>
USBTxPLen	USB Transmit Packet Length	WO <sup>[3]</sup>	0	0x2008 C224	<a href="#">285</a>
USBCtrl	USB Control	R/W	0	0x2008 C228	<a href="#">286</a>
<b>SIE Command registers</b>					

Table 254. USB device register map

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
USBCmdCode	USB Command Code	WO <sup>[3]</sup>	0	0x2008 C210	<a href="#">287</a>
USBCmdData	USB Command Data	RO	0	0x2008 C214	<a href="#">288</a>
<b>DMA registers</b>					
USBDMARSt	USB DMA Request Status	RO	0	0x2008 C250	<a href="#">289</a>
USBDMARClr	USB DMA Request Clear	WO <sup>[3]</sup>	0	0x2008 C254	<a href="#">291</a>
USBDMARSet	USB DMA Request Set	WO <sup>[3]</sup>	0	0x2008 C258	<a href="#">292</a>
USBUDCAH	USB UDCA Head	R/W	0	0x2008 C280	<a href="#">293</a>
USBEPDMASt	USB Endpoint DMA Status	RO	0	0x2008 C284	<a href="#">294</a>
USBEPDMAEn	USB Endpoint DMA Enable	WO <sup>[3]</sup>	0	0x2008 C288	<a href="#">295</a>
USBEPDMADis	USB Endpoint DMA Disable	WO <sup>[3]</sup>	0	0x2008 C28C	<a href="#">296</a>
USBDMAIntSt	USB DMA Interrupt Status	RO	0	0x2008 C290	<a href="#">297</a>
USBDMAIntEn	USB DMA Interrupt Enable	R/W	0	0x2008 C294	<a href="#">298</a>
USBEoTIntSt	USB End of Transfer Interrupt Status	RO	0	0x2008 C2A0	<a href="#">299</a>
USBEoTIntClr	USB End of Transfer Interrupt Clear	WO <sup>[3]</sup>	0	0x2008 C2A4	<a href="#">300</a>
USBEoTIntSet	USB End of Transfer Interrupt Set	WO <sup>[3]</sup>	0	0x2008 C2A8	<a href="#">301</a>
USBNDDRIntSt	USB New DD Request Interrupt Status	RO	0	0x2008 C2AC	<a href="#">302</a>
USBNDDRIntClr	USB New DD Request Interrupt Clear	WO <sup>[3]</sup>	0	0x2008 C2B0	<a href="#">303</a>
USBNDDRIntSet	USB New DD Request Interrupt Set	WO <sup>[3]</sup>	0	0x2008 C2B4	<a href="#">304</a>
USBSysErrIntSt	USB System Error Interrupt Status	RO	0	0x2008 C2B8	<a href="#">305</a>
USBSysErrIntClr	USB System Error Interrupt Clear	WO <sup>[3]</sup>	0	0x2008 C2BC	<a href="#">306</a>
USBSysErrIntSet	USB System Error Interrupt Set	WO <sup>[3]</sup>	0	0x2008 C2C0	<a href="#">307</a>

- [1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.
- [2] The USBPortSel register is identical to the OTGStCtrl register (see [Section 15.8.6](#)). In device-only operations only bits 0 and 1 of this register are used to control the routing of USB pins to Port 1 or Port 2.
- [3] Reading WO register will return an invalid value.

### 13.10.1 Port select register

#### 13.10.1.1 USB Port Select register (USBPortSel - 0x2008 C110)

This register selects the USB port pins that the USB device signals are routed to. USBPortSel is a read/write register.

Table 255. USB Port Select register (USBPortSel - 0x2008 C110) bit description

Bit	Symbol	Value	Description	Reset value
1:0	PORTSEL		Selects which USB port the device controller signals are mapped to.	0
		0x0	The USB device controller signals are mapped to the U1 port: USB_CONNECT1, USB_UP_LED1, USB_D+1, USB_D-1.	
		0x3	The USB device controller signals are mapped to the U2 port: USB_CONNECT2, USB_UP_LED2, USB_D+2, USB_D-2.	
		Others	Reserved.	
31:2	-		Reserved. Read value is undefined, only zero should be written.	NA

### 13.10.2 Clock control registers

#### 13.10.2.1 USB Clock Control register (USBClkCtrl - 0x2008 CFF4)

This register controls the clocking of the USB Device Controller. Whenever software wants to access the device controller registers, both DEV\_CLK\_EN and AHB\_CLK\_EN must be set. The PORTSEL\_CLK\_EN bit need only be set when accessing the USBPortSel register.

The software does not have to repeat this exercise for every register access, provided that the corresponding USBClkCtrl bits are already set. Note that this register is functional only when the PCUSB bit of PCONP is set; when PCUSB is cleared, all clocks to the device controller are disabled irrespective of the contents of this register. USBClkCtrl is a read/write register.

**Table 256. USBClkCtrl register (USBClkCtrl - address 0x2008 CFF4) bit description**

Bit	Symbol	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	NA
1	DEV_CLK_EN	Device clock enable. Enables the usbclk input to the device controller	0
2	-	Reserved. Read value is undefined, only zero should be written.	NA
3	PORTSEL_CLK_EN	Port select register clock enable.	NA
4	AHB_CLK_EN	AHB clock enable	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 13.10.2.2 USB Clock Status register (USBClkSt - 0x2008 CFF8)

This register holds the clock availability status. The bits of this register are ORed together to form the USB\_NEED\_CLK signal. When enabling a clock via USBClkCtrl, software should poll the corresponding bit in USBClkSt. If it is set, then software can go ahead with the register access. Software does not have to repeat this exercise for every access, provided that the USBClkCtrl bits are not disturbed. USBClkSt is a read-only register.

**Table 257. USB Clock Status register (USBClkSt - address 0x2008 CFF8) bit description**

Bit	Symbol	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	NA
1	DEV_CLK_ON	Device clock on. The usbclk input to the device controller is active.	0
2	-	Reserved. Read value is undefined, only zero should be written.	NA
3	PORTSEL_CLK_ON	Port select register clock on.	NA
4	AHB_CLK_ON	AHB clock on.	0
31:5	-	Reserved. The value read from a reserved bit is not defined.	NA

### 13.10.3 Device interrupt registers

#### 13.10.3.1 USB Interrupt Status register (USBIntSt - 0x2008 C1C0)

The USB Device Controller has three interrupt lines. This register allows software to determine their status with a single read operation. All three interrupt lines are ORed together to a single channel of the vectored interrupt controller. This register also contains the USB\_NEED\_CLK status and EN\_USB\_INTS control bits. USBIntSt is a read/write register.

**Table 258. USB Interrupt Status register (USBIntSt - address 0x2008 C1C0) bit description**

Bit	Symbol	Description	Reset value
0	USB_INT_REQ_LP	Low priority interrupt line status. This bit is read-only.	0
1	USB_INT_REQ_HP	High priority interrupt line status. This bit is read-only.	0
2	USB_INT_REQ_DMA	DMA interrupt line status. This bit is read-only.	0
7:3	-	Reserved. Read value is undefined, only zero should be written.	NA
8	USB_NEED_CLK	USB need clock indicator. This bit is set to 1 when USB activity or a change of state on the USB data pins is detected, and it indicates that a PLL supplied clock of 48 MHz is needed. Once USB_NEED_CLK becomes one, it resets to zero 5 ms after the last packet has been received/sent, or 2 ms after the Suspend Change (SUS_CH) interrupt has occurred. A change of this bit from 0 to 1 can wake up the microcontroller if activity on the USB bus is selected to wake up the part from the Power-down mode (see <a href="#">Section 4.7.8 “Wake-up from Reduced Power Modes”</a> for details). Also see <a href="#">Section 4.5.8 “PLLs and Power-down mode”</a> and <a href="#">Section 4.7.9 “Power Control for Peripherals register (PCONP - 0x400F C0C4)”</a> for considerations about the PLL and invoking the Power-down mode. This bit is read-only.	1
30:9	-	Reserved. Read value is undefined, only zero should be written.	NA
31	EN_USB_INTS	Enable all USB interrupts. When this bit is cleared, the Vectored Interrupt Controller does not see the ORed output of the USB interrupt lines.	1

#### 13.10.3.2 USB Device Interrupt Status register (USBDevIntSt - 0x2008 C200)

The USBDevIntSt register holds the status of each interrupt. A 0 indicates no interrupt and 1 indicates the presence of the interrupt. USBDevIntSt is a read-only register.

**Table 259. USB Device Interrupt Status register (USBDevIntSt - address 0x2008 C200) bit allocation**

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	-	-	-	-	-	-
Bit	15	14	13	12	11	10	9	8
Symbol	-	-	-	-	-	-	ERR_INT	EP_RLZED
Bit	7	6	5	4	3	2	1	0
Symbol	TxENDPKT	Rx ENDPKT	CDFULL	CCEMPTY	DEV_STAT	EP_SLOW	EP_FAST	FRAME

**Table 260. USB Device Interrupt Status register (USBDevIntSt - address 0x2008 C200) bit description**

Bit	Symbol	Description	Reset value
0	FRAME	The frame interrupt occurs every 1 ms. This is used in isochronous packet transfers.	0
1	EP_FAST	Fast endpoint interrupt. If an Endpoint Interrupt Priority register (USBEPIntPri) bit is set, the corresponding endpoint interrupt will be routed to this bit.	0
2	EP_SLOW	Slow endpoints interrupt. If an Endpoint Interrupt Priority Register (USBEPIntPri) bit is not set, the corresponding endpoint interrupt will be routed to this bit.	0
3	DEV_STAT	Set when USB Bus reset, USB suspend change or Connect change event occurs. Refer to <a href="#">Section 13.12.6 “Set Device Status (Command: 0xFE, Data: write 1 byte)” on page 344.</a>	0
4	CCEMPTY	The command code register (USBCmdCode) is empty (New command can be written).	1
5	CDFULL	Command data register (USBCmdData) is full (Data can be read now).	0
6	RxENDPKT	The current packet in the endpoint buffer is transferred to the CPU.	0
7	TxENDPKT	The number of data bytes transferred to the endpoint buffer equals the number of bytes programmed in the TxPacket length register (USBTxPLen).	0
8	EP_RLZED	Endpoints realized. Set when Realize Endpoint register (USBReEp) or MaxPacketSize register (USBMaxPSize) is updated and the corresponding operation is completed.	0
9	ERR_INT	Error Interrupt. Any bus error interrupt from the USB device. Refer to <a href="#">Section 13.12.9 “Read Error Status (Command: 0xFB, Data: read 1 byte)” on page 346</a>	0
31:10	-	Reserved. The value read from a reserved bit is not defined.	NA

**13.10.3.3 USB Device Interrupt Enable register (USBDevIntEn - 0x2008 C204)**

Writing a one to a bit in this register enables the corresponding bit in USBDevIntSt to generate an interrupt on one of the interrupt lines when set. By default, the interrupt is routed to the USB\_INT\_REQ\_LP interrupt line. Optionally, either the EP\_FAST or FRAME interrupt may be routed to the USB\_INT\_REQ\_HP interrupt line by changing the value of USBDevIntPri. USBDevIntEn is a read/write register.

**Table 261. USB Device Interrupt Enable register (USBDevIntEn - address 0x2008 C204) bit allocation**

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	-	-	-	-	-	-
Bit	15	14	13	12	11	10	9	8
Symbol	-	-	-	-	-	-	ERR_INT	EP_RLZED
Bit	7	6	5	4	3	2	1	0
Symbol	TxENDPKT	Rx ENDPKT	CDFULL	CCEMPTY	DEV_STAT	EP_SLOW	EP_FAST	FRAME

**Table 262. USB Device Interrupt Enable register (USBDevIntEn - address 0x2008 C204) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBDevIntEn bit allocation table above	0	No interrupt is generated.	0
		1	An interrupt will be generated when the corresponding bit in the Device Interrupt Status (USBDevIntSt) register ( <a href="#">Table 259</a> ) is set. By default, the interrupt is routed to the USB_INT_REQ_LP interrupt line. Optionally, either the EP_FAST or FRAME interrupt may be routed to the USB_INT_REQ_HP interrupt line by changing the value of USBDevIntPri.	

**13.10.3.4 USB Device Interrupt Clear register (USBDevIntClr - 0x2008 C208)**

Writing one to a bit in this register clears the corresponding bit in USBDevIntSt. Writing a zero has no effect.

**Remark:** Before clearing the EP\_SLOW or EP\_FAST interrupt bits, the corresponding endpoint interrupts in USBEpIntSt should be cleared.

USBDevIntClr is a write-only register.

**Table 263. USB Device Interrupt Clear register (USBDevIntClr - address 0x2008 C208) bit allocation**

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	-	-	-	-	-	-
Bit	15	14	13	12	11	10	9	8
Symbol	-	-	-	-	-	-	ERR_INT	EP_RLZED
Bit	7	6	5	4	3	2	1	0
Symbol	TxENDPKT	Rx ENDPKT	CDFULL	CCEMPTY	DEV_STAT	EP_SLOW	EP_FAST	FRAME

**Table 264. USB Device Interrupt Clear register (USBDevIntClr - address 0x2008 C208) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBDevIntClr bit allocation table above	0	No effect.	0
		1	The corresponding bit in USBDevIntSt ( <a href="#">Section 13.10.3.2</a> ) is cleared.	

**13.10.3.5 USB Device Interrupt Set register (USBDevIntSet - 0x2008 C20C)**

Writing one to a bit in this register sets the corresponding bit in the USBDevIntSt. Writing a zero has no effect

USBDevIntSet is a write-only register.

**Table 265. USB Device Interrupt Set register (USBDevIntSet - address 0x2008 C20C) bit allocation**

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	-	-	-	-	-	-
Bit	15	14	13	12	11	10	9	8
Symbol	-	-	-	-	-	-	ERR_INT	EP_RLZED
Bit	7	6	5	4	3	2	1	0
Symbol	TxENDPKT	Rx ENDPKT	CDFULL	CCEMPTY	DEV_STAT	EP_SLOW	EP_FAST	FRAME



**Table 266. USB Device Interrupt Set register (USBDevIntSet - address 0x2008 C20C) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBDevIntSet bit allocation table above	0	No effect.	0
		1	The corresponding bit in USBDevIntSt ( <a href="#">Section 13.10.3.2</a> ) is set.	

**13.10.3.6 USB Device Interrupt Priority register (USBDevIntPri - 0x2008 C22C)**

Writing one to a bit in this register causes the corresponding interrupt to be routed to the USB\_INT\_REQ\_HP interrupt line. Writing zero causes the interrupt to be routed to the USB\_INT\_REQ\_LP interrupt line. Either the EP\_FAST or FRAME interrupt can be routed to USB\_INT\_REQ\_HP, but not both. If the software attempts to set both bits to one, no interrupt will be routed to USB\_INT\_REQ\_HP. USBDevIntPri is a write-only register.

**Table 267. USB Device Interrupt Priority register (USBDevIntPri - address 0x2008 C22C) bit description**

Bit	Symbol	Value	Description	Reset value
0	FRAME	0	FRAME interrupt is routed to USB_INT_REQ_LP.	0
		1	FRAME interrupt is routed to USB_INT_REQ_HP.	
1	EP_FAST	0	EP_FAST interrupt is routed to USB_INT_REQ_LP.	0
		1	EP_FAST interrupt is routed to USB_INT_REQ_HP.	
31:2	-		Reserved. Read value is undefined, only zero should be written.	NA

### 13.10.4 Endpoint interrupt registers

The registers in this group facilitate handling of endpoint interrupts. Endpoint interrupts are used in Slave mode operation.

#### 13.10.4.1 USB Endpoint Interrupt Status register (USBEPIntSt - 0x2008 C230)

Each physical non-isochronous endpoint is represented by a bit in this register to indicate that it has generated an interrupt. All non-isochronous OUT endpoints generate an interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet is successfully transmitted, or when a NAK handshake is sent on the bus and the interrupt on NAK feature is enabled (see [Section 13.12.3 “Set Mode \(Command: 0xF3, Data: write 1 byte\)” on page 343](#)). A bit set to one in this register causes either the EP\_FAST or EP\_SLOW bit of USBDevIntSt to be set depending on the value of the corresponding bit of USBEpDevIntPri. USBEPIntSt is a read-only register.

Note that for Isochronous endpoints, handling of packet data is done when the FRAME interrupt occurs.

**Table 268. USB Endpoint Interrupt Status register (USBEPIntSt - address 0x2008 C230) bit allocation**

Reset value: 0x0000 0000

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP15TX	EP15RX	EP14TX	EP14RX	EP13TX	EP13RX	EP12TX	EP12RX
<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP11TX	EP11RX	EP10TX	EP10RX	EP9TX	EP9RX	EP8TX	EP8RX
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP7TX	EP7RX	EP6TX	EP6RX	EP5TX	EP5RX	EP4TX	EP4RX
<b>Bit</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Symbol</b>	EP3TX	EP3RX	EP2TX	EP2RX	EP1TX	EP1RX	EP0TX	EP0RX

**Table 269. USB Endpoint Interrupt Status register (USBEPIntSt - address 0x2008 C230) bit description**

Bit	Symbol	Description	Reset value
0	EP0RX	Endpoint 0, Data Received Interrupt bit.	0
1	EP0TX	Endpoint 0, Data Transmitted Interrupt bit or sent a NAK.	0
2	EP1RX	Endpoint 1, Data Received Interrupt bit.	0
3	EP1TX	Endpoint 1, Data Transmitted Interrupt bit or sent a NAK.	0
4	EP2RX	Endpoint 2, Data Received Interrupt bit.	0
5	EP2TX	Endpoint 2, Data Transmitted Interrupt bit or sent a NAK.	0
6	EP3RX	Endpoint 3, Isochronous endpoint.	NA
7	EP3TX	Endpoint 3, Isochronous endpoint.	NA
8	EP4RX	Endpoint 4, Data Received Interrupt bit.	0
9	EP4TX	Endpoint 4, Data Transmitted Interrupt bit or sent a NAK.	0
10	EP5RX	Endpoint 5, Data Received Interrupt bit.	0
11	EP5TX	Endpoint 5, Data Transmitted Interrupt bit or sent a NAK.	0
12	EP6RX	Endpoint 6, Isochronous endpoint.	NA
13	EP6TX	Endpoint 6, Isochronous endpoint.	NA
14	EP7RX	Endpoint 7, Data Received Interrupt bit.	0

**Table 269. USB Endpoint Interrupt Status register (USBEPIntSt - address 0x2008 C230) bit description**

Bit	Symbol	Description	Reset value
15	EP7TX	Endpoint 7, Data Transmitted Interrupt bit or sent a NAK.	0
16	EP8RX	Endpoint 8, Data Received Interrupt bit.	0
17	EP8TX	Endpoint 8, Data Transmitted Interrupt bit or sent a NAK.	0
18	EP9RX	Endpoint 9, Isochronous endpoint.	NA
19	EP9TX	Endpoint 9, Isochronous endpoint.	NA
20	EP10RX	Endpoint 10, Data Received Interrupt bit.	0
21	EP10TX	Endpoint 10, Data Transmitted Interrupt bit or sent a NAK.	0
22	EP11RX	Endpoint 11, Data Received Interrupt bit.	0
23	EP11TX	Endpoint 11, Data Transmitted Interrupt bit or sent a NAK.	0
24	EP12RX	Endpoint 12, Isochronous endpoint.	NA
25	EP12TX	Endpoint 12, Isochronous endpoint.	NA
26	EP13RX	Endpoint 13, Data Received Interrupt bit.	0
27	EP13TX	Endpoint 13, Data Transmitted Interrupt bit or sent a NAK.	0
28	EP14RX	Endpoint 14, Data Received Interrupt bit.	0
29	EP14TX	Endpoint 14, Data Transmitted Interrupt bit or sent a NAK.	0
30	EP15RX	Endpoint 15, Data Received Interrupt bit.	0
31	EP15TX	Endpoint 15, Data Transmitted Interrupt bit or sent a NAK.	0

### 13.10.4.2 USB Endpoint Interrupt Enable register (USBEPIntEn - 0x2008 C234)

Setting a bit to 1 in this register causes the corresponding bit in USBEPIntSt to be set when an interrupt occurs for the associated endpoint. Setting a bit to 0 causes the corresponding bit in USBDMARSt to be set when an interrupt occurs for the associated endpoint. USBEPIntEn is a read/write register.

**Table 270. USB Endpoint Interrupt Enable register (USBEPIntEn - address 0x2008 C234) bit allocation**

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	EP15TX	EP15RX	EP14TX	EP14RX	EP13TX	EP13RX	EP12TX	EP12RX
Bit	23	22	21	20	19	18	17	16
Symbol	EP11TX	EP11RX	EP10TX	EP10RX	EP9TX	EP9RX	EP8TX	EP8RX
Bit	15	14	13	12	11	10	9	8
Symbol	EP7TX	EP7RX	EP6TX	EP6RX	EP5TX	EP5RX	EP4TX	EP4RX
Bit	7	6	5	4	3	2	1	0
Symbol	EP3TX	EP3RX	EP2TX	EP2RX	EP1TX	EP1RX	EP0TX	EP0RX

**Table 271. USB Endpoint Interrupt Enable register (USBEPIntEn - address 0x2008 C234) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBEPIntEn bit allocation table above	0	The corresponding bit in USBDMARSt is set when an interrupt occurs for this endpoint.	0
		1	The corresponding bit in USBEPIntSt is set when an interrupt occurs for this endpoint. Implies Slave mode for this endpoint.	

**13.10.4.3 USB Endpoint Interrupt Clear register (USBEpIntClr - 0x2008 C238)**

Writing a one to this a bit in this register causes the SIE Select Endpoint/Clear Interrupt command to be executed (Table 315) for the corresponding physical endpoint. Writing zero has no effect. Before executing the Select Endpoint/Clear Interrupt command, the CDFULL bit in USBDevIntSt is cleared by hardware. On completion of the command, the CDFULL bit is set, USBCmdData contains the status of the endpoint, and the corresponding bit in USBEpIntSt is cleared.

Notes:

- When clearing interrupts using USBEpIntClr, software should wait for CDFULL to be set to ensure the corresponding interrupt has been cleared before proceeding.
- While setting multiple bits in USBEpIntClr simultaneously is possible, it is not recommended; only the status of the endpoint corresponding to the least significant interrupt bit cleared will be available at the end of the operation.
- Alternatively, the SIE Select Endpoint/Clear Interrupt command can be directly invoked using the SIE command registers, but using USBEpIntClr is recommended because of its ease of use.

Each physical endpoint has its own reserved bit in this register. The bit field definition is the same as that of USBEpIntSt shown in Table 268. USBEpIntClr is a write-only register.

**Table 272. USB Endpoint Interrupt Clear register (USBEpIntClr - address 0x2008 C238) bit allocation**

Reset value: 0x0000 0000

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP15TX	EP15RX	EP14TX	EP14RX	EP13TX	EP13RX	EP12TX	EP12RX
<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP11TX	EP11RX	EP10TX	EP10RX	EP9TX	EP9RX	EP8TX	EP8RX
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP7TX	EP7RX	EP6TX	EP6RX	EP5TX	EP5RX	EP4TX	EP4RX
<b>Bit</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Symbol</b>	EP3TX	EP3RX	EP2TX	EP2RX	EP1TX	EP1RX	EP0TX	EP0RX

**Table 273. USB Endpoint Interrupt Clear register (USBEpIntClr - address 0x2008 C238) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBEpIntClr bit allocation table above	0	No effect.	0
		1	Clears the corresponding bit in USBEpIntSt, by executing the SIE Select Endpoint/Clear Interrupt command for this endpoint.	

**13.10.4.4 USB Endpoint Interrupt Set register (USBEpIntSet - 0x2008 C23C)**

Writing a one to a bit in this register sets the corresponding bit in USBEpIntSt. Writing zero has no effect. Each endpoint has its own bit in this register. USBEpIntSet is a write-only register.

**Table 274. USB Endpoint Interrupt Set register (USBEpIntSet - address 0x2008 C23C) bit allocation**

Reset value: 0x0000 0000

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP15TX	EP15RX	EP14TX	EP14RX	EP13TX	EP13RX	EP12TX	EP12RX

<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP11TX	EP11RX	EP10TX	EP10RX	EP9TX	EP9RX	EP8TX	EP8RX
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP7TX	EP7RX	EP6TX	EP6RX	EP5TX	EP5RX	EP4TX	EP4RX
<b>Bit</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Symbol</b>	EP3TX	EP3RX	EP2TX	EP2RX	EP1TX	EP1RX	EP0TX	EP0RX

**Table 275. USB Endpoint Interrupt Set register (USBEPIntSet - address 0x2008 C23C) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBEPIntSet bit allocation table above	0	No effect.	0
		1	Sets the corresponding bit in USBEPIntSt.	

**13.10.4.5 USB Endpoint Interrupt Priority register (USBEPIntPri - 0x2008 C240)**

This register determines whether an endpoint interrupt is routed to the EP\_FAST or EP\_SLOW bits of USBDevIntSt. If a bit in this register is set to one, the interrupt is routed to EP\_FAST, if zero it is routed to EP\_SLOW. Routing of multiple endpoints to EP\_FAST or EP\_SLOW is possible.

Note that the USBDevIntPri register determines whether the EP\_FAST interrupt is routed to the USB\_INT\_REQ\_HP or USB\_INT\_REQ\_LP interrupt line.

USBEPIntPri is a write-only register.

**Table 276. USB Endpoint Interrupt Priority register (USBEPIntPri - address 0x2008 C240) bit allocation**

Reset value: 0x0000 0000

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP15TX	EP15RX	EP14TX	E14RX	EP13TX	EP13RX	EP12TX	EP12RX
<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP11TX	EP11RX	EP10TX	EP10RX	EP9TX	EP9RX	EP8TX	EP8RX
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP7TX	EP7RX	EP6TX	EP6RX	EP5TX	EP5RX	EP4TX	EP4RX
<b>Bit</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Symbol</b>	EP3TX	EP3RX	EP2TX	EP2RX	EP1TX	EP1RX	EP0TX	EP0RX

**Table 277. USB Endpoint Interrupt Priority register (USBEPIntPri - address 0x2008 C240) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	See USBEPIntPri bit allocation table above	0	The corresponding interrupt is routed to the EP_SLOW bit of USBDevIntSt	0
		1	The corresponding interrupt is routed to the EP_FAST bit of USBDevIntSt	

**13.10.5 Endpoint realization registers**

The registers in this group allow realization and configuration of endpoints at run time.

**13.10.5.1 EP RAM requirements**

The USB device controller uses a RAM based FIFO for each endpoint buffer. The RAM dedicated for this purpose is called the Endpoint RAM (EP\_RAM). Each endpoint has space reserved in the EP\_RAM. The EP\_RAM space required for an endpoint depends on its MaxPacketSize and whether it is double buffered. 32 words of EP\_RAM are used by the device for storing the endpoint buffer pointers. The EP\_RAM is word aligned but the MaxPacketSize is defined in bytes hence the RAM depth has to be adjusted to the next word boundary. Also, each buffer has one word header showing the size of the packet length received.

The EP\_RAM space (in words) required for the physical endpoint can be expressed as

$$EPRAMspace = \left( \frac{MaxPacketSize + 3}{4} + 1 \right) \times dbstatus$$

where dbstatus = 1 for a single buffered endpoint and 2 for double a buffered endpoint.

Since all the realized endpoints occupy EP\_RAM space, the total EP\_RAM requirement is

$$TotalEPRAMspace = 32 + \sum_{n=0}^N EPRAMspace(n)$$

where N is the number of realized endpoints. Total EP\_RAM space should not exceed 4096 bytes (4 kB, 1 kwords).

**13.10.5.2 USB Realize Endpoint register (USBReEp - 0x2008 C244)**

Writing one to a bit in this register causes the corresponding endpoint to be realized. Writing zeros causes it to be unrealized. This register returns to its reset state when a bus reset occurs. USBReEp is a read/write register.

**Table 278. USB Realize Endpoint register (USBReEp - address 0x2008 C244) bit allocation**

Reset value: 0x0000 0003

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP31	EP30	EP29	EP28	EP27	EP26	EP25	EP24
<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP23	EP22	EP21	EP20	EP19	EP18	EP17	EP16
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8

Bit	7	6	5	4	3	2	1	0
Symbol	EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0

Table 279. USB Realize Endpoint register (USBReEp - address 0x2008 C244) bit description

Bit	Symbol	Value	Description	Reset value
0	EP0	0	Control endpoint EP0 is not realized.	1
		1	Control endpoint EP0 is realized.	
1	EP1	0	Control endpoint EP1 is not realized.	1
		1	Control endpoint EP1 is realized.	
31:2	EPxx	0	Endpoint EPxx is not realized.	0
		1	Endpoint EPxx is realized.	

On reset, only the control endpoints are realized. Other endpoints, if required, are realized by programming the corresponding bits in USBReEp. To calculate the required EP\_RAM space for the realized endpoints, see [Section 13.10.5.1](#).

Realization of endpoints is a multi-cycle operation. Pseudo code for endpoint realization is shown below.

```

Clear EP_RLZED bit in USBDevIntSt;

for every endpoint to be realized,
{
    /* OR with the existing value of the Realize Endpoint register */
    USBReEp |= (UInt32) ((0x1 << endpt));
    /* Load Endpoint index Reg with physical endpoint no.*/
    USBEpIn = (UInt32) endpointnumber;

    /* load the max packet size Register */
    USBEpMaxPSize = MPS;

    /* check whether the EP_RLZED bit in the Device Interrupt Status register is set
    */
    while (!(USBDevIntSt & EP_RLZED))
    {
        /* wait until endpoint realization is complete */
    }
    /* Clear the EP_RLZED bit */
    Clear EP_RLZED bit in USBDevIntSt;
}

```

The device will not respond to any transactions to unrealized endpoints. The SIE Configure Device command will only cause realized and enabled endpoints to respond to transactions. For details see [Table 310](#).

### 13.10.5.3 USB Endpoint Index register (USBEPIn - 0x2008 C248)

Each endpoint has a register carrying the MaxPacketSize value for that endpoint. This is in fact a register array. Hence before writing, this register is addressed through the USBEPIn register.

The USBEPIn register will hold the physical endpoint number. Writing to USBMaxPSize will set the array element pointed to by USBEPIn. USBEPIn is a write-only register.

**Table 280. USB Endpoint Index register (USBEPIn - address 0x2008 C248) bit description**

Bit	Symbol	Description	Reset value
4:0	PHY_EP	Physical endpoint number (0-31)	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	NA

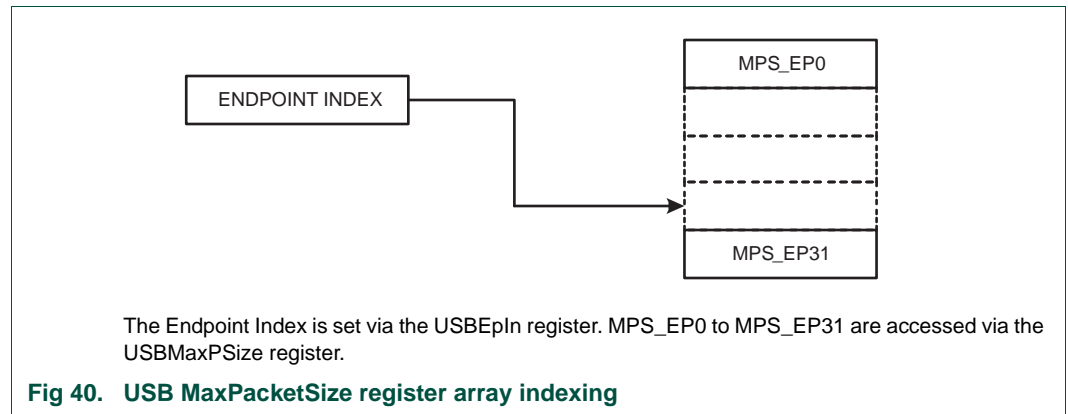
**13.10.5.4 USB MaxPacketSize register (USBMaxPSize - 0x2008 C24C)**

On reset, the control endpoint is assigned the maximum packet size of 8 bytes. Other endpoints are assigned 0. Modifying USBMaxPSize will cause the endpoint buffer addresses within the EP\_RAM to be recalculated. This is a multi-cycle process. At the end, the EP\_RLZED bit will be set in USBDevIntSt (Table 259). USBMaxPSize array indexing is shown in Figure 40. USBMaxPSize is a read/write register.

**Table 281. USB MaxPacketSize register (USBMaxPSize - address 0x2008 C24C) bit description**

Bit	Symbol	Description	Reset value
9:0	MPS	The maximum packet size value.	0x008 <sup>[1]</sup>
31:10	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Reset value for EP0 and EP1. All other endpoints have a reset value of 0x0.





### 13.10.6 USB transfer registers

The registers in this group are used for transferring data between endpoint buffers and RAM in Slave mode operation. See [Section 13.14 "Slave mode operation"](#).

#### 13.10.6.1 USB Receive Data register (USBRxData - 0x2008 C218)

For an OUT transaction, the CPU reads the endpoint buffer data from this register. Before reading this register, the RD\_EN bit and LOG\_ENDPOINT field of the USBCtrl register should be set appropriately. On reading this register, data from the selected endpoint buffer is fetched. The data is in little endian format: the first byte received from the USB bus will be available in the least significant byte of USBRxData. USBRxData is a read-only register.

**Table 282. USB Receive Data register (USBRxData - address 0x2008 C218) bit description**

Bit	Symbol	Description	Reset value
31:0	RX_DATA	Data received.	0

#### 13.10.6.2 USB Receive Packet Length register (USBRxPLen - 0x2008 C220)

This register contains the number of bytes remaining in the endpoint buffer for the current packet being read via the USBRxData register, and a bit indicating whether the packet is valid or not. Before reading this register, the RD\_EN bit and LOG\_ENDPOINT field of the USBCtrl register should be set appropriately. This register is updated on each read of the USBRxData register. USBRxPLen is a read-only register.

**Table 283. USB Receive Packet Length register (USBRxPLen - address 0x2008 C220) bit description**

Bit	Symbol	Value	Description	Reset value
9:0	PKT_LNGTH	-	The remaining number of bytes to be read from the currently selected endpoint's buffer. When this field decrements to 0, the RxENDPKT bit will be set in USBDevIntSt.	0
10	DV		Data valid. This bit is useful for isochronous endpoints. Non-isochronous endpoints do not raise an interrupt when an erroneous data packet is received. But invalid data packet can be produced with a bus reset. For isochronous endpoints, data transfer will happen even if an erroneous packet is received. In this case DV bit will not be set for the packet.	0
		0	Data is invalid.	
		1	Data is valid.	
11	PKT_RDY	-	The PKT_LNGTH field is valid and the packet is ready for reading.	0
31:12	-	-	Reserved. The value read from a reserved bit is not defined.	NA

#### 13.10.6.3 USB Transmit Data register (USBTxData - 0x2008 C21C)

For an IN transaction, the CPU writes the endpoint data into this register. Before writing to this register, the WR\_EN bit and LOG\_ENDPOINT field of the USBCtrl register should be set appropriately, and the packet length should be written to the USBTxPLen register. On writing this register, the data is written to the selected endpoint buffer. The data is in little endian format: the first byte sent on the USB bus will be the least significant byte of USBTxData. USBTxData is a write-only register.

**Table 284. USB Transmit Data register (USBTxData - address 0x2008 C21C) bit description**

Bit	Symbol	Description	Reset value
31:0	TX_DATA	Transmit Data.	0

**13.10.6.4 USB Transmit Packet Length register (USBTxPLeN - 0x2008 C224)**

This register contains the number of bytes transferred from the CPU to the selected endpoint buffer. Before writing data to USBTxData, software should first write the packet length ( $\leq$  MaxPacketSize) to this register. After each write to USBTxData, hardware decrements USBTxPLeN by 4. The WR\_EN bit and LOG\_ENDPOINT field of the USBCtrl register should be set to select the desired endpoint buffer before starting this process.

For data buffers larger than the endpoint's MaxPacketSize, software should submit data in packets of MaxPacketSize, and send the remaining extra bytes in the last packet. For example, if the MaxPacketSize is 64 bytes and the data buffer to be transferred is of length 130 bytes, then the software sends two 64-byte packets and the remaining 2 bytes in the last packet. So, a total of 3 packets are sent on USB. USBTxPLeN is a write-only register.

**Table 285. USB Transmit Packet Length register (USBTxPLeN - address 0x2008 C224) bit description**

Bit	Symbol	Value	Description	Reset value
9:0	PKT_LNGTH	-	The remaining number of bytes to be written to the selected endpoint buffer. This field is decremented by 4 by hardware after each write to USBTxData. When this field decrements to 0, the TxENDPKT bit will be set in USBDevIntSt.	0
31:10	-	-	Reserved. Read value is undefined, only zero should be written.	NA

**13.10.6.5 USB Control register (USBCtrl - 0x2008 C228)**

This register controls the data transfer operation of the USB device. It selects the endpoint buffer that is accessed by the USBRxData and USBTxData registers, and enables reading and writing them. USBCtrl is a read/write register.

**Table 286. USB Control register (USBCtrl - address 0x2008 C228) bit description**

Bit	Symbol	Value	Description	Reset value
0	RD_EN		Read mode control. Enables reading data from the OUT endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBRxData register. This bit is cleared by hardware when the last word of the current packet is read from USBRxData.	0
		0	Read mode is disabled.	
		1	Read mode is enabled.	
1	WR_EN		Write mode control. Enables writing data to the IN endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBTxData register. This bit is cleared by hardware when the number of bytes in USBTxLen have been sent.	0
		0	Write mode is disabled.	
		1	Write mode is enabled.	
5:2	LOG_ENDPOINT	-	Logical Endpoint number.	0
31:6	-	-	Reserved. Read value is undefined, only zero should be written.	NA

**13.10.7 SIE command code registers**

The SIE command code registers are used for communicating with the Serial Interface Engine. See [Section 13.12 “Serial interface engine command description”](#) for more information.

**13.10.7.1 USB Command Code register (USBCmdCode - 0x2008 C210)**

This register is used for sending the command and write data to the SIE. The commands written here are propagated to the SIE and executed there. After executing the command, the register is empty, and the CCEMPTY bit of USBDevIntSt register is set. See [Section 13.12](#) for details. USBCmdCode is a write-only register.

**Table 287. USB Command Code register (USBCmdCode - address 0x2008 C210) bit description**

Bit	Symbol	Value	Description	Reset value
7:0	-		Reserved. Read value is undefined, only zero should be written.	NA
15:8	CMD_PHASE		The command phase:	0
		0x02	Read	
		0x01	Write	
		0x05	Command	
23:16	CMD_CODE/ CMD_WDATA		This is a multi-purpose field. When CMD_PHASE is Command or Read, this field contains the code for the command (CMD_CODE). When CMD_PHASE is Write, this field contains the command write data (CMD_WDATA).	0
31:24	-		Reserved. Read value is undefined, only zero should be written.	NA

**13.10.7.2 USB Command Data register (USBCmdData - 0x2008 C214)**

This register contains the data retrieved after executing a SIE command. When the data is ready to be read, the CD\_FULLL bit of the USBDevIntSt register is set. See [Table 259](#) for details. USBCmdData is a read-only register.

**Table 288. USB Command Data register (USBCmdData - address 0x2008 C214) bit description**

Bit	Symbol	Description	Reset value
7:0	CMD_RDATA	Command Read Data.	0
31:8	-	Reserved. The value read from a reserved bit is not defined.	NA

### 13.10.8 DMA registers

The registers in this group are used for the DMA mode of operation (see [Section 13.15 “DMA operation”](#))

#### 13.10.8.1 USB DMA Request Status register (USBDMARSt - 0x2008 C250)

A bit in this register associated with a non-isochronous endpoint is set by hardware when an endpoint interrupt occurs (see the description of USBEpIntSt) and the corresponding bit in USBEpIntEn is 0. A bit associated with an isochronous endpoint is set when the corresponding bit in USBEpIntEn is 0 and a FRAME interrupt occurs. A set bit serves as a flag for the DMA engine to start the data transfer if the DMA is enabled for the corresponding endpoint in the USBEpDMASt register. The DMA cannot be enabled for control endpoints (EP0 and EP1). USBDMARSt is a read-only register.

**Table 289. USB DMA Request Status register (USBDMARSt - address 0x2008 C250) bit allocation**

Reset value: 0x0000 0000

<b>Bit</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>
<b>Symbol</b>	EP31	EP30	EP29	EP28	EP27	EP26	EP25	EP24
<b>Bit</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>
<b>Symbol</b>	EP23	EP22	EP21	EP20	EP19	EP18	EP17	EP16
<b>Bit</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>
<b>Symbol</b>	EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8
<b>Bit</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Symbol</b>	EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0

**Table 290. USB DMA Request Status register (USBDMARSt - address 0x2008 C250) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and EP0 bit must be 0).	0
1	EP1	0	Control endpoint IN (DMA cannot be enabled for this endpoint and EP1 bit must be 0).	0
31:2	EPxx		Endpoint xx ( $2 \leq xx \leq 31$ ) DMA request.	0
		0	DMA not requested by endpoint xx.	
		1	DMA requested by endpoint xx.	

[1] DMA can not be enabled for this endpoint and the corresponding bit in the USBDMARSt must be 0.

#### 13.10.8.2 USB DMA Request Clear register (USBDMARClr - 0x2008 C254)

Writing one to a bit in this register will clear the corresponding bit in the USBDMARSt register. Writing zero has no effect.

This register is intended for initialization prior to enabling the DMA for an endpoint. When the DMA is enabled for an endpoint, hardware clears the corresponding bit in USBDMARSt on completion of a packet transfer. Therefore, software should not clear the bit using this register while the endpoint is enabled for DMA operation.

USBDMARClr is a write-only register.

The USBDMARClr bit allocation is identical to the USBDMARSt register ([Table 289](#)).

**Table 291. USB DMA Request Clear register (USBDMARClr - address 0x2008 C254) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0 bit must be 0).	0
1	EP1	0	Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1 bit must be 0).	0
31:2	EPxx		Clear the endpoint xx ( $2 \leq xx \leq 31$ ) DMA request.	0
		0	No effect.	
		1	Clear the corresponding bit in USBDMARSt.	

### 13.10.8.3 USB DMA Request Set register (USBDMARSet - 0x2008 C258)

Writing one to a bit in this register sets the corresponding bit in the USBDMARSt register. Writing zero has no effect.

This register allows software to raise a DMA request. This can be useful when switching from Slave to DMA mode of operation for an endpoint: if a packet to be processed in DMA mode arrives before the corresponding bit of USBEPIntEn is cleared, the DMA request is not raised by hardware. Software can then use this register to manually start the DMA transfer.

Software can also use this register to initiate a DMA transfer to proactively fill an IN endpoint buffer before an IN token packet is received from the host.

USBDMARSet is a write-only register.

The USBDMARSet bit allocation is identical to the USBDMARSt register ([Table 289](#)).

**Table 292. USB DMA Request Set register (USBDMARSet - address 0x2008 C258) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0 bit must be 0).	0
1	EP1	0	Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1 bit must be 0).	0
31:2	EPxx		Set the endpoint xx ( $2 \leq xx \leq 31$ ) DMA request.	0
		0	No effect.	
		1	Set the corresponding bit in USBDMARSt.	

### 13.10.8.4 USB UDCA Head register (USBUDCAH - 0x2008 C280)

The UDCA (USB Device Communication Area) Head register maintains the address where the UDCA is located in the RAM. Refer to [Section 13.15.2 “USB device communication area”](#) and [Section 13.15.4 “The DMA descriptor”](#) for more details on the UDCA and DMA descriptors. USBUDCAH is a read/write register.

**Table 293. USB UDCA Head register (USBUDCAH - address 0x2008 C280) bit description**

Bit	Symbol	Description	Reset value
6:0	-	Reserved. Read value is undefined, only zero should be written. The UDCA is aligned to 128-byte boundaries.	0
31:7	UDCA_ADDR	Start address of the UDCA.	0

**13.10.8.5 USB EP DMA Status register (USBEPDMASt - 0x2008 C284)**

Bits in this register indicate whether DMA operation is enabled for the corresponding endpoint. A DMA transfer for an endpoint can start only if the corresponding bit is set in this register. USBEPDMASt is a read-only register.

**Table 294. USB EP DMA Status register (USBEPDMASt - address 0x2008 C284) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0_DMA_ENABLE	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_ENABLE bit must be 0).	0
1	EP1_DMA_ENABLE	0	Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_ENABLE bit must be 0).	0
31:2	EPxx_DMA_ENABLE		endpoint xx (2 ≤ xx ≤ 31) DMA enabled bit.	0
		0	The DMA for endpoint EPxx is disabled.	
		1	The DMA for endpoint EPxx is enabled.	

**13.10.8.6 USB EP DMA Enable register (USBEPDMAEn - 0x2008 C288)**

Writing one to a bit to this register will enable the DMA operation for the corresponding endpoint. Writing zero has no effect. The DMA cannot be enabled for control endpoints EP0 and EP1. USBEPDMAEn is a write-only register.

**Table 295. USB EP DMA Enable register (USBEPDMAEn - address 0x2008 C288) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0_DMA_ENABLE	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_ENABLE bit value must be 0).	0
1	EP1_DMA_ENABLE	0	Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_ENABLE bit must be 0).	0
31:2	EPxx_DMA_ENABLE		Endpoint xx(2 ≤ xx ≤ 31) DMA enable control bit.	0
		0	No effect.	
		1	Enable the DMA operation for endpoint EPxx.	

**13.10.8.7 USB EP DMA Disable register (USBEPDMADis - 0x2008 C28C)**

Writing a one to a bit in this register clears the corresponding bit in USBEPDMASt. Writing zero has no effect on the corresponding bit of USBEPDMASt. Any write to this register clears the internal DMA\_PROCEED flag. Refer to [Section 13.15.5.4 "Optimizing descriptor fetch"](#) for more information on the DMA\_PROCEED flag. If a DMA transfer is in progress for an endpoint when its corresponding bit is cleared, the transfer is completed before the DMA is disabled. When an error condition is detected during a DMA transfer, the corresponding bit is cleared by hardware. USBEPDMADis is a write-only register.

**Table 296. USB EP DMA Disable register (USBEPDMADis - address 0x2008 C28C) bit description**

Bit	Symbol	Value	Description	Reset value
0	EP0_DMA_DISABLE	0	Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_DISABLE bit value must be 0).	0
1	EP1_DMA_DISABLE	0	Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_DISABLE bit value must be 0).	0
31:2	EPxx_DMA_DISABLE		Endpoint xx (2 ≤ xx ≤ 31) DMA disable control bit.	0
		0	No effect.	
		1	Disable the DMA operation for endpoint EPxx.	

**13.10.8.8 USB DMA Interrupt Status register (USBDMAIntSt - 0x2008 C290)**

Each bit of this register reflects whether any of the 32 bits in the corresponding interrupt status register are set. USBDMAIntSt is a read-only register.

**Table 297. USB DMA Interrupt Status register (USBDMAIntSt - address 0x2008 C290) bit description**

Bit	Symbol	Value	Description	Reset value
0	EOT		End of Transfer Interrupt bit.	0
		0	All bits in the USBEoTIntSt register are 0.	
		1	At least one bit in the USBEoTIntSt is set.	
1	NDDR		New DD Request Interrupt bit.	0
		0	All bits in the USBNDDRIntSt register are 0.	
		1	At least one bit in the USBNDDRIntSt is set.	
2	ERR		System Error Interrupt bit.	0
		0	All bits in the USBSysErrIntSt register are 0.	
		1	At least one bit in the USBSysErrIntSt is set.	
31:3	-		Reserved. The value read from a reserved bit is not defined.	NA

**13.10.8.9 USB DMA Interrupt Enable register (USBDMAIntEn - 0x2008 C294)**

Writing a one to a bit in this register enables the corresponding bit in USBDMAIntSt to generate an interrupt on the USB\_INT\_REQ\_DMA interrupt line when set. USBDMAIntEn is a read/write register.

**Table 298. USB DMA Interrupt Enable register (USBDMAIntEn - address 0x2008 C294) bit description**

Bit	Symbol	Value	Description	Reset value
0	EOT		End of Transfer Interrupt enable bit.	0
		0	The End of Transfer Interrupt is disabled.	
		1	The End of Transfer Interrupt is enabled.	
1	NDDR		New DD Request Interrupt enable bit.	0
		0	The New DD Request Interrupt is disabled.	
		1	The New DD Request Interrupt is enabled.	
2	ERR		System Error Interrupt enable bit.	0
		0	The System Error Interrupt is disabled.	
		1	The System Error Interrupt is enabled.	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

**13.10.8.10 USB End of Transfer Interrupt Status register (USBEoTIntSt - 0x2008 C2A0)**

When the DMA transfer completes for the current DMA descriptor, either normally (descriptor is retired) or because of an error, the bit corresponding to the endpoint is set in this register. The cause of the interrupt is recorded in the DD\_status field of the descriptor. USBEoTIntSt is a read-only register.

**Table 299. USB End of Transfer Interrupt Status register (USBEoTIntSt - address 0x2008 C2A0) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Endpoint xx (2 ≤ xx ≤ 31) End of Transfer Interrupt request.	0
		0	There is no End of Transfer interrupt request for endpoint xx.	
		1	There is an End of Transfer Interrupt request for endpoint xx.	

**13.10.8.11 USB End of Transfer Interrupt Clear register (USBEoTIntClr - 0x2008 C2A4)**

Writing one to a bit in this register clears the corresponding bit in the USBEoTIntSt register. Writing zero has no effect. USBEoTIntClr is a write-only register.

**Table 300. USB End of Transfer Interrupt Clear register (USBEoTIntClr - address 0x2008 C2A4) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Clear endpoint xx ( $2 \leq xx \leq 31$ ) End of Transfer Interrupt request.	0
		0	No effect.	
		1	Clear the EPxx End of Transfer Interrupt request in the USBEoTIntSt register.	

**13.10.8.12 USB End of Transfer Interrupt Set register (USBEoTIntSet - 0x2008 C2A8)**

Writing one to a bit in this register sets the corresponding bit in the USBEoTIntSt register. Writing zero has no effect. USBEoTIntSet is a write-only register.

**Table 301. USB End of Transfer Interrupt Set register (USBEoTIntSet - address 0x2008 C2A8) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Set endpoint xx ( $2 \leq xx \leq 31$ ) End of Transfer Interrupt request.	0
		0	No effect.	
		1	Set the EPxx End of Transfer Interrupt request in the USBEoTIntSt register.	

**13.10.8.13 USB New DD Request Interrupt Status register (USBNDDRIntSt - 0x2008 C2AC)**

A bit in this register is set when a transfer is requested from the USB device and no valid DD is detected for the corresponding endpoint. USBNDDRIntSt is a read-only register.

**Table 302. USB New DD Request Interrupt Status register (USBNDDRIntSt - address 0x2008 C2AC) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Endpoint xx ( $2 \leq xx \leq 31$ ) new DD interrupt request.	0
		0	There is no new DD interrupt request for endpoint xx.	
		1	There is a new DD interrupt request for endpoint xx.	

**13.10.8.14 USB New DD Request Interrupt Clear register (USBNDDRIntClr - 0x2008 C2B0)**

Writing one to a bit in this register clears the corresponding bit in the USBNDDRIntSt register. Writing zero has no effect. USBNDDRIntClr is a write-only register.

**Table 303. USB New DD Request Interrupt Clear register (USBNDDRIntClr - address 0x2008 C2B0) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Clear endpoint xx ( $2 \leq xx \leq 31$ ) new DD interrupt request.	0
		0	No effect.	
		1	Clear the EPxx new DD interrupt request in the USBNDDRIntSt register.	

**13.10.8.15 USB New DD Request Interrupt Set register (USBNDDRIntSet - 0x2008 C2B4)**

Writing one to a bit in this register sets the corresponding bit in the USBNDDRIntSt register. Writing zero has no effect. USBNDDRIntSet is a write-only register.



**Table 304. USB New DD Request Interrupt Set register (USBNDDRIntSet - address 0x2008 C2B4) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Set endpoint xx ( $2 \leq xx \leq 31$ ) new DD interrupt request.	0
		0	No effect.	
		1	Set the EPxx new DD interrupt request in the USBNDDRIntSt register.	

**13.10.8.16 USB System Error Interrupt Status register (USBSysErrIntSt - 0x2008 C2B8)**

If a system error (AHB bus error) occurs when transferring the data or when fetching or updating the DD the corresponding bit is set in this register. USBSysErrIntSt is a read-only register.

**Table 305. USB System Error Interrupt Status register (USBSysErrIntSt - address 0x2008 C2B8) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Endpoint xx ( $2 \leq xx \leq 31$ ) System Error Interrupt request.	0
		0	There is no System Error Interrupt request for endpoint xx.	
		1	There is a System Error Interrupt request for endpoint xx.	

**13.10.8.17 USB System Error Interrupt Clear register (USBSysErrIntClr - 0x2008 C2BC)**

Writing one to a bit in this register clears the corresponding bit in the USBSysErrIntSt register. Writing zero has no effect. USBSysErrIntClr is a write-only register.

**Table 306. USB System Error Interrupt Clear register (USBSysErrIntClr - address 0x2008 C2BC) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Clear endpoint xx ( $2 \leq xx \leq 31$ ) System Error Interrupt request.	0
		0	No effect.	
		1	Clear the EPxx System Error Interrupt request in the USBSysErrIntSt register.	

**13.10.8.18 USB System Error Interrupt Set register (USBSysErrIntSet - 0x2008 C2C0)**

Writing one to a bit in this register sets the corresponding bit in the USBSysErrIntSt register. Writing zero has no effect. USBSysErrIntSet is a write-only register.

**Table 307. USB System Error Interrupt Set register (USBSysErrIntSet - address 0x2008 C2C0) bit description**

Bit	Symbol	Value	Description	Reset value
31:0	EPxx		Set endpoint xx ( $2 \leq xx \leq 31$ ) System Error Interrupt request.	0
		0	No effect.	
		1	Set the EPxx System Error Interrupt request in the USBSysErrIntSt register.	

## 13.11 Interrupt handling

This section describes how an interrupt event on any of the endpoints is routed to the Nested Vectored Interrupt Controller (NVIC). For a diagram showing interrupt event handling, see [Figure 41](#).

All non-isochronous OUT endpoints (control, bulk, and interrupt endpoints) generate an interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet has been successfully transmitted or when a NAK signal is sent and interrupts on NAK are enabled by the SIE Set Mode command, see [Section 13.12.3](#). For isochronous endpoints, a frame interrupt is generated every 1 ms.

The interrupt handling is different for Slave and DMA mode.

### Slave mode

If an interrupt event occurs on an endpoint and the endpoint interrupt is enabled in the USBEPIntEn register, the corresponding status bit in the USBEPIntSt is set. For non-isochronous endpoints, all endpoint interrupt events are divided into two types by the corresponding USBEPIntPri[n] registers: fast endpoint interrupt events and slow endpoint interrupt events. All fast endpoint interrupt events are ORed and routed to bit EP\_FAST in the USBDevIntSt register. All slow endpoint interrupt events are ORed and routed to the EP\_SLOW bit in USBDevIntSt.

For isochronous endpoints, the FRAME bit in USBDevIntSt is set every 1 ms.

The USBDevIntSt register holds the status of all endpoint interrupt events as well as the status of various other interrupts (see [Section 13.10.3.2](#)). By default, all interrupts (if enabled in USBDevIntEn) are routed to the USB\_INT\_REQ\_LP bit in the USBIntSt register to request low priority interrupt handling. However, the USBDevIntPri register can route either the FRAME or the EP\_FAST bit to the USB\_INT\_REQ\_HP bit in the USBIntSt register.

Only one of the EP\_FAST and FRAME interrupt events can be routed to the USB\_INT\_REQ\_HP bit. If routing both bits to USB\_INT\_REQ\_HP is attempted, both interrupt events are routed to USB\_INT\_REQ\_LP.

Slow endpoint interrupt events are always routed directly to the USB\_INT\_REQ\_LP bit for low priority interrupt handling by software.

The final interrupt signal to the NVIC is gated by the EN\_USB\_INTS bit in the USBIntSt register. The USB interrupts are routed to the NVIC only if EN\_USB\_INTS is set.

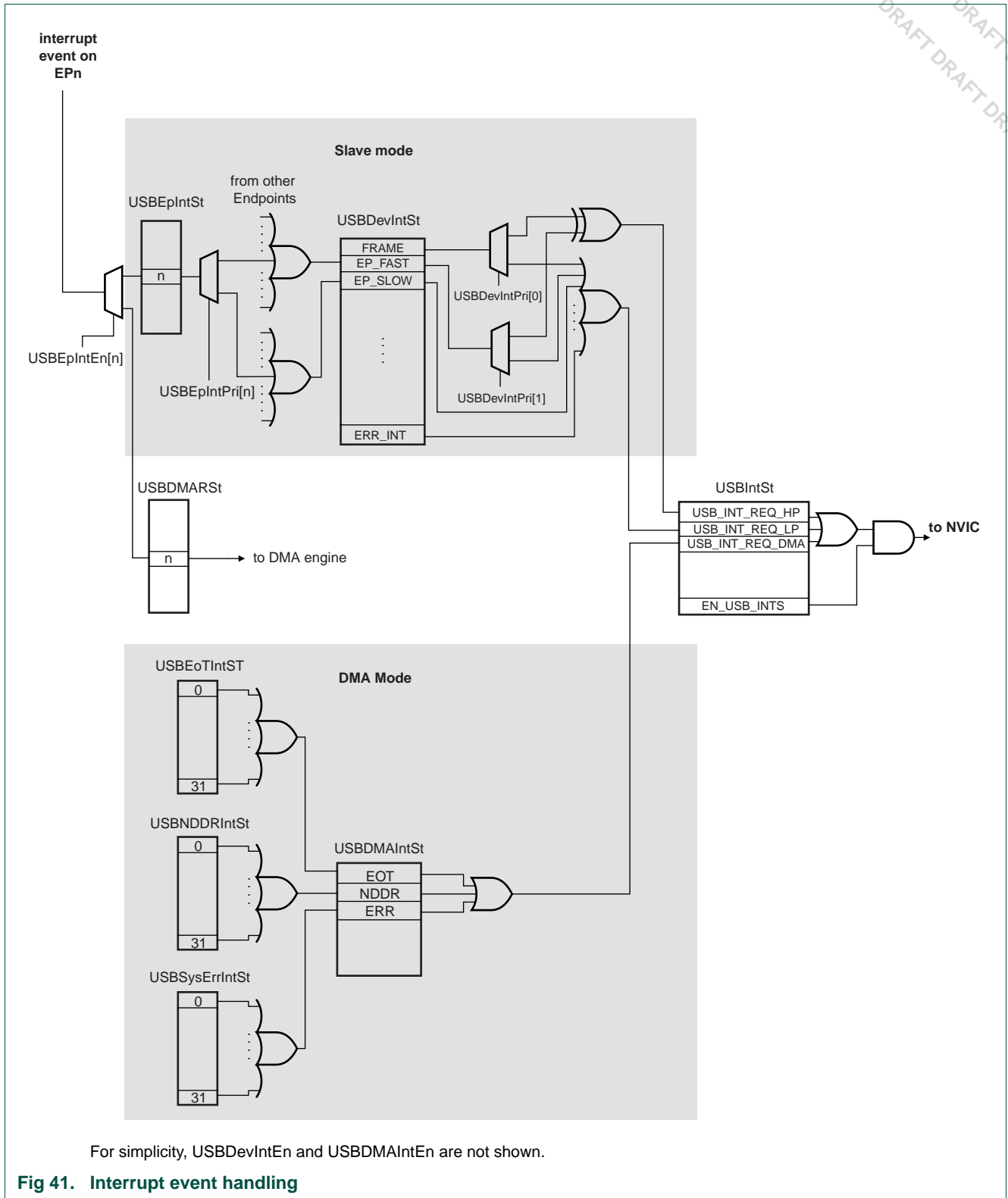
### DMA mode

If an interrupt event occurs on a non-control endpoint and the endpoint interrupt is not enabled in the USBEPIntEn register, the corresponding status bit in the USBDMARSt is set by hardware. This serves as a flag for the DMA engine to transfer data if DMA transfer is enabled for the corresponding endpoint in the USBEPDMASt register.

Three types of interrupts can occur for each endpoint for data transfers in DMA mode: End of transfer interrupt, new DD request interrupt, and system error interrupt. These interrupt events set a bit for each endpoint in the respective registers USBEoTIntSt, USBNDDRIntSt, and USBSysErrIntSt. The End of transfer interrupts from all endpoints

are then ORed and routed to the EOT bit in USBDMAIntSt. Likewise, all New DD request interrupts and system error interrupt events are routed to the NDDR and ERR bits respectively in the USBDMAStInt register.

The EOT, NDDR, and ERR bits (if enabled in USBDMAIntEn) are ORed to set the USB\_INT\_REQ\_DMA bit in the USBIntSt register. If the EN\_USB\_INTS bit is set in USBIntSt, the interrupt is routed to the NVIC.



## 13.12 Serial interface engine command description

The functions and registers of the Serial Interface Engine (SIE) are accessed using commands, which consist of a command code followed by optional data bytes (read or write action). The USBCmdCode ([Table 287](#)) and USBCmdData ([Table 288](#)) registers are used for these accesses.

A complete access consists of two phases:

1. **Command phase:** the USBCmdCode register is written with the CMD\_PHASE field set to the value 0x05 (Command), and the CMD\_CODE field set to the desired command code. On completion of the command, the CCEMPTY bit of USBDevIntSt is set.
2. **Data phase (optional):** for writes, the USBCmdCode register is written with the CMD\_PHASE field set to the value 0x01 (Write), and the CMD\_WDATA field set with the desired write data. On completion of the write, the CCEMPTY bit of USBDevIntSt is set. For reads, USBCmdCode register is written with the CMD\_PHASE field set to the value 0x02 (Read), and the CMD\_CODE field set with command code the read corresponds to. On completion of the read, the CDFULL bit of USBDevIntSt will be set, indicating the data is available for reading in the USBCmdData register. In the case of multi-byte registers, the least significant byte is accessed first.

An overview of the available commands is given in [Table 308](#).

Here is an example of the Read Current Frame Number command (reading 2 bytes):

```

USBDevIntClr = 0x30;           // Clear both CCEMPTY & CDFULL
USBCmdCode = 0x00F50500;      // CMD_CODE=0xF5, CMD_PHASE=0x05(Command)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY interrupt bit.
USBCmdCode = 0x00F50200;      // CMD_CODE=0xF5, CMD_PHASE=0x02(Read)
while (!(USBDevIntSt & 0x20)); // Wait for CDFULL.
USBDevIntClr = 0x20;           // Clear CDFULL.
CurFrameNum = USBCmdData;     // Read Frame number LSB byte.
USBCmdCode = 0x00F50200;      // CMD_CODE=0xF5, CMD_PHASE=0x02(Read)
while (!(USBDevIntSt & 0x20)); // Wait for CDFULL.
Temp = USBCmdData;            // Read Frame number MSB byte
USBDevIntClr = 0x20;           // Clear CDFULL interrupt bit.
CurFrameNum = CurFrameNum | (Temp << 8);

```

Here is an example of the Set Address command (writing 1 byte):

```

USBDevIntClr = 0x10;           // Clear CCEMPTY.
USBCmdCode = 0x00D00500;      // CMD_CODE=0xD0, CMD_PHASE=0x05(Command)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY.
USBCmdCode = 0x008A0100;      // CMD_WDATA=0x8A(DEV_EN=1, DEV_ADDR=0xA),
                               // CMD_PHASE=0x01(Write)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY.

```

Table 308. SIE command code table

Command name	Recipient	Code (Hex)	Data phase
<b>Device commands</b>			
Set Address	Device	D0	Write 1 byte
Configure Device	Device	D8	Write 1 byte
Set Mode	Device	F3	Write 1 byte
Read Current Frame Number	Device	F5	Read 1 or 2 bytes
Read Test Register	Device	FD	Read 2 bytes
Set Device Status	Device	FE	Write 1 byte
Get Device Status	Device	FE	Read 1 byte
Get Error Code	Device	FF	Read 1 byte
Read Error Status	Device	FB	Read 1 byte
<b>Endpoint Commands</b>			
Select Endpoint	Endpoint 0	00	Read 1 byte (optional)
	Endpoint 1	01	Read 1 byte (optional)
	Endpoint xx	xx	Read 1 byte (optional)
Select Endpoint/Clear Interrupt	Endpoint 0	40	Read 1 byte
	Endpoint 1	41	Read 1 byte
	Endpoint xx	xx + 40	Read 1 byte
Set Endpoint Status	Endpoint 0	40	Write 1 byte
	Endpoint 1	41	Write 1 byte
	Endpoint xx	xx + 40	Write 1 byte
Clear Buffer	Selected Endpoint	F2	Read 1 byte (optional)
Validate Buffer	Selected Endpoint	FA	None

### 13.12.1 Set Address (Command: 0xD0, Data: write 1 byte)

The Set Address command is used to set the USB assigned address and enable the (embedded) function. The address set in the device will take effect after the status stage of the control transaction. After a bus reset, DEV\_ADDR is set to 0x00, and DEV\_EN is set to 1. The device will respond to packets for function address 0x00, endpoint 0 (default endpoint).

Table 309. Set Address command bit description

Bit	Symbol	Description	Reset value
6:0	DEV_ADDR	Device address set by the software. After a bus reset this field is set to 0x00.	0
7	DEV_EN	Device Enable. After a bus reset this bit is set to 1. 0: Device will not respond to any packets. 1: Device will respond to packets for function address DEV_ADDR.	0

### 13.12.2 Configure Device (Command: 0xD8, Data: write 1 byte)

A value of 1 written to the register indicates that the device is configured and all the enabled non-control endpoints will respond. Control endpoints are always enabled and respond even if the device is not configured, in the default state.

Table 310. Configure Device command bit description

Bit	Symbol	Description	Reset value
0	CONF_DEVICE	Device is configured. All enabled non-control endpoints will respond. This bit is cleared by hardware when a bus reset occurs. When set, the UP_LED signal is driven LOW if the device is not in the suspended state (SUS=0).	0
7:1	-	Reserved. Read value is undefined, only zero should be written.	NA

### 13.12.3 Set Mode (Command: 0xF3, Data: write 1 byte)

Table 311. Set Mode command bit description

Bit	Symbol	Value	Description	Reset value
0	AP_CLK		Always PLL Clock.	0
		0	USB_NEED_CLK is functional; the 48 MHz clock can be stopped when the device enters suspend state.	
		1	USB_NEED_CLK is fixed to 1; the 48 MHz clock cannot be stopped when the device enters suspend state.	
1	INAK_CI		Interrupt on NAK for Control IN endpoint.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed IN transactions generate interrupts.	
2	INAK_CO		Interrupt on NAK for Control OUT endpoint.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed OUT transactions generate interrupts.	
3	INAK_I1		Interrupt on NAK for Interrupt IN endpoint.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed IN transactions generate interrupts.	
4	INAK_IO <sup>[1]</sup>		Interrupt on NAK for Interrupt OUT endpoints.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed OUT transactions generate interrupts.	
5	INAK_B1		Interrupt on NAK for Bulk IN endpoints.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed IN transactions generate interrupts.	
6	INAK_BO <sup>[2]</sup>		Interrupt on NAK for Bulk OUT endpoints.	0
		0	Only successful transactions generate an interrupt.	
		1	Both successful and NAKed OUT transactions generate interrupts.	
7	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] This bit should be reset to 0 if the DMA is enabled for any of the Interrupt OUT endpoints.

[2] This bit should be reset to 0 if the DMA is enabled for any of the Bulk OUT endpoints.

### 13.12.4 Read Current Frame Number (Command: 0xF5, Data: read 1 or 2 bytes)

Returns the frame number of the last successfully received SOF. The frame number is eleven bits wide. The frame number returns least significant byte first. In case the user is only interested in the lower 8 bits of the frame number, only the first byte needs to be read.

- In case no SOF was received by the device at the beginning of a frame, the frame number returned is that of the last successfully received SOF.
- In case the SOF frame number contained a CRC error, the frame number returned will be the corrupted frame number as received by the device.

### 13.12.5 Read Test Register (Command: 0xFD, Data: read 2 bytes)

The test register is 16 bits wide. It returns the value of 0xA50F if the USB clocks (usbclk and AHB slave clock) are running.

### 13.12.6 Set Device Status (Command: 0xFE, Data: write 1 byte)

The Set Device Status command sets bits in the Device Status Register.

Table 312. Set Device Status command bit description

Bit	Symbol	Value	Description	Reset value
0	CON		The Connect bit indicates the current connect status of the device. It controls the CONNECT output pin, used for SoftConnect. Reading the connect bit returns the current connect status. This bit is cleared by hardware when the V <sub>BUS</sub> status input is LOW for more than 3 ms. The 3 ms delay filters out temporary dips in the V <sub>BUS</sub> voltage.	0
		0	Writing a 0 will make the CONNECT pin go HIGH.	
		1	Writing a 1 will make the CONNECT pin go LOW.	
1	CON_CH		Connect Change.	0
		0	This bit is cleared when read.	
2	SUS		Suspend: The Suspend bit represents the current suspend state. When the device is suspended (SUS = 1) and the CPU writes a 0 into it, the device will generate a remote wake-up. This will only happen when the device is connected (CON = 1). When the device is not connected or not suspended, writing a 0 has no effect. Writing a 1 to this bit has no effect.	0
		0	This bit is reset to 0 on any activity.	
3	SUS_CH		Suspend (SUS) bit change indicator. The SUS bit can toggle because: <ul style="list-style-type: none"> <li>• The device goes into the suspended state.</li> <li>• The device is disconnected.</li> <li>• The device receives resume signalling on its upstream port.</li> </ul> This bit is cleared when read.	0
		0	SUS bit not changed.	
		1	SUS bit changed. At the same time a DEV_STAT interrupt is generated.	



Table 312. Set Device Status command bit description

Bit	Symbol	Value	Description	Reset value
4	RST		Bus Reset bit. On a bus reset, the device will automatically go to the default state. In the default state: <ul style="list-style-type: none"> <li>• Device is unconfigured.</li> <li>• Will respond to address 0.</li> <li>• Control endpoint will be in the Stalled state.</li> <li>• All endpoints are unrealized except control endpoints EP0 and EP1.</li> <li>• Data toggling is reset for all endpoints.</li> <li>• All buffers are cleared.</li> <li>• There is no change to the endpoint interrupt status.</li> <li>• DEV_STAT interrupt is generated.</li> </ul> Note: Bus resets are ignored when the device is not connected (CON=0).	0
		0	This bit is cleared when read.	
		1	This bit is set when the device receives a bus reset. A DEV_STAT interrupt is generated.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 13.12.7 Get Device Status (Command: 0xFE, Data: read 1 byte)

The Get Device Status command returns the Device Status Register. Reading the device status returns 1 byte of data. The bit field definition is same as the Set Device Status Register as shown in [Table 312](#).

**Remark:** To ensure correct operation, the DEV\_STAT bit of USBDevIntSt must be cleared before executing the Get Device Status command.

### 13.12.8 Get Error Code (Command: 0xFF, Data: read 1 byte)

Different error conditions can arise inside the SIE. The Get Error Code command returns the last error code that occurred. The 4 least significant bits form the error code.

Table 313. Get Error Code command bit description

Bit	Symbol	Value	Description	Reset value
3:0	EC		Error Code.	0
		0000	No Error.	
		0001	PID Encoding Error.	
		0010	Unknown PID.	
		0011	Unexpected Packet - any packet sequence violation from the specification.	
		0100	Error in Token CRC.	
		0101	Error in Data CRC.	
		0110	Time Out Error.	
		0111	Babble.	
		1000	Error in End of Packet.	
		1001	Sent/Received NAK.	
		1010	Sent Stall.	
		1011	Buffer Overrun Error.	
4	EA	-	The Error Active bit will be reset once this register is read.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 13.12.9 Read Error Status (Command: 0xFB, Data: read 1 byte)

This command reads the 8-bit Error register from the USB device. This register records which error events have recently occurred in the SIE. If any of these bits are set, the ERR\_INT bit of USBDevIntSt is set. The error bits are cleared after reading this register.

Table 314. Read Error Status command bit description

Bit	Symbol	Description	Reset value
0	PID_ERR	PID encoding error or Unknown PID or Token CRC.	0
1	UEPKT	Unexpected Packet - any packet sequence violation from the specification.	0
2	DCRC	Data CRC error.	0
3	TIMEOUT	Time out error.	0
4	EOP	End of packet error.	0
5	B_OVRN	Buffer Overrun.	0
6	BTSTF	Bit stuff error.	0
7	TGL_ERR	Wrong toggle bit in data PID, ignored data.	0

### 13.12.10 Select Endpoint (Command: 0x00 - 0x1F, Data: read 1 byte (optional))

The Select Endpoint command initializes an internal pointer to the start of the selected buffer in EP\_RAM. Optionally, this command can be followed by a data read, which returns some additional information on the packet(s) in the endpoint buffer(s). The command code of the Select Endpoint command is equal to the physical endpoint number. In the case of a single buffered endpoint the B\_2\_FULL bit is not valid.

Table 315. Select Endpoint command bit description

Bit	Symbol	Value	Description	Reset value
0	FE		Full/Empty. This bit indicates the full or empty status of the endpoint buffer(s). For IN endpoints, the FE bit gives the ANDed result of the B_1_FULL and B_2_FULL bits. For OUT endpoints, the FE bit gives ORed result of the B_1_FULL and B_2_FULL bits. For single buffered endpoints, this bit simply reflects the status of B_1_FULL.	0
		0	For an IN endpoint, at least one write endpoint buffer is empty.	
		1	For an OUT endpoint, at least one endpoint read buffer is full.	
1	ST		Stalled endpoint indicator.	0
		0	The selected endpoint is not stalled.	
		1	The selected endpoint is stalled.	
2	STP		SETUP bit: the value of this bit is updated after each successfully received packet (i.e. an ACKed package on that particular physical endpoint).	0
		0	The STP bit is cleared by doing a Select Endpoint/Clear Interrupt on this endpoint.	
		1	The last received packet for the selected endpoint was a SETUP packet.	
3	PO		Packet over-written bit.	0
		0	The PO bit is cleared by the 'Select Endpoint/Clear Interrupt' command.	
		1	The previously received packet was over-written by a SETUP packet.	
4	EPN		EP NAKed bit indicates sending of a NAK. If the host sends an OUT packet to a filled OUT buffer, the device returns NAK. If the host sends an IN token packet to an empty IN buffer, the device returns NAK.	0
		0	The EPN bit is reset after the device has sent an ACK after an OUT packet or when the device has seen an ACK after sending an IN packet.	
		1	The EPN bit is set when a NAK is sent and the interrupt on NAK feature is enabled.	
5	B_1_FULL		The buffer 1 status.	0
		0	Buffer 1 is empty.	
		1	Buffer 1 is full.	
6	B_2_FULL		The buffer 2 status.	0
		0	Buffer 2 is empty.	
		1	Buffer 2 is full.	
7	-		Reserved. Read value is undefined, only zero should be written.	NA

### 13.12.11 Select Endpoint/Clear Interrupt (Command: 0x40 - 0x5F, Data: read 1 byte)

Commands 0x40 to 0x5F are identical to their Select Endpoint equivalents, with the following differences:

- They clear the bit corresponding to the endpoint in the USBEPIntSt register.
- In case of a control OUT endpoint, they clear the STP and PO bits in the corresponding Select Endpoint Register.
- Reading one byte is obligatory.

**Remark:** This command may be invoked by using the USBCmdCode and USBCmdData registers, or by setting the corresponding bit in USBEplntClr. For ease of use, using the USBEplntClr register is recommended.

### 13.12.12 Set Endpoint Status (Command: 0x40 - 0x55, Data: write 1 byte (optional))

The Set Endpoint Status command sets status bits 7:5 and 0 of the endpoint. The Command Code of Set Endpoint Status is equal to the sum of 0x40 and the physical endpoint number in hex. Not all bits can be set for all types of endpoints.

**Table 316. Set Endpoint Status command bit description**

Bit	Symbol	Value	Description	Reset value
0	ST		Stalled endpoint bit. A Stalled control endpoint is automatically unstalled when it receives a SETUP token, regardless of the content of the packet. If the endpoint should stay in its stalled state, the CPU can stall it again by setting this bit. When a stalled endpoint is unstalled - either by the Set Endpoint Status command or by receiving a SETUP token - it is also re-initialized. This flushes the buffer: in case of an OUT buffer it waits for a DATA 0 PID; in case of an IN buffer it writes a DATA 0 PID. There is no change of the interrupt status of the endpoint. When already unstalled, writing a zero to this bit initializes the endpoint. When an endpoint is stalled by the Set Endpoint Status command, it is also re-initialized.	0
		0	The endpoint is unstalled.	
		1	The endpoint is stalled.	
4:1	-		Reserved. Read value is undefined, only zero should be written.	NA
5	DA		Disabled endpoint bit.	0
		0	The endpoint is enabled.	
		1	The endpoint is disabled.	
6	RF_MO		Rate Feedback Mode.	0
		0	Interrupt endpoint is in the Toggle mode.	
		1	Interrupt endpoint is in the Rate Feedback mode. This means that transfer takes place without data toggle bit.	
7	CND_ST		Conditional Stall bit.	0
		0	Unstalls both control endpoints.	
		1	Stall both control endpoints, unless the STP bit is set in the Select Endpoint register. It is defined only for control OUT endpoints.	

### 13.12.13 Clear Buffer (Command: 0xF2, Data: read 1 byte (optional))

When an OUT packet sent by the host has been received successfully, an internal hardware FIFO status Buffer\_Full flag is set. All subsequent packets will be refused by returning a NAK. When the device software has read the data, it should free the buffer by issuing the Clear Buffer command. This clears the internal Buffer\_Full flag. When the buffer is cleared, new packets will be accepted.

When bit 0 of the optional data byte is 1, the previously received packet was over-written by a SETUP packet. The Packet over-written bit is used only in control transfers. According to the USB specification, a SETUP packet should be accepted irrespective of the buffer status. The software should always check the status of the PO bit after reading

the SETUP data. If it is set then it should discard the previously read data, clear the PO bit by issuing a Select Endpoint/Clear Interrupt command, read the new SETUP data and again check the status of the PO bit.

See [Section 13.14 “Slave mode operation”](#) for a description of when this command is used.

**Table 317. Clear Buffer command bit description**

Bit	Symbol	Value	Description	Reset value
0	PO		Packet over-written bit. This bit is only applicable to the control endpoint EP0.	0
		0	The previously received packet is intact.	
		1	The previously received packet was over-written by a later SETUP packet.	
7:1	-		Reserved. Read value is undefined, only zero should be written.	NA

**13.12.14 Validate Buffer (Command: 0xFA, Data: none)**

When the CPU has written data into an IN buffer, software should issue a Validate Buffer command. This tells hardware that the buffer is ready for sending on the USB bus. Hardware will send the contents of the buffer when the next IN token packet is received.

Internally, there is a hardware FIFO status flag called Buffer\_Full. This flag is set by the Validate Buffer command and cleared when the data has been sent on the USB bus and the buffer is empty.

A control IN buffer cannot be validated when its corresponding OUT buffer has the Packet Over-written (PO) bit (see the Clear Buffer Register) set or contains a pending SETUP packet. For the control endpoint the validated buffer will be invalidated when a SETUP packet is received.

See [Section 13.14 “Slave mode operation”](#) for a description of when this command is used.

## 13.13 USB device controller initialization

The LPC178x/177x USB device controller initialization includes the following steps:

1. Enable the device controller by setting the PCUSB bit of PCONP.
2. Configure and enable the PLL and Clock Dividers to provide 48 MHz for usbcclk and the desired frequency for cclk. For the procedure for determining the PLL setting and configuration, see [Section 4.5.10 “Procedure for determining PLL settings”](#).
3. Enable the device controller clocks by setting DEV\_CLK\_EN and AHB\_CLK\_EN bits in the USBClkCtrl register. Poll the respective clock bits in the USBClkSt register until they are set.
4. Select the desired USB port pins using the USBPortSel register. The PORTSEL\_CLK\_EN bit must be set in USBClkCtrl before accessing USBPortSel and should be cleared after accessing USBPortSel.
5. Enable the USB pin functions by writing to the corresponding IOCON registers.
6. Disable the pull-ups and pull-downs on the V<sub>BUS</sub> pin using the corresponding IOCON register by putting the pin in the “plain-input” mode. See [Section 8.4.1 “I/O configuration register contents \(IOCON\)”](#).
7. Set USBEPIn and USBMaxPSize registers for EP0 and EP1, and wait until the EP\_RLZED bit in USBDevIntSt is set so that EP0 and EP1 are realized.
8. Enable endpoint interrupts (Slave mode):
  - Clear all endpoint interrupts using USBEPIntClr.
  - Clear any device interrupts using USBDevIntClr.
  - Enable Slave mode for the desired endpoints by setting the corresponding bits in USBEPIntEn.
  - Set the priority of each enabled interrupt using USBEPIntPri.
  - Configure the desired interrupt mode using the SIE Set Mode command.
  - Enable device interrupts using USBDevIntEn (normally DEV\_STAT, EP\_SLOW, and possibly EP\_FAST).
9. Configure the DMA (DMA mode):
  - Disable DMA operation for all endpoints using USBEPDMADis.
  - Clear any pending DMA requests using USBDMARClr.
  - Clear all DMA interrupts using USBEoTIntClr, USBNDDRIntClr, and USBSysErrIntClr.
  - Prepare the UDCA in system memory.
  - Write the desired address for the UDCA to USBUDCAH.
  - Enable the desired endpoints for DMA operation using USBEPDMAEn.
  - Set EOT, DDR, and ERR bits in USBDMAIntEn.
10. Install USB interrupt handler in the NVIC by writing its address to the appropriate vector table location and enabling the USB interrupt in the NVIC.
11. Set default USB address to 0x0 and DEV\_EN to 1 using the SIE Set Address command. A bus reset will also cause this to happen.
12. Set CON bit to 1 to make CONNECT active using the SIE Set Device Status command.

The configuration of the endpoints varies depending on the software application. By default, all the endpoints are disabled except control endpoints EP0 and EP1. Additional endpoints are enabled and configured by software after a SET\_CONFIGURATION or SET\_INTERFACE device request is received from the host.

## 13.14 Slave mode operation

In Slave mode, the CPU transfers data between RAM and the endpoint buffer using the Register Interface.

### 13.14.1 Interrupt generation

In slave mode, data packet transfer between RAM and an endpoint buffer can be initiated in response to an endpoint interrupt. Endpoint interrupts are enabled using the USBEPIntEn register, and are observable in the USBEPIntSt register.

All non-isochronous OUT endpoints generate an endpoint interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet is successfully transmitted, or when a NAK handshake is sent on the bus and the interrupt on NAK feature is enabled.

For Isochronous endpoints, transfer of data is done when the FRAME interrupt (in USBDevIntSt) occurs.

### 13.14.2 Data transfer for OUT endpoints

When the software wants to read the data from an endpoint buffer it should set the RD\_EN bit and program LOG\_ENDPOINT with the desired endpoint number in the USBCtrl register. The control logic will fetch the packet length to the USBRxPLen register, and set the PKT\_RDY bit ([Table 283](#)).

Software can now start reading the data from the USBRxData register ([Table 282](#)). When the end of packet is reached, the RD\_EN bit is cleared, and the RxENDPKT bit is set in the USBDevSt register. Software now issues a Clear Buffer (refer to [Table 317](#)) command. The endpoint is now ready to accept the next packet. For OUT isochronous endpoints, the next packet will be received irrespective of whether the buffer has been cleared. Any data not read from the buffer before the end of the frame is lost. See [Section 13.16 “Double buffered endpoint operation”](#) for more details.

If the software clears RD\_EN before the entire packet is read, reading is terminated, and the data remains in the endpoint's buffer. When RD\_EN is set again for this endpoint, the data will be read from the beginning.

### 13.14.3 Data transfer for IN endpoints

When writing data to an endpoint buffer, WR\_EN ([Section 13.10.6.5 “USB Control register \(USBCtrl - 0x2008 C228\)”](#)) is set and software writes to the number of bytes it is going to send in the packet to the USBTxPLen register ([Section 13.10.6.4](#)). It can then write data continuously in the USBTxData register.

When the number of bytes programmed in USBTxPLen have been written to USBTxData, the WR\_EN bit is cleared, and the TxENDPKT bit is set in the USBDevIntSt register. Software issues a Validate Buffer ([Section 13.12.14 “Validate Buffer \(Command: 0xFA, Data: none\)”](#)) command. The endpoint is now ready to send the packet. For IN isochronous endpoints, the data in the buffer will be sent only if the buffer is validated before the next FRAME interrupt occurs; otherwise, an empty packet will be sent in the next frame. If the software clears WR\_EN before the entire packet is written, writing will start again from the beginning the next time WR\_EN is set for this endpoint.



Both RD\_EN and WR\_EN can be high at the same time for the same logical endpoint. Interleaved read and write operation is possible.

## 13.15 DMA operation

In DMA mode, the DMA transfers data between RAM and the endpoint buffer.

The following sections discuss DMA mode operation. Background information is given in sections [Section 13.15.2 “USB device communication area”](#) and [Section 13.15.3 “Triggering the DMA engine”](#). The fields of the DMA Descriptor are described in [Section 13.15.4 “The DMA descriptor”](#). The last three sections describe DMA operation: [Section 13.15.5 “Non-isochronous endpoint operation”](#), [Section 13.15.6 “Isochronous endpoint operation”](#), and [Section 13.15.7 “Auto Length Transfer Extraction \(ATLE\) mode operation”](#).

### 13.15.1 Transfer terminology

Within this section three types of transfers are mentioned:

1. USB transfers – transfer of data over the USB bus. The USB 2.0 specification refers to these simply as transfers. Within this section they are referred to as USB transfers to distinguish them from DMA transfers. A USB transfer is composed of transactions. Each transaction is composed of packets.
2. DMA transfers – the transfer of data between an endpoint buffer and system memory (RAM).
3. Packet transfers – in this section, a packet transfer refers to the transfer of a packet of data between an endpoint buffer and system memory (RAM). A DMA transfer is composed of one or more packet transfers.

### 13.15.2 USB device communication area

The CPU and DMA controller communicate through a common area of memory, called the USB Device Communication Area, or UDCA. The UDCA is a 32-word array of DMA Descriptor Pointers (DDPs), each of which corresponds to a physical endpoint. Each DDP points to the start address of a DMA Descriptor, if one is defined for the endpoint. DDPs for unrealized endpoints and endpoints disabled for DMA operation are ignored and can be set to a NULL (0x0) value.

The start address of the UDCA is stored in the USBUDCAH register. The UDCA can reside at any 128-byte boundary of RAM that is accessible to both the CPU and DMA controller.

[Figure 42](#) illustrates the UDCA and its relationship to the UDCA Head (USBUDCAH) register and DMA Descriptors.

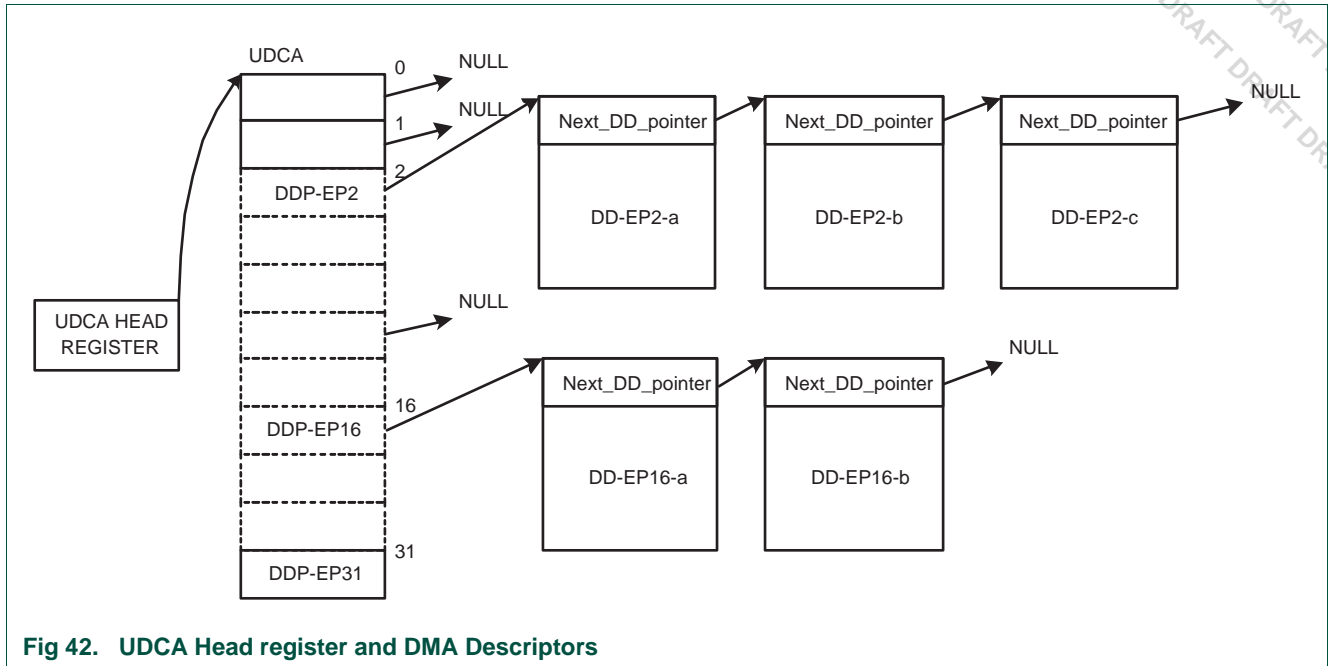


Fig 42. UDCA Head register and DMA Descriptors

### 13.15.3 Triggering the DMA engine

An endpoint raises a DMA request when Slave mode is disabled by setting the corresponding bit in the USBEpIntEn register to 0 (Section 13.10.4.2) and an endpoint interrupt occurs (see Section 13.10.8.1 “USB DMA Request Status register (USBDMARSt - 0x2008 C250)”).

A DMA transfer for an endpoint starts when the endpoint is enabled for DMA operation in USBEpDMASt, the corresponding bit in USBDMARSt is set, and a valid DD is found for the endpoint.

All endpoints share a single DMA channel to minimize hardware overhead. If more than one DMA request is active in USBDMARSt, the endpoint with the lowest physical endpoint number is processed first.

In DMA mode, the bits corresponding to Interrupt on NAK for Bulk OUT and Interrupt OUT endpoints (INAK\_BO and INAK\_IO) should be set to 0 using the SIE Set Mode command (Section 13.12.3).

### 13.15.4 The DMA descriptor

DMA transfers are described by a data structure called the DMA Descriptor (DD).

DDs are placed in RAM. These descriptors can be located anywhere in on-chip RAM at word-aligned addresses.

DDs for non-isochronous endpoints are four words long. DDs for isochronous endpoints are five words long.

The parameters associated with a DMA transfer are:

- The start address of the DMA buffer

- The length of the DMA buffer
- The start address of the next DMA descriptor
- Control information
- Count information (number of bytes transferred)
- Status information

Table 318 lists the DMA descriptor fields.

Table 318. DMA descriptor

Word position	Access (H/W)	Access (S/W)	Bit position	Description	
0	R	R/W	31:0	Next_DD_pointer	
1	R	R/W	1:0	DMA_mode (00 -Normal; 01 - ATLE)	
	R	R/W	2	Next_DD_valid (1 - valid; 0 - invalid)	
	-	-	3	Reserved. Read value is undefined, only zero should be written.	
	R	R/W	4	Isochronous_endpoint (1 - isochronous; 0 - non-isochronous)	
	R	R/W	15:5	Max_packet_size	
	R/W <sup>[1]</sup>	R/W	31:16	DMA_buffer_length This value is specified in bytes for non-isochronous endpoints and in number of packets for isochronous endpoints.	
	2	R/W	R/W	31:0	DMA_buffer_start_addr
3	R/W	R/I	0	DD_retired (To be initialized to 0)	
	W	R/I	4:1	DD_status (To be initialized to 0000): 0000 - NotServiced 0001 - BeingServiced 0010 - NormalCompletion 0011 - DataUnderrun (short packet) 1000 - DataOverrun 1001 - SystemError	
	W	R/I	5	Packet_valid (To be initialized to 0)	
	W	R/I	6	LS_byte_extracted (ATLE mode) (To be initialized to 0)	
	W	R/I	7	MS_byte_extracted (ATLE mode) (To be initialized to 0)	
	R	W	13:8	Message_length_position (ATLE mode)	
	-	-	15:14	Reserved. Read value is undefined, only zero should be written.	
	R/W	R/I	31:16	Present_DMA_count (To be initialized to 0)	
	4	R/W	R/W	31:0	Isochronous_packetsize_memory_address

[1] Write-only in ATLE mode

Legend: R - Read; W - Write; I - Initialize

### 13.15.4.1 Next\_DD\_pointer

Pointer to the memory location from where the next DMA descriptor will be fetched.

#### 13.15.4.2 DMA\_mode

Specifies the DMA mode of operation. Two modes have been defined: Normal and Automatic Transfer Length Extraction (ATLE) mode. In normal mode, software initializes the DMA\_buffer\_length for OUT endpoints. In ATLE mode, the DMA\_buffer\_length is extracted from the incoming data. See [Section 13.15.7 “Auto Length Transfer Extraction \(ATLE\) mode operation” on page 362](#) for more details.

#### 13.15.4.3 Next\_DD\_valid

This bit indicates whether the software has prepared the next DMA descriptor. If set, the DMA engine fetches the new descriptor when it is finished with the current one.

#### 13.15.4.4 Isochronous\_endpoint

When set, this bit indicates that the descriptor belongs to an isochronous endpoint. Hence 5 words have to be read when fetching it.

#### 13.15.4.5 Max\_packet\_size

The maximum packet size of the endpoint. This parameter is used while transferring the data for IN endpoints from the memory. It is used for OUT endpoints to detect the short packet. This is applicable to non-isochronous endpoints only. This field should be set to the same MPS value that is assigned for the endpoint using the USBMaxPSize register.

#### 13.15.4.6 DMA\_buffer\_length

This indicates the depth of the DMA buffer allocated for transferring the data. The DMA engine will stop using this descriptor when this limit is reached and will look for the next descriptor.

In Normal mode operation, software sets this value for both IN and OUT endpoints. In ATLE mode operation, software sets this value for IN endpoints only. For OUT endpoints, hardware sets this value using the extracted length of the data stream.

For isochronous endpoints, DMA\_buffer\_length is specified in number of packets, for non-isochronous endpoints in bytes.

#### 13.15.4.7 DMA\_buffer\_start\_addr

The address where the data is read from or written to. This field is updated each time the DMA engine finishes transferring a packet.

#### 13.15.4.8 DD\_retired

This bit is set by hardware when the DMA engine finishes the current descriptor. This happens when the end of the buffer is reached, a short packet is transferred (non-isochronous endpoints), or an error condition is detected.

#### 13.15.4.9 DD\_status

The status of the DMA transfer is encoded in this field. The following codes are defined:

- **NotServiced** - No packet has been transferred yet.
- **BeingServiced** - At least one packet is transferred.
- **NormalCompletion** - The DD is retired because the end of the buffer is reached and there were no errors. The DD\_retired bit is also set.

- **DataUnderrun** - Before reaching the end of the DMA buffer, the USB transfer is terminated because a short packet is received. The DD\_retired bit is also set.
- **DataOverrun** - The end of the DMA buffer is reached in the middle of a packet transfer. This is an error situation. The DD\_retired bit is set. The present DMA count field is equal to the value of DMA\_buffer\_length. The packet must be re-transmitted from the endpoint buffer in another DMA transfer. The corresponding EPxx\_DMA\_ENABLE bit in USBEpDMASt is cleared.
- **SystemError** - The DMA transfer being serviced is terminated because of an error on the AHB bus. The DD\_retired bit is not set in this case. The corresponding EPxx\_DMA\_ENABLE in USBEpDMASt is cleared. Since a system error can happen while updating the DD, the DD fields in RAM may be unreliable.

#### 13.15.4.10 Packet\_valid

This bit is used for isochronous endpoints. It indicates whether the last packet transferred to the memory is received with errors or not. This bit is set if the packet is valid, i.e., it was received without errors. See [Section 13.15.6 "Isochronous endpoint operation" on page 360](#) for isochronous endpoint operation.

This bit is unnecessary for non-isochronous endpoints because a DMA request is generated only for packets without errors, and thus Packet\_valid will always be set when the request is generated.

#### 13.15.4.11 LS\_byte\_extracted

Used in ATLE mode. When set, this bit indicates that the Least Significant Byte (LSB) of the transfer length has been extracted. The extracted size is reflected in the DMA\_buffer\_length field, bits 23:16.

#### 13.15.4.12 MS\_byte\_extracted

Used in ATLE mode. When set, this bit indicates that the Most Significant Byte (MSB) of the transfer size has been extracted. The size extracted is reflected in the DMA\_buffer\_length field, bits 31:24. Extraction stops when LS\_Byte\_extracted and MS\_byte\_extracted bits are set.

#### 13.15.4.13 Present\_DMA\_count

The number of bytes transferred by the DMA engine. The DMA engine updates this field after completing each packet transfer.

For isochronous endpoints, Present\_DMA\_count is the number of packets transferred; for non-isochronous endpoints, Present\_DMA\_count is the number of bytes.

#### 13.15.4.14 Message\_length\_position

Used in ATLE mode. This field gives the offset of the message length position embedded in the incoming data packets. This is applicable only for OUT endpoints. Offset 0 indicates that the message length starts from the first byte of the first packet.

#### 13.15.4.15 Isochronous\_packetsize\_memory\_address

The memory buffer address where the packet size information along with the frame number has to be transferred or fetched. See [Figure 43](#). This is applicable to isochronous endpoints only.

## 13.15.5 Non-isochronous endpoint operation

### 13.15.5.1 Setting up DMA transfers

Software prepares the DMA Descriptors (DDs) for those physical endpoints to be enabled for DMA transfer. These DDs are present in on-chip RAM. The start address of the first DD is programmed into the DMA Description pointer (DDP) location for the corresponding endpoint in the UDCA. Software then sets the EPxx\_DMA\_ENABLE bit for this endpoint in the USBEpDMAEn register ([Section 13.10.8.6](#)). The DMA\_mode bit field in the descriptor is set to '00' for normal mode operation. All other DD fields are initialized as specified in [Table 318](#).

DMA operation is not supported for physical endpoints 0 and 1 (default control endpoints).

### 13.15.5.2 Finding DMA Descriptor

When there is a trigger for a DMA transfer for an endpoint, the DMA engine will first determine whether a new descriptor has to be fetched or not. A new descriptor does not have to be fetched if the last packet transferred was for the same endpoint and the DD is not yet in the retired state. An internal flag called DMA\_PROCEED is used to identify this condition (see [Section 13.15.5.4 "Optimizing descriptor fetch" on page 359](#)).

If a new descriptor has to be read, the DMA engine will calculate the location of the DDP for this endpoint and will fetch the start address of the DD from this location. A DD start address at location zero is considered invalid. In this case the NDDR interrupt is raised. All other word-aligned addresses are considered valid.

When the DD is fetched, the DD status word (word 3) is read first and the status of the DD\_retired bit is checked. If not set, DDP points to a valid DD. If DD\_retired is set, the DMA engine will read the control word (word 1) of the DD.

If Next\_DD\_valid bit is set, the DMA engine will fetch the Next\_DD\_pointer field (word 0) of the DD and load it to the DDP. The new DDP is written to the UDCA area.

The full DD (4 words) will then be fetched from the address in the DDP. The DD will give the details of the DMA transfer to be done. The DMA engine will load its hardware resources with the information fetched from the DD (start address, DMA count etc.).

If Next\_DD\_valid is not set and DD\_retired bit is set, the DMA engine raises the NDDR interrupt for this endpoint and clears the corresponding EPxx\_DMA\_ENABLE bit.

### 13.15.5.3 Transferring the data

For OUT endpoints, the current packet is read from the EP\_RAM by the DMA Engine and transferred to on-chip RAM memory locations starting from DMA\_buffer\_start\_addr. For IN endpoints, the data is fetched from on-chip RAM at DMA\_buffer\_start\_addr and written to the EP\_RAM. The DMA\_buffer\_start\_addr and Present\_DMA\_count fields are updated after each packet is transferred.

### 13.15.5.4 Optimizing descriptor fetch

A DMA transfer normally involves multiple packet transfers. Hardware will not re-fetch a new DD from memory unless the endpoint changes. To indicate an ongoing multi-packet transfer, hardware sets an internal flag called DMA\_PROCEED.

The DMA\_PROCEED flag is cleared after the required number of bytes specified in the DMA\_buffer\_length field is transferred. It is also cleared when the software writes into the USBEPDMADis register. The ability to clear the DMA\_PROCEED flag allows software to force the DD to be re-fetched for the next packet transfer. Writing all zeros into the USBEPDMADis register clears the DMA\_PROCEED flag without disabling DMA operation for any endpoint.

### 13.15.5.5 Ending the packet transfer

On completing a packet transfer, the DMA engine writes back the DD with updated status information to the same memory location from where it was read. The DMA\_buffer\_start\_addr, Present\_DMA\_count, and the DD\_status fields in the DD are updated.

A DD can have the following types of completion:

**Normal completion** - If the current packet is fully transferred and the Present\_DMA\_count field equals the DMA\_buffer\_length, the DD has completed normally. The DD will be written back to memory with DD\_retired set and DD\_status set to NormalCompletion. The EOT interrupt is raised for this endpoint.

**USB transfer end completion** - If the current packet is fully transferred and its size is less than the Max\_packet\_size field, and the end of the DMA buffer is still not reached, the USB transfer end completion occurs. The DD will be written back to the memory with DD\_retired set and DD\_Status set to the DataUnderrun completion code. The EOT interrupt is raised for this endpoint.

**Error completion** - If the current packet is partially transferred i.e. the end of the DMA buffer is reached in the middle of the packet transfer, an error situation occurs. The DD is written back with DD\_retired set and DD\_status set to the DataOverrun status code. The EOT interrupt is raised for this endpoint and the corresponding bit in USBEPDMASt register is cleared. The packet will be re-sent from the endpoint buffer to memory when the corresponding EPxx\_DMA\_ENABLE bit is set again using the USBEPDMAEn register.

### 13.15.5.6 No\_Packet DD

For an IN transfer, if the system does not have any data to send for a while, it can respond to an NDDR interrupt by programming a No\_Packet DD. This is done by setting both the Max\_packet\_size and DMA\_buffer\_length fields in the DD to 0. On processing a No\_Packet DD, the DMA engine clears the DMA request bit in USBDMARSt corresponding to the endpoint without transferring a packet. The DD is retired with a status code of NormalCompletion. This can be repeated as often as necessary. The device will respond to IN token packets on the USB bus with a NAK until a DD with a data packet is programmed and the DMA transfers the packet into the endpoint buffer.

## 13.15.6 Isochronous endpoint operation

For isochronous endpoints, the packet size can vary for each packet. There is one packet per isochronous endpoint for each frame.

### 13.15.6.1 Setting up DMA transfers

Software sets the isochronous endpoint bit to 1 in the DD, and programs the initial value of the Isochronous\_packet\_size\_memory\_address field. All other fields are initialized the same as for non-isochronous endpoints.



For isochronous endpoints, the DMA\_buffer\_length and Present\_DMA\_count fields are in frames rather than bytes.

### 13.15.6.2 Finding the DMA Descriptor

Finding the descriptors is done in the same way as that for a non-isochronous endpoint.

A DMA request will be placed for DMA-enabled isochronous endpoints on every FRAME interrupt. On processing the request, the DMA engine will fetch the descriptor and if Isochronous\_endpoint is set, will fetch the Isochronous\_packetsize\_memory\_address from the fifth word of the DD.

### 13.15.6.3 Transferring the Data

The data is transferred to or from the memory location DMA\_buffer\_start\_addr. After the end of the packet transfer the Present\_DMA\_count value is incremented by 1.

The isochronous packet size is stored in memory as shown in [Figure 43](#). Each word in the packet size memory shown is divided into fields: Frame\_number (bits 31 to 17), Packet\_valid (bit 16), and Packet\_length (bits 15 to 0). The space allocated for the packet size memory for a given DD should be DMA\_buffer\_length words in size – one word for each packet to transfer.

#### OUT endpoints

At the completion of each frame, the packet size is written to the address location in Isochronous\_packet\_size\_memory\_address, and Isochronous\_packet\_size\_memory\_address is incremented by 4.

#### IN endpoints

Only the Packet\_length field of the isochronous packet size word is used. For each frame, an isochronous data packet of size specified by this field is transferred from the USB device to the host, and Isochronous\_packet\_size\_memory\_address is incremented by 4 at the end of the packet transfer. If Packet\_length is zero, an empty packet will be sent by the USB device.

### 13.15.6.4 DMA descriptor completion

DDs for isochronous endpoints can only end with a status code of NormalCompletion since there is no short packet on Isochronous endpoints, and the USB transfer continues indefinitely until a SystemError occurs. There is no DataOverrun detection for isochronous endpoints.

### 13.15.6.5 Isochronous OUT Endpoint Operation Example

Assume that an isochronous endpoint is programmed for the transfer of 10 frames and that the transfer begins when the frame number is 21. After transferring four frames with packet sizes of 10, 15, 8 and 20 bytes without errors, the descriptor and memory map appear as shown in [Figure 43](#).

The\_total\_number\_of\_bytes\_transferred = 0x0A + 0x0F + 0x08 + 0x14 = 0x35.

The Packet\_valid bit (bit 16) of all the words in the packet length memory is set to 1.

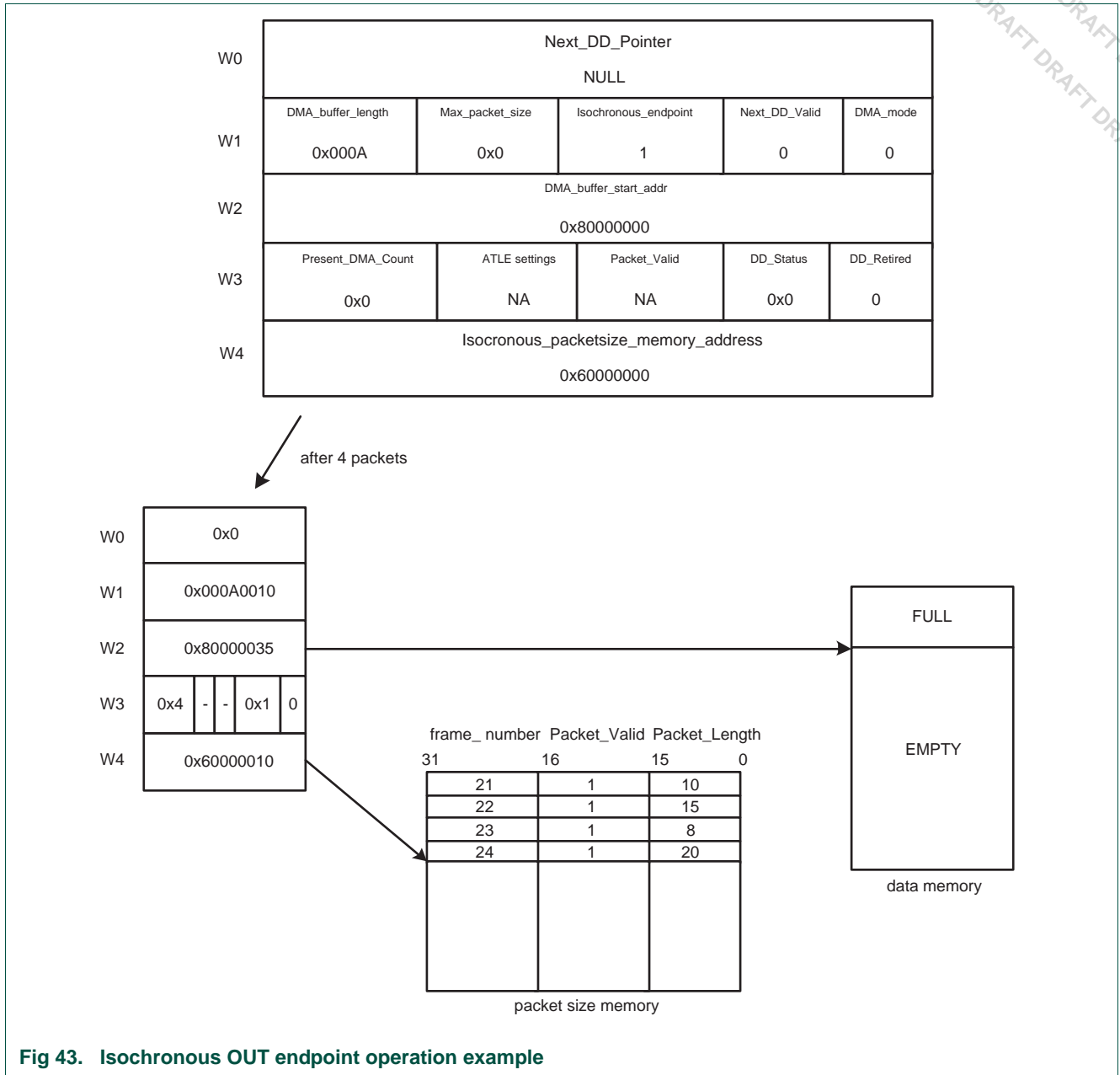


Fig 43. Isochronous OUT endpoint operation example

### 13.15.7 Auto Length Transfer Extraction (ATLE) mode operation

Some host drivers such as NDIS (Network Driver Interface Specification) host drivers are capable of concatenating small USB transfers (delta transfers) to form a single large USB transfer. For OUT USB transfers, the device hardware has to break up this concatenated transfer back into the original delta transfers and transfer them to separate DMA buffers. This is achieved by setting the DMA mode to Auto Transfer Length Extraction (ATLE) mode in the DMA descriptor. ATLE mode is supported for Bulk endpoints only.

#### OUT transfers in ATLE mode

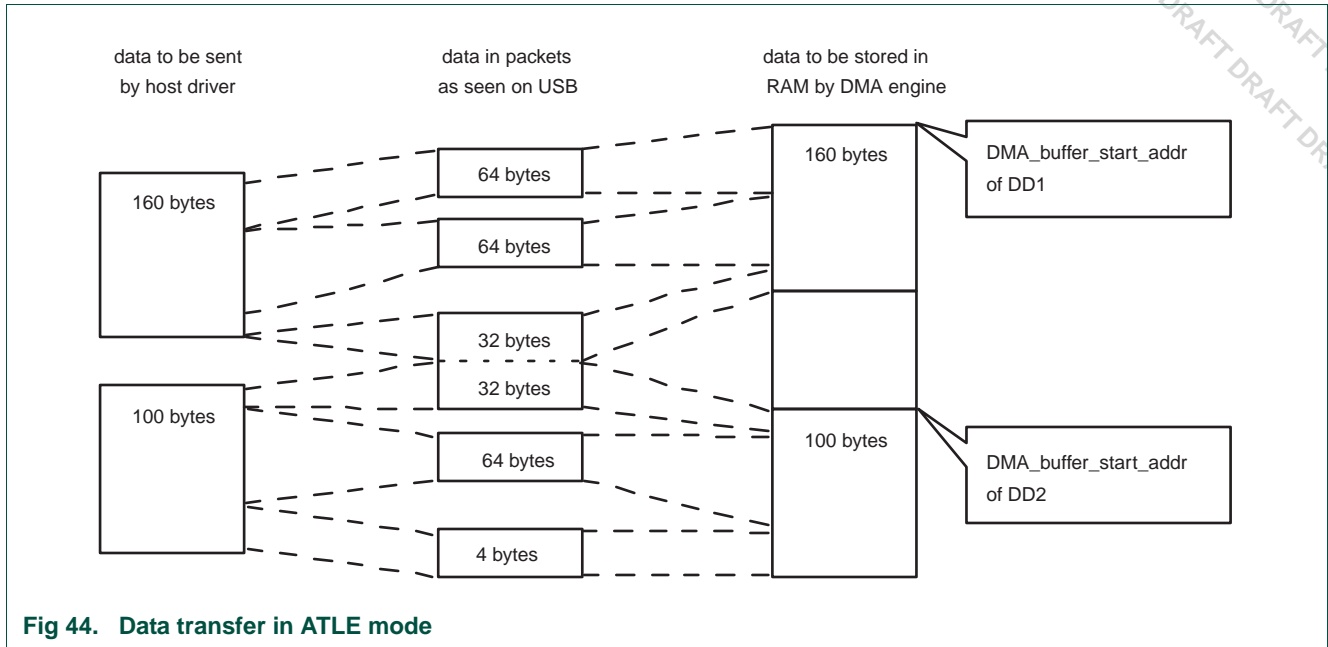


Fig 44. Data transfer in ATLE mode

Figure 44 shows a typical OUT USB transfer in ATLE mode, where the host concatenates two USB transfers of 160 bytes and 100 bytes, respectively. Given a MaxPacketSize of 64, the device hardware interprets this USB transfer as four packets of 64 bytes and a short packet of 4 bytes. The third and fourth packets are concatenated. Note that in Normal mode, the USB transfer would be interpreted as packets of 64, 64, 32, and 64 and 36 bytes.

It is now the responsibility of the DMA engine to separate these two USB transfers and put them in the memory locations in the DMA\_buffer\_start\_addr field of DMA Descriptor 1 (DD1) and DMA Descriptor 2 (DD2).

Hardware reads the two-byte-wide DMA\_buffer\_length at the offset (from the start of the USB transfer) specified by Message\_length\_position from the incoming data packets and writes it in the DMA\_buffer\_length field of the DD. To ensure that both bytes of the DMA\_buffer\_length are extracted in the event they are split between two packets, the flags LS\_byte\_extracted and MS\_byte\_extracted are set by hardware after the respective byte is extracted. After the extraction of the MS byte, the DMA transfer continues as in the normal mode.

The flags LS\_byte\_extracted and MS\_byte\_extracted are set to 0 by software when preparing a new DD. Therefore, once a DD is retired, the transfer length is extracted again for the next DD.

If DD1 is retired during the transfer of a concatenated packet (such as the third packet in Figure 44), and DD2 is not programmed (Next\_DD\_valid field of DD1 is 0), then DD1 is retired with DD\_status set to the DataOverrun status code. This is treated as an error condition and the corresponding EPxx\_DMA\_ENABLE bit of USBEPDMASt is cleared by hardware.

In ATLE mode, the last buffer length to be transferred always ends with a short or empty packet indicating the end of the USB transfer. If the concatenated transfer lengths are such that the USB transfer ends on a MaxPacketSize packet boundary, the (NDIS) host will send an empty packet to mark the end of the USB transfer.

### IN transfers in ATLE mode

For IN USB transfers from the device to the host, DMA\_buffer\_length is set by the device software as in normal mode.

In ATLE mode, the device concatenates data from multiple DDs to form a single USB transfer. If a DD is retired in the middle of a packet (packet size is less than MaxPacketSize), the next DD referenced by Next\_DD\_pointer is fetched, and the remaining bytes to form a packet of MaxPacketSize are transferred from the next DD's buffer.

If the next DD is not programmed (i.e. Next\_DD\_valid field in DD is 0), and the DMA buffer length for the current DD has completed before the MaxPacketSize packet boundary, then the available bytes from current DD are sent as a short packet on USB, which marks the end of the USB transfer for the host.

If the last buffer length completes on a MaxPacketSize packet boundary, the device software must program the next DD with DMA\_buffer\_length field 0, so that an empty packet is sent by the device to mark the end of the USB transfer for the host.

#### 13.15.7.1 Setting up the DMA transfer

For OUT endpoints, the host hardware needs to set the field Message\_length\_position in the DD. This indicates the start location of the message length in the incoming data packets. Also the device software has to set the DMA\_buffer\_length field to 0 for OUT endpoints because this field is updated by the device hardware after the extraction of the buffer length.

For IN endpoints, descriptors are set in the same way as in normal mode operation.

Since a single packet can be split between two DDs, software should always keep two DDs ready, except for the last DMA transfer which ends with a short or empty packet.

#### 13.15.7.2 Finding the DMA Descriptor

DMA descriptors are found in the same way as the normal mode operation.

#### 13.15.7.3 Transferring the Data

##### OUT endpoints

If the LS\_byte\_extracted or MS\_byte\_extracted bit in the status field is not set, the hardware will extract the transfer length from the data stream and program DMA\_buffer\_length. Once the extraction is complete both the LS\_byte\_extracted and MS\_byte\_extracted bits will be set.

##### IN endpoints

The DMA transfer proceeds as in normal mode and continues until the number of bytes transferred equals the DMA\_buffer\_length.

#### 13.15.7.4 Ending the packet transfer

The DMA engine proceeds with the transfer until the number of bytes specified in the field `DMA_buffer_length` is transferred to or from on-chip RAM. Then the EOT interrupt will be generated. If this happens in the middle of the packet, the linked DD will get loaded and the remaining part of the packet gets transferred to or from the address pointed by the new DD.

##### OUT endpoints

If the linked DD is not valid and the packet is partially transferred to memory, the DD ends with `DataOverrun` status code set, and the DMA will be disabled for this endpoint. Otherwise `DD_status` will be updated with the `NormalCompletion` status code.

##### IN endpoints

If the linked DD is not valid and the packet is partially transferred to USB, the DD ends with a status code of `NormalCompletion` in the `DD_status` field. This situation corresponds to the end of the USB transfer, and the packet will be sent as a short packet. Also, when the linked DD is valid and buffer length is 0, an empty packet will be sent to indicate the end of the USB transfer.

## 13.16 Double buffered endpoint operation

The Bulk and Isochronous endpoints of the USB Device Controller are double buffered to increase data throughput.

When a double-buffered endpoint is realized, enough space for both endpoint buffers is automatically allocated in the EP\_RAM. See [Section 13.10.5.1](#).

For the following discussion, the endpoint buffer currently accessible to the CPU or DMA engine for reading or writing is said to be the active buffer.

### 13.16.1 Bulk endpoints

For Bulk endpoints, the active endpoint buffer is switched by the SIE Clear Buffer or Validate Buffer commands.

The following example illustrates how double buffering works for a Bulk OUT endpoint in Slave mode:

Assume that both buffer 1 (B\_1) and buffer 2 (B\_2) are empty, and that the active buffer is B\_1.

1. The host sends a data packet to the endpoint. The device hardware puts the packet into B\_1, and generates an endpoint interrupt.
2. Software clears the endpoint interrupt and begins reading the packet data from B\_1. While B\_1 is still being read, the host sends a second packet, which device hardware places in B\_2, and generates an endpoint interrupt.
3. Software is still reading from B\_1 when the host attempts to send a third packet. Since both B\_1 and B\_2 are full, the device hardware responds with a NAK.
4. Software finishes reading the first packet from B\_1 and sends a SIE Clear Buffer command to free B\_1 to receive another packet. B\_2 becomes the active buffer.
5. Software sends the SIE Select Endpoint command to read the Select Endpoint Register and test the FE bit. Software finds that the active buffer (B\_2) has data (FE=1). Software clears the endpoint interrupt and begins reading the contents of B\_2.
6. The host re-sends the third packet which device hardware places in B\_1. An endpoint interrupt is generated.
7. Software finishes reading the second packet from B\_2 and sends a SIE Clear Buffer command to free B\_2 to receive another packet. B\_1 becomes the active buffer. Software waits for the next endpoint interrupt to occur (it already has been generated back in step 6).
8. Software responds to the endpoint interrupt by clearing it and begins reading the third packet from B\_1.
9. Software finishes reading the third packet from B\_1 and sends a SIE Clear Buffer command to free B\_1 to receive another packet. B\_2 becomes the active buffer.
10. Software tests the FE bit and finds that the active buffer (B\_2) is empty (FE=0).
11. Both B\_1 and B\_2 are empty. Software waits for the next endpoint interrupt to occur. The active buffer is now B\_2. The next data packet sent by the host will be placed in B\_2.

The following example illustrates how double buffering works for a Bulk IN endpoint in Slave mode:

Assume that both buffer 1 (B\_1) and buffer 2 (B\_2) are empty and that the active buffer is B\_1. The interrupt on NAK feature is enabled.

1. The host requests a data packet by sending an IN token packet. The device responds with a NAK and generates an endpoint interrupt.
2. Software clears the endpoint interrupt. The device has three packets to send. Software fills B\_1 with the first packet and sends a SIE Validate Buffer command. The active buffer is switched to B\_2.
3. Software sends the SIE Select Endpoint command to read the Select Endpoint Register and test the FE bit. It finds that B\_2 is empty (FE=0) and fills B\_2 with the second packet. Software sends a SIE Validate Buffer command, and the active buffer is switched to B\_1.
4. Software waits for the endpoint interrupt to occur.
5. The device successfully sends the packet in B\_1 and clears the buffer. An endpoint interrupt occurs.
6. Software clears the endpoint interrupt. Software fills B\_1 with the third packet and validates it using the SIE Validate Buffer command. The active buffer is switched to B\_2.
7. The device successfully sends the second packet from B\_2 and generates an endpoint interrupt.
8. Software has no more packets to send, so it simply clears the interrupt.
9. The device successfully sends the third packet from B\_1 and generates an endpoint interrupt.
10. Software has no more packets to send, so it simply clears the interrupt.
11. Both B\_1 and B\_2 are empty, and the active buffer is B\_2. The next packet written by software will go into B\_2.

In DMA mode, switching of the active buffer is handled automatically in hardware. For Bulk IN endpoints, proactively filling an endpoint buffer to take advantage of the double buffering can be accomplished by manually starting a packet transfer using the USBDMARSet register.

### 13.16.2 Isochronous endpoints

For isochronous endpoints, the active data buffer is switched by hardware when the FRAME interrupt occurs. The SIE Clear Buffer and Validate Buffer commands do not cause the active buffer to be switched.

Double buffering allows the software to make full use of the frame interval writing or reading a packet to or from the active buffer, while the packet in the other buffer is being sent or received on the bus.

For an OUT isochronous endpoint, any data not read from the active buffer before the end of the frame is lost when it switches.

For an IN isochronous endpoint, if the active buffer is not validated before the end of the frame, an empty packet is sent on the bus when the active buffer is switched, and its contents will be overwritten when it becomes active again.



### 14.1 How to read this chapter

---

This chapter describes the USB host controller which is present on some LPC178x/177x devices (see [Section 1.4](#) for details). On these devices, the USB controller can be configured for device, Host, or OTG operation.

### 14.2 Basic configuration

---

The USB controller is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCUSB.  
**Remark:** On reset, the USB block is disabled (PCUSB = 0).
2. Clock: The USB block can be used with the Alt PLL (PLL1) to obtain the USB clock or with the Main PLL (PLL0). See [Section 4.5.2](#).
3. Pins: Select USB pins and their modes in the relevant IOCON registers ([Section 8.4.1](#)).
4. Wake-up: Activity on the USB bus port can wake up the microcontroller from Power-down mode, see [Section 4.7.8](#).
5. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
6. Initialization: see [Section 15.11](#).

## 14.3 Introduction

This section describes the host portion of the USB 2.0 OTG dual role core which integrates the host controller (OHCI compliant), device controller, and I<sup>2</sup>C interface. The I<sup>2</sup>C interface controls the external OTG ATX.

The USB is a 4 wire bus that supports communication between a host and a number (127 max.) of peripherals. The host controller allocates the USB bandwidth to attached devices through a token based protocol. The bus supports hot plugging, un-plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The host controller enables data exchange with various USB devices attached to the bus. It consists of register interface, serial interface engine and DMA controller. The register interface complies to the OHCI specification.

**Table 319. USB (OHCI) related acronyms and abbreviations used in this chapter**

Acronym/abbreviation	Description
AHB	Advanced High-Performance Bus
ATX	Analog Transceiver
DMA	Direct Memory Access
FS	Full Speed
LS	Low Speed
OHCI	Open Host Controller Interface
USB	Universal Serial Bus

### 14.3.1 Features

- OHCI compliant.
- OpenHCI specifies the operation and interface of the USB Host Controller and SW Driver
  - USBOperational: Process Lists and generate SOF Tokens.
  - USBReset: Forces reset signaling on the bus, SOF disabled.
  - USBSuspend: Monitor USB for wake-up activity.
  - USBResume: Forces resume signaling on the bus.
- The Host Controller has four USB states visible to the SW Driver.
- HCCA register points to Interrupt and Isochronous Descriptors List.
- ControlHeadED and BulkHeadED registers point to Control and Bulk Descriptors List.

### 14.3.2 Architecture

The architecture of the USB host controller is shown below in [Figure 45](#).

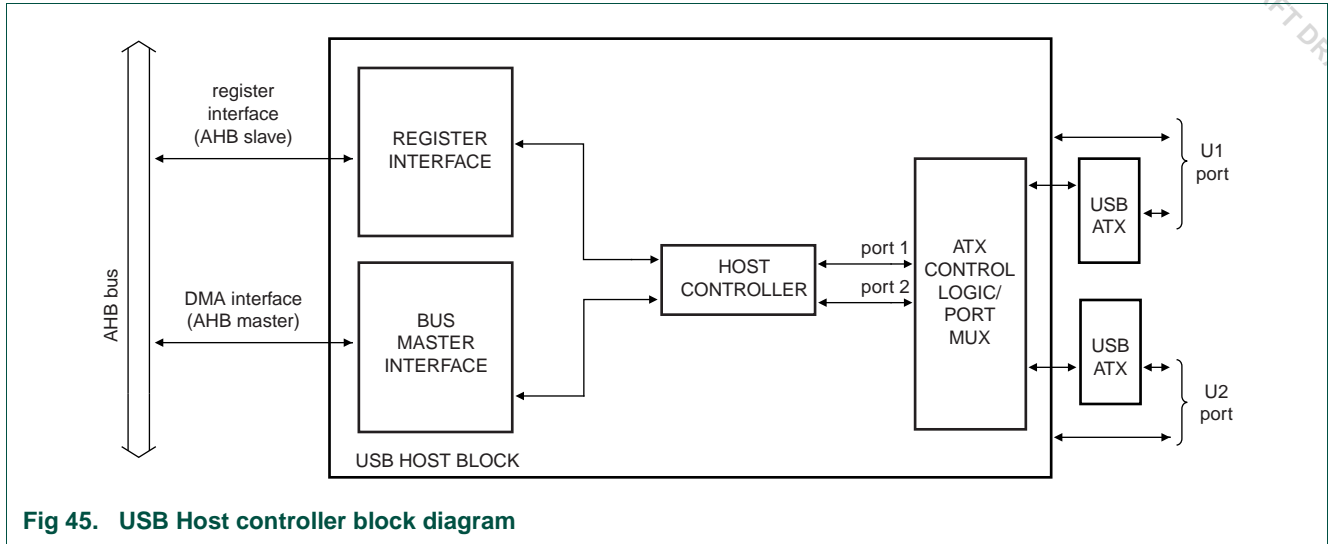


Fig 45. USB Host controller block diagram

## 14.4 Interfaces

The OTG controller has two USB ports indicated by suffixes 1 and 2 in the USB pin names and referred to as USB port 1 (U1) and USB port 2 (U2) in the following text.

### 14.4.1 Pin description

Table 320. USB OTG port pins

Pin name	Direction	Description	Pin category
V <sub>BUS</sub>	I	V <sub>BUS</sub> status input. When this function is not enabled via its corresponding IOCON register, it is driven HIGH internally.	USB Connector
<b>Port U1</b>			
USB_D+1	I/O	Positive differential data	USB Connector
USB_D-1	I/O	Negative differential data	USB Connector
USB_CONNECT1	O	SoftConnect control signal	Control
USB_UP_LED1	O	GoodLink LED control signal	Control
USB_INT1	I	OTG ATX interrupt	External OTG transceiver
USB_SCL1	I/O	I <sup>2</sup> C serial clock	External OTG transceiver
USB_SDA1	I/O	I <sup>2</sup> C serial data	External OTG transceiver
USB_TX_E1	O	Transmit enable	External OTG transceiver
USB_TX_DP1	O	D+ transmit data	External OTG transceiver
USB_TX_DM1	O	D- transmit data	External OTG transceiver
USB_RCV1	I	Differential receive data	External OTG transceiver
USB_RX_DP1	I	D+ receive data	External OTG transceiver
USB_RX_DM1	I	D- receive data	External OTG transceiver
USB_LS1	O	Low speed status (applies to host functionality only)	External OTG transceiver
USB_SSPND1	O	Bus suspend status	External OTG transceiver
USB_PPWR1	O	Port power enable	Host power switch
USB_PWRD1	I	Port power status	Host power switch
USB_OVRCR1	I	Over-current status	Host power switch
USB_HSTEN1	O	Host enabled status	
<b>Port U2</b>			
USB_D+2	I/O	Positive differential data	USB Connector
USB_D-2	I/O	Negative differential data	USB Connector
USB_CONNECT2	O	SoftConnect control signal	Control
USB_UP_LED2	O	GoodLink LED control signal	Control
USB_PPWR2	O	Port power enable	Host power switch
USB_PWRD2	I	Port power status	Host power switch
USB_OVRCR2	I	Over-current status	Host power switch
USB_HSTEN2	O	Host enabled status	Control

#### 14.4.1.1 USB host usage note

Both ports can be configured as USB hosts. For details on how to connect the USB ports, see the USB OTG chapter, [Section 15.7](#).

The USB device/host/OTG controller is disabled after RESET and must be enabled by writing a 1 to the PCUSB bit in the PCONP register, see [Table 37](#).

### 14.4.2 Software interface

The software interface of the USB host block consists of a register view and the format definitions for the endpoint descriptors. For details on these two aspects see the OHCI specification. The register map is shown in the next subsection.

#### 14.4.2.1 Register map

The following registers are located in the AHB clock 'cclk' domain. They can be accessed directly by the processor. All registers are 32 bits wide and aligned in the word address boundaries.

**Table 321. USB Host register address definitions**

Name	Function	Address	R/W <sup>U</sup>	Reset value
HcRevision	BCD representation of the version of the HCI specification that is implemented by the Host Controller.	0x2008 C000	R	0x10
HcControl	Defines the operating modes of the HC.	0x2008 C004	R/W	0
HcCommandStatus	This register is used to receive the commands from the Host Controller Driver (HCD). It also indicates the status of the HC.	0x2008 C008	R/W	0
HcInterruptStatus	Indicates the status on various events that cause hardware interrupts by setting the appropriate bits.	0x2008 C00C	R/W	0
HcInterruptEnable	Controls the bits in the HcInterruptStatus register and indicates which events will generate a hardware interrupt.	0x2008 C010	R/W	0
HcInterruptDisable	The bits in this register are used to disable corresponding bits in the HcInterruptStatus register and in turn disable that event leading to hardware interrupt.	0x2008 C014	R/W	0
HcHCCA	Contains the physical address of the host controller communication area.	0x2008 C018	R/W	0
HcPeriodCurrentED	Contains the physical address of the current isochronous or interrupt endpoint descriptor.	0x2008 C01C	R	0
HcControlHeadED	Contains the physical address of the first endpoint descriptor of the control list.	0x2008 C020	R/W	0
HcControlCurrentED	Contains the physical address of the current endpoint descriptor of the control list	0x2008 C024	R/W	0
HcBulkHeadED	Contains the physical address of the first endpoint descriptor of the bulk list.	0x2008 C028	R/W	0
HcBulkCurrentED	Contains the physical address of the current endpoint descriptor of the bulk list.	0x2008 C02C	R/W	0
HcDoneHead	Contains the physical address of the last transfer descriptor added to the 'Done' queue.	0x2008 C030	R	0
HcFmInterval	Defines the bit time interval in a frame and the full speed maximum packet size which would not cause an overrun.	0x2008 C034	R/W	0x2EDF
HcFmRemaining	A 14-bit counter showing the bit time remaining in the current frame.	0x2008 C038	R	0
HcFmNumber	Contains a 16-bit counter and provides the timing reference among events happening in the HC and the HCD.	0x2008 C03C	R	0

Table 321. USB Host register address definitions ...continued

Name	Function	Address	R/W <sup>[1]</sup>	Reset value
HcPeriodicStart	Contains a programmable 14-bit value which determines the earliest time HC should start processing a periodic list.	0x2008 C040	R/W	0
HcLSThreshold	Contains 11-bit value which is used by the HC to determine whether to commit to transfer a maximum of 8-byte LS packet before EOF.	0x2008 C044	R/W	0x628h
HcRhDescriptorA	First of the two registers which describes the characteristics of the root hub.	0x2008 C048	R/W	0xFF000902
HcRhDescriptorB	Second of the two registers which describes the characteristics of the Root Hub.	0x2008 C04C	R/W	0x60000h
HcRhStatus	This register is divided into two parts. The lower D-word represents the hub status field and the upper word represents the hub status change field.	0x2008 C050	R/W	0
HcRhPortStatus[1]	Controls and reports the port events on a per-port basis.	0x2008 C054	R/W	0
HcRhPortStatus[2]	Controls and reports the port events on a per port basis.	0x2008 C058	R/W	0
Module_ID/ Ver_Rev_ID	IP number, where yy (0x00) is unique version number and zz (0x00) is a unique revision number.	0x2008 C0FC	R	0x3505yyzz

- [1] The R/W column lists the accessibility of the register:
- a) Registers marked 'R' for access will return their current value when read.
  - b) Registers marked 'R/W' allow both read and write.

### 14.4.2.2 USB Host Register Definitions

Refer to the OHCI specification document on the Compaq website for register definitions.

### 15.1 How to read this chapter

---

This chapter describes the USB OTG controller which is present on some LPC178x/177x devices (see [Section 1.4](#) for details). On these devices, the USB controller can be configured for device, Host, or OTG operation.

### 15.2 Basic configuration

---

The USB controller is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCUSB.  
**Remark:** On reset, the USB block is disabled (PCUSB = 0).
2. Clock: The USB clock can be generated using the Alt PLL (PLL1) or with the Main PLL (PLL0). See [Section 4.5.2](#).
3. Pins: Select USB pins and their modes in the relevant IOCON registers ([Section 8.4.1](#)).
4. Wake-up: Activity on the USB bus port can wake up the microcontroller from Power-down mode (see [Section 15.10.2](#) and [Section 4.7.8](#)).
5. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
6. Initialization: see [Section 15.11](#).

### 15.3 Introduction

---

This chapter describes the OTG and I<sup>2</sup>C portions of the USB 2.0 OTG dual role device controller which integrates the (OHCI) host controller, device controller, and I<sup>2</sup>C. The I<sup>2</sup>C interface that is part of the USB block is intended to control an external OTG transceiver, and is not the same as the I<sup>2</sup>C peripherals described in [Section 22.1](#).

USB OTG (On-The-Go) is a supplement to the USB 2.0 specification that augments the capability of existing mobile devices and USB peripherals by adding host functionality for connection to USB peripherals. The specification and more information on USB OTG can be found on the USB Implementers Forum web site.

### 15.4 Features

---

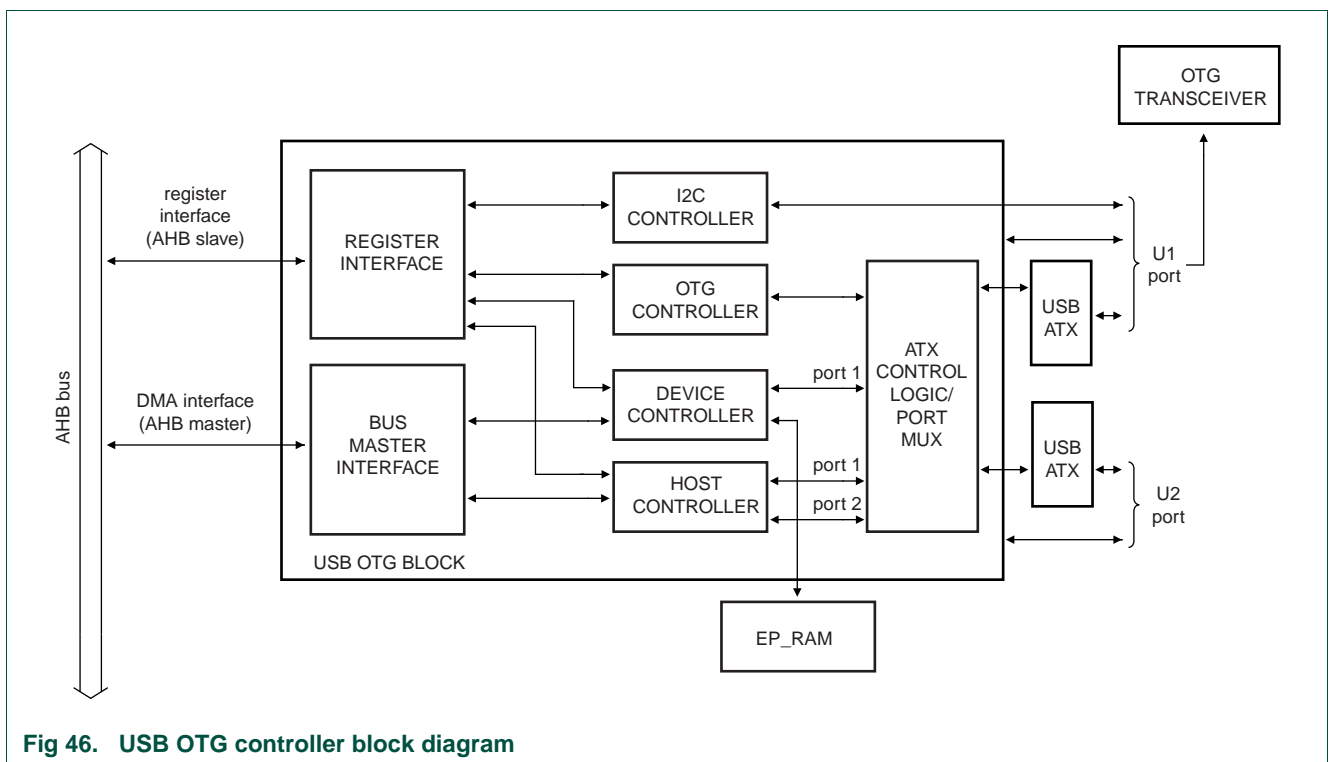
- Fully compliant with On-The-Go supplement to the *USB 2.0 Specification, Revision 1.0a*.
- Hardware support for Host Negotiation Protocol (HNP).
- Includes a programmable timer required for HNP and SRP.
- Supports any OTG transceiver compliant with the *OTG Transceiver Specification (CEA-2011), Rev. 1.0*.

## 15.5 Architecture

The architecture of the USB OTG controller is shown below in the block diagram.

The host, device, OTG, and I<sup>2</sup>C controllers can be programmed through the register interface. The OTG controller enables dynamic switching between host and device roles through the HNP protocol. One port may be connected to an external OTG transceiver to support an OTG connection. The communication between the register interface and an external OTG transceiver is handled through an I<sup>2</sup>C interface and through the external OTG transceiver interrupt signal.

For USB connections that use the device or host controller only (not OTG), the ports use an embedded USB Analog Transceiver (ATX).



## 15.6 Modes of operation

The OTG controller is capable of operating in the following modes:

- One dual-role OTG port and optionally another Host port (see [Figure 47](#) and [Figure 48](#))
- Two Host ports (see [Figure 49](#))
- One Host port and one Device port (see [Figure 50](#))



## 15.7 Pin configuration

The OTG controller has two USB ports indicated by suffixes 1 and 2 in the USB pin names and referred to as USB port 1 (U1) and USB port 2 (U2) in the following text.

**Table 322. USB OTG port 1 pins**

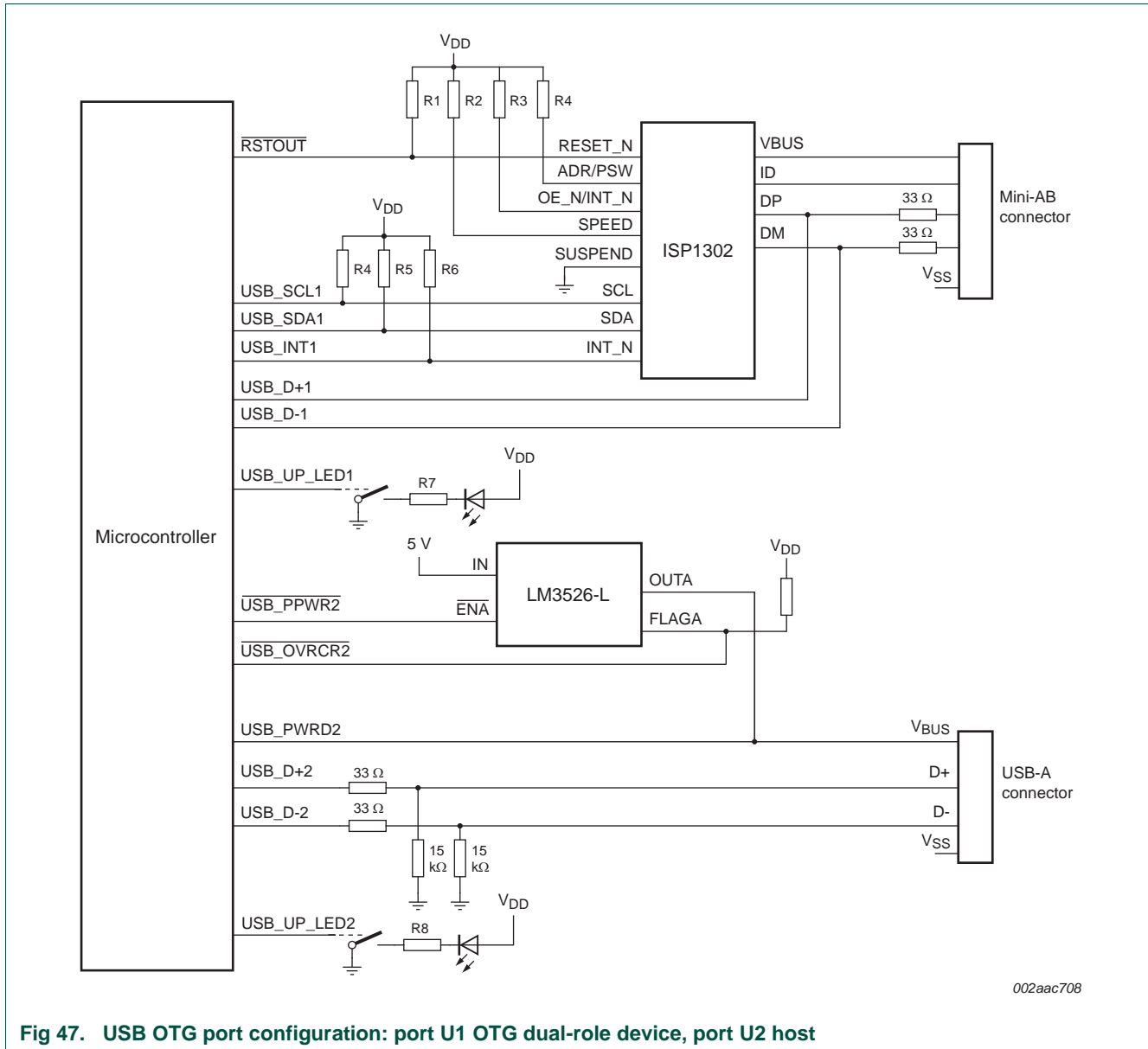
Pin name	Direction	Description	Pin category
V <sub>BUS</sub>	I	V <sub>BUS</sub> status input. When this function is not enabled via its corresponding IOCON register, it is driven HIGH internally.	USB Connector
<b>Port U1</b>			
USB_D+1	I/O	Positive differential data	USB Connector
USB_D-1	I/O	Negative differential data	USB Connector
USB_CONNECT1	O	SoftConnect control signal	Control
USB_UP_LED1	O	GoodLink LED control signal	Control
USB_INT1	I	OTG ATX interrupt	External OTG transceiver
USB_SCL1	I/O	I <sup>2</sup> C serial clock	External OTG transceiver
USB_SDA1	I/O	I <sup>2</sup> C serial data	External OTG transceiver
USB_TX_E1	O	Transmit enable	External OTG transceiver
USB_TX_DP1	O	D+ transmit data	External OTG transceiver
USB_TX_DM1	O	D- transmit data	External OTG transceiver
USB_RCV1	I	Differential receive data	External OTG transceiver
USB_RX_DP1	I	D+ receive data	External OTG transceiver
USB_RX_DM1	I	D- receive data	External OTG transceiver
USB_LS1	O	Low speed status (applies to host functionality only)	External OTG transceiver
USB_SSPND1	O	Bus suspend status	External OTG transceiver
USB_PPWR1	O	Port power enable	Host power switch
USB_PWRD1	I	Port power status	Host power switch
USB_OVRCCR1	I	Over-current status	Host power switch
USB_HSTEN1	O	Host enabled status	
<b>Port U2</b>			
USB_D+2	I/O	Positive differential data	USB Connector
USB_D-2	I/O	Negative differential data	USB Connector
USB_CONNECT2	O	SoftConnect control signal	Control
USB_UP_LED2	O	GoodLink LED control signal	Control
USB_PPWR2	O	Port power enable	Host power switch
USB_PWRD2	I	Port power status	Host power switch
USB_OVRCCR2	I	Over-current status	Host power switch
USB_HSTEN2	O	Host enabled status	Control

### 15.7.1 Using port U1 for OTG operation

The following figures show different ways to realize connections to a USB device using ports U1 and U2. The example described here uses an ISP1302 (ST-Ericsson) for the external OTG transceiver and the USB Host power switch LM3526-L (National Semiconductors). There are two ways to connect the OTG transceiver:

1. Use the internal USB transceiver for USB signalling and use the external OTG transceiver for OTG functionality only (see [Figure 47](#)). This option uses the internal transceiver in VP/VM mode.
2. Use the external OTG transceiver in VP/VM mode for OTG functionality and USB signalling (see [Figure 48](#)).

In both cases port U2 is connected as a host. Solution one uses fewer pins.



002aac708

Fig 47. USB OTG port configuration: port U1 OTG dual-role device, port U2 host

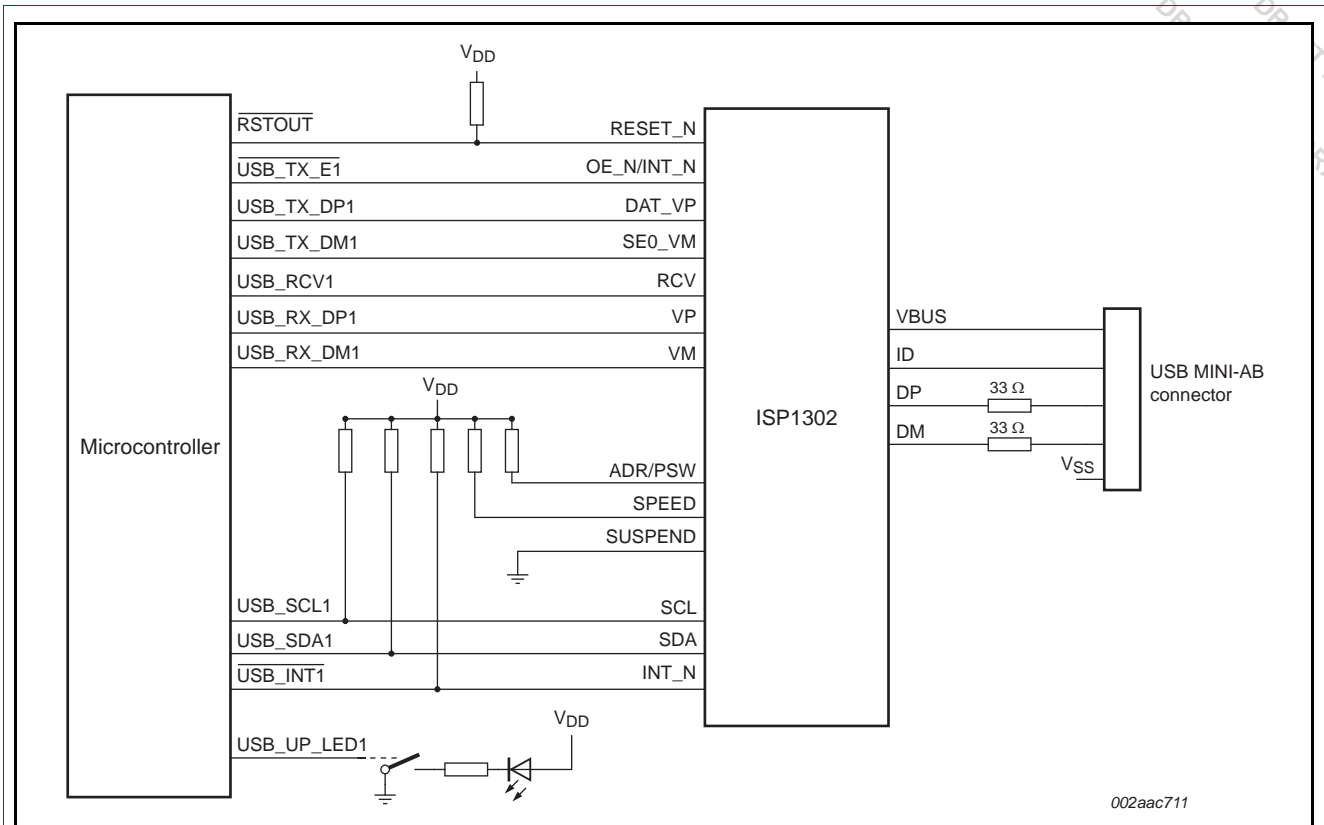


Fig 48. USB OTG port configuration: VP\_VM mode

15.7.2 Using both ports U1 and U2 for host operation

Both ports U1 and U2 are connected as hosts using an embedded USB transceiver. There is no OTG functionality on the port.

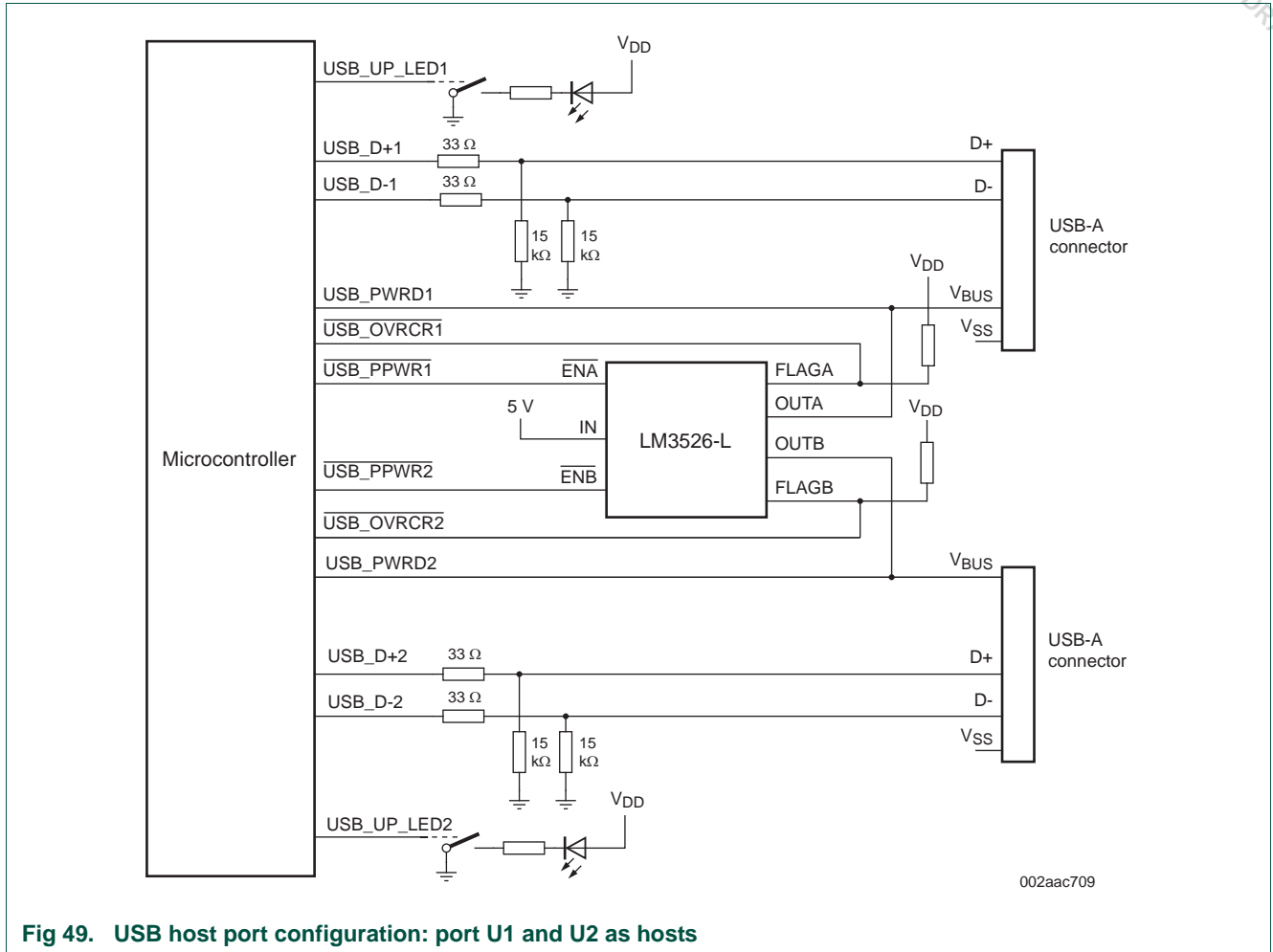


Fig 49. USB host port configuration: port U1 and U2 as hosts

15.7.3 Using U1 for host operation and U2 for device operation

Port U2 is connected as device, and port U1 is connected as host. Both ports use embedded USB transceivers. There is no OTG functionality on either USB port.

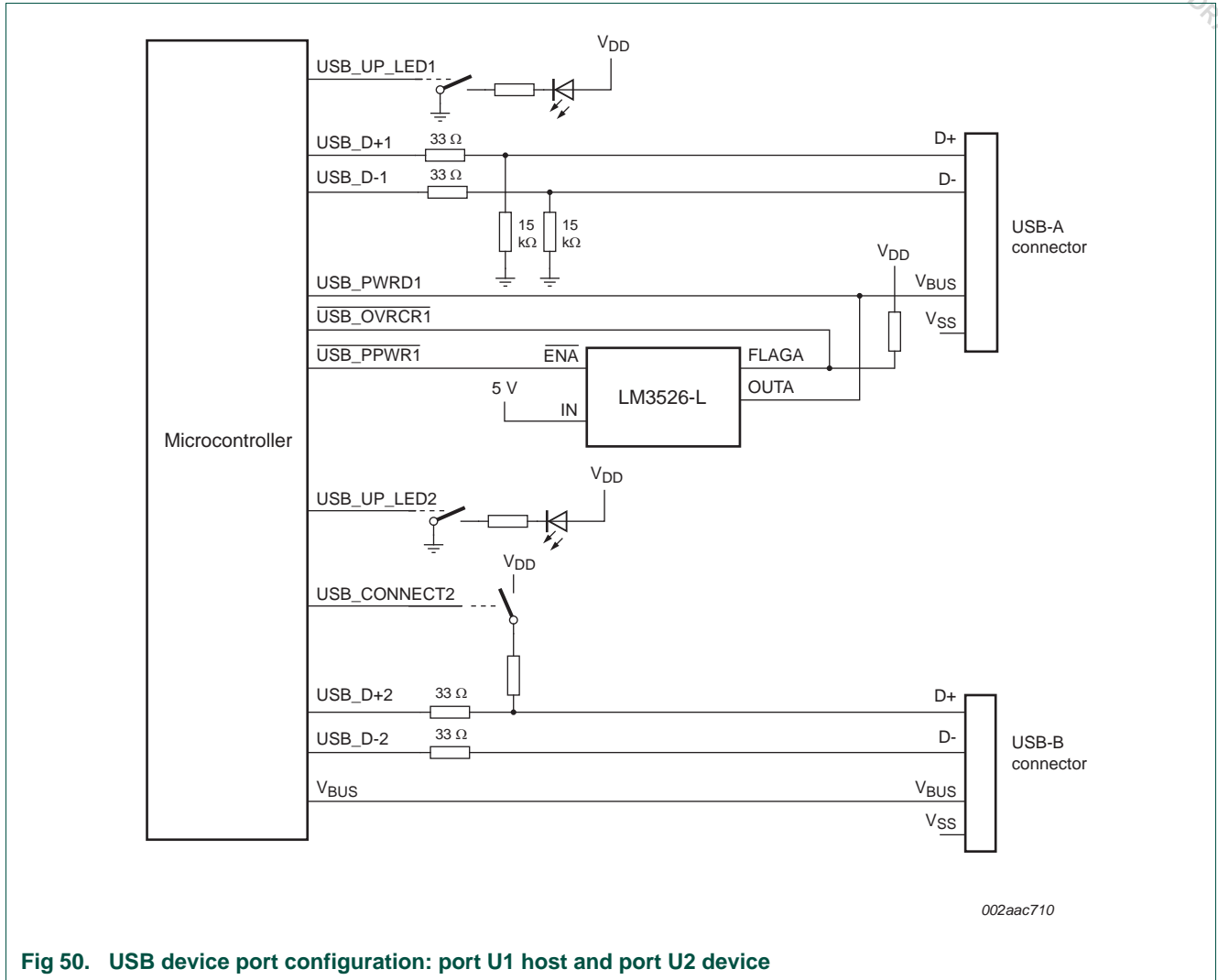


Fig 50. USB device port configuration: port U1 host and port U2 device

## 15.8 Register description

The OTG and I<sup>2</sup>C registers are summarized in the following table.

The Device and Host registers are explained in [Table 254](#) and [Table 321](#) in the USB Device Controller and USB Host (OHCI) Controller chapters. All registers are 32 bits wide and aligned to word address boundaries.

**Table 323. USB OTG and I<sup>2</sup>C register address definitions**

Name	Description	Access	Reset value	Address	Table
<b>Interrupt register</b>					
USBIntSt	USB Interrupt Status	R/W	0x8000 0100	0x400F C1C0	<a href="#">324</a>
<b>OTG registers</b>					
OTGIntSt	OTG Interrupt Status	RO	0	0x2008 C100	<a href="#">325</a>
OTGIntEn	OTG Interrupt Enable	R/W	0	0x2008 C104	<a href="#">325</a>
OTGIntSet	OTG Interrupt Set	WO	NA	0x2008 C108	<a href="#">325</a>
OTGIntClr	OTG Interrupt Clear	WO	NA	0x2008 C10C	<a href="#">325</a>
OTGStCtrl <sup>[1]</sup>	OTG Status and Control	R/W	0	0x2008 C110	<a href="#">326</a>
OTGTmr	OTG Timer	R/W	0xFFFF	0x2008 C114	<a href="#">327</a>
<b>I<sup>2</sup>C registers</b>					
I2C_RX	I <sup>2</sup> C Receive	RO	NA	0x2008 C300	<a href="#">330</a>
I2C_TX	I <sup>2</sup> C Transmit	WO	NA	0x2008 C300	<a href="#">331</a>
I2C_STS	I <sup>2</sup> C Status	RO	0x0A00	0x2008 C304	<a href="#">332</a>
I2C_CTL	I <sup>2</sup> C Control	R/W	0	0x2008 C308	<a href="#">333</a>
I2C_CLKHI	I <sup>2</sup> C Clock High	R/W	0xB9	0x2008 C30C	<a href="#">334</a>
I2C_CLKLO	I <sup>2</sup> C Clock Low	WO	0xB9	0x2008 C310	<a href="#">335</a>
<b>Clock control registers</b>					
OTGClkCtrl	OTG clock controller	R/W	0	0x2008 CFF4	<a href="#">328</a>
OTGClkSt	OTG clock status	RO	0	0x2008 CFF8	<a href="#">329</a>

[1] Bits 0 and 1 of this register are used to control the routing of the USB pins to ports 1 and 2 in device-only applications (see [Section 13.10.1](#)).

### 15.8.1 USB Interrupt Status Register (USBIntSt - 0x2008 C1C0)

The USB OTG controller has seven interrupt lines. The interrupt lines are ORed together to a single channel of the vectored interrupt controller. This register allows software to determine their status with a single read operation.

**Table 324. USB Interrupt Status register - (USBIntSt - address 0x2008 C1C0) bit description**

Bit	Symbol	Description	Reset Value
0	USB_INT_REQ_LP	Low priority interrupt line status. This bit is read-only.	0
1	USB_INT_REQ_HP	High priority interrupt line status. This bit is read-only.	0
2	USB_INT_REQ_DMA	DMA interrupt line status. This bit is read-only.	0
3	USB_HOST_INT	USB host interrupt line status. This bit is read-only.	0
4	USB_ATX_INT	External ATX interrupt line status. This bit is read-only.	0
5	USB_OTG_INT	OTG interrupt line status. This bit is read-only.	0

**Table 324. USB Interrupt Status register - (USBIntSt - address 0x2008 C1C0) bit description**

Bit	Symbol	Description	Reset Value
6	USB_I2C_INT	I <sup>2</sup> C module interrupt line status. This bit is read-only.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	USB_NEED_CLK	USB need clock indicator. This bit is read-only.	1
30:9	-	Reserved. Read value is undefined, only zero should be written.	NA
31	EN_USB_INTS	Enable all USB interrupts. When this bit is cleared, the NVIC does not see the ORed output of the USB interrupt lines.	1

### 15.8.2 OTG Interrupt Status Register (OTGIntSt - 0x2008 C100)

Bits in this register are set by hardware when the interrupt event occurs during the HNP handoff sequence. See [Section 15.9](#) for more information on when these bits are set.

**Table 325. OTG Interrupt Status register (OTGIntSt - address 0x2008 C100) bit description**

Bit	Symbol	Description	Reset Value
0	TMR	Timer time-out.	0
1	REMOVE_PU	Remove pull-up. This bit is set by hardware to indicate that software needs to disable the D+ pull-up resistor.	0
2	HNP_FAILURE	HNP failed. This bit is set by hardware to indicate that the HNP switching has failed.	0
3	HNP_SUCCESS	HNP succeeded. This bit is set by hardware to indicate that the HNP switching has succeeded.	0
31:4	-	Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.3 OTG Interrupt Enable Register (OTGIntEn - 0x2008 C104)

Writing a one to a bit in this register enables the corresponding bit in OTGIntSt to generate an interrupt on one of the interrupt lines. The interrupt is routed to the USB\_OTG\_INT interrupt line in the USBIntSt register.

The bit allocation and reset value of OTGIntEn is the same as OTGIntSt.

### 15.8.4 OTG Interrupt Set Register (OTGIntSet - 0x2008 C20C)

Writing a one to a bit in this register will set the corresponding bit in the OTGIntSt register. Writing a zero has no effect. The bit allocation of OTGIntSet is the same as in OTGIntSt.

### 15.8.5 OTG Interrupt Clear Register (OTGIntClr - 0x2008 C10C)

Writing a one to a bit in this register will clear the corresponding bit in the OTGIntSt register. Writing a zero has no effect. The bit allocation of OTGIntClr is the same as in OTGIntSt.

### 15.8.6 OTG Status and Control Register (OTGStCtrl - 0x2008 C110)

The OTGStCtrl register allows enabling hardware tracking during the HNP hand over sequence, controlling the OTG timer, monitoring the timer count, and controlling the functions mapped to port U1 and U2.

Time critical events during the switching sequence are controlled by the OTG timer. The timer can operate in two modes:

1. Monoshot mode: an interrupt is generated at the end of TIMEOUT\_CNT (see [Section 15.8.7 “OTG Timer Register \(OTGTmr - 0x2008 C114\)”](#)), the TMR bit is set in OTGIntSt, and the timer will be disabled.
2. Free running mode: an interrupt is generated at the end of TIMEOUT\_CNT (see [Section 15.8.7 “OTG Timer Register \(OTGTmr - 0x2008 C114\)”](#)), the TMR bit is set, and the timer value is reloaded into the counter. The timer is not disabled in this mode.

**Table 326. OTG Status Control register (OTGStCtrl - address 0x2008 C110) bit description**

Bit	Symbol	Description	Reset Value
1:0	PORT_FUNC	Controls connection of USB functions (see <a href="#">Figure 51</a> ). Bit 0 is set or cleared by hardware when B_HNP_TRACK or A_HNP_TRACK is set and HNP succeeds. See <a href="#">Section 15.9</a> . 00: U1 = device (OTG), U2 = host 01: U1 = host (OTG), U2 = host 10: Reserved 11: U1 = host, U2 = device	-
3:2	TMR_SCALE	Timer scale selection. This field determines the duration of each timer count. 00: 10 μs (100 KHz) 01: 100 μs (10 KHz) 10: 1000 μs (1 KHz) 11: Reserved	0
4	TMR_MODE	Timer mode selection. 0: monoshot 1: free running	0
5	TMR_EN	Timer enable. When set, TMR_CNT increments. When cleared, TMR_CNT is reset to 0.	0
6	TMR_RST	Timer reset. Writing one to this bit resets TMR_CNT to 0. This provides a single bit control for the software to restart the timer when the timer is enabled.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	B_HNP_TRACK	Enable HNP tracking for B-device (peripheral), see <a href="#">Section 15.9</a> . Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set.	0
9	A_HNP_TRACK	Enable HNP tracking for A-device (host), see <a href="#">Section 15.9</a> . Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set.	0
10	PU_REMOVED	When the B-device changes its role from peripheral to host, software sets this bit when it removes the D+ pull-up, see <a href="#">Section 15.9</a> . Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set.	0
15:11	-	Reserved. Read value is undefined, only zero should be written.	NA
31:16	TMR_CNT	Current timer count value.	0



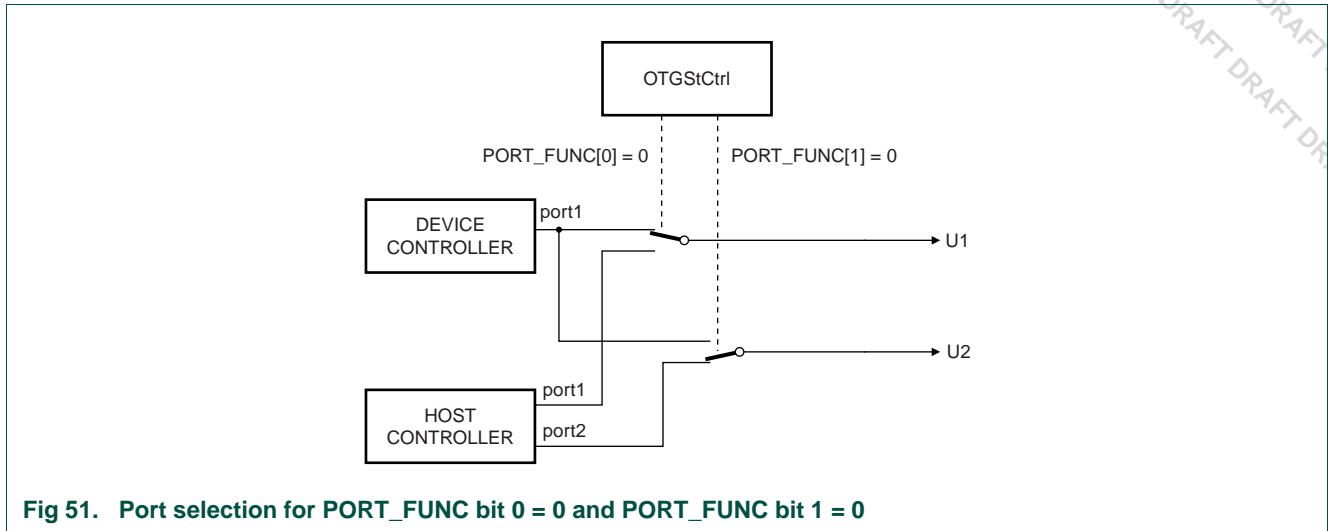


Fig 51. Port selection for PORT\_FUNC bit 0 = 0 and PORT\_FUNC bit 1 = 0

### 15.8.7 OTG Timer Register (OTGTmr - 0x2008 C114)

Table 327. OTG Timer register (OTGTmr - address 0x2008 C114) bit description

Bit	Symbol	Description	Reset Value
15:0	TIMEOUT_CNT	The TMR interrupt is set when TMR_CNT reaches this value.	0xFFFF
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.8 OTG Clock Control Register (OTGClkCtrl - 0x2008 CFF4)

This register controls the clocking of the OTG controller. Whenever software wants to access the registers, the corresponding clock control bit needs to be set. The software does not have to repeat this exercise for every register access, provided that the corresponding OTGClkCtrl bits are already set.

Table 328. OTG clock control register (OTGClkCtrl - address 0x2008 CFF4) bit description

Bit	Symbol	Value	Description	Reset Value
0	HOST_CLK_EN		Host clock enable	0
		0	Disable the Host clock.	
		1	Enable the Host clock.	
1	DEV_CLK_EN		Device clock enable	0
		0	Disable the Device clock.	
		1	Enable the Device clock.	
2	I2C_CLK_EN		I <sup>2</sup> C clock enable	0
		0	Disable the I <sup>2</sup> C clock.	
		1	Enable the I <sup>2</sup> C clock.	
3	OTG_CLK_EN		OTG clock enable	0
		0	Disable the OTG clock.	
		1	Enable the OTG clock.	

**Table 328. OTG clock control register (OTGCkCtrl - address 0x2008 CFF4) bit description**

Bit	Symbol	Value	Description	Reset Value
4	AHB_CLK_EN		AHB master clock enable	0
		0	Disable the AHB clock.	
		1	Enable the AHB clock.	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.9 OTG Clock Status Register (OTGCkSt - 0x2008 CFF8)

This register holds the clock availability status. When enabling a clock via OTGCkCtrl, software should poll the corresponding bit in this register. If it is set, then software can go ahead with the register access. Software does not have to repeat this exercise for every access, provided that the OTGCkCtrl bits are not disturbed.

**Table 329. OTG clock status register (OTGCkSt - address 0x2008 CFF8) bit description**

Bit	Symbol	Value	Description	Reset Value
0	HOST_CLK_ON		Host clock status.	0
		0	Host clock is not available.	
		1	Host clock is available.	
1	DEV_CLK_ON		Device clock status.	0
		0	Device clock is not available.	
		1	Device clock is available.	
2	I2C_CLK_ON		I <sup>2</sup> C clock status.	0
		0	I <sup>2</sup> C clock is not available.	
		1	I <sup>2</sup> C clock is available.	
3	OTG_CLK_ON		OTG clock status.	0
		0	OTG clock is not available.	
		1	OTG clock is available.	
4	AHB_CLK_ON		AHB master clock status.	0
		0	AHB clock is not available.	
		1	AHB clock is available.	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.10 I<sup>2</sup>C Receive Register (I2C\_RX - 0x2008 C300)

This register is the top byte of the receive FIFO. The receive FIFO is 4 bytes deep. The Rx FIFO is flushed by a hard reset or by a soft reset (I2C\_CTL bit 7). Reading an empty FIFO gives unpredictable data results.

**Table 330. I<sup>2</sup>C Receive register (I2C\_RX - address 0x2008 C300) bit description**

Bit	Symbol	Description	Reset Value
7:0	RX Data	Receive data.	-

### 15.8.11 I<sup>2</sup>C Transmit Register (I2C\_TX - 0x2008 C300)

This register is the top byte of the transmit FIFO. The transmit FIFO is 4 bytes deep.

The Tx FIFO is flushed by a hard reset, soft reset (I2C\_CTL bit 7) or if an arbitration failure occurs (I2C\_STS bit 3). Data writes to a full FIFO are ignored.

I2C\_TX must be written for both write and read operations to transfer each byte. Bits [7:0] are ignored for master-receive operations. The master-receiver must write a dummy byte to the TX FIFO for each byte it expects to receive in the RX FIFO. When the STOP bit is set or the START bit is set to cause a RESTART condition on a byte written to the TX FIFO (master-receiver), then the byte read from the slave is not acknowledged. That is, the last byte of a master-receive operation is not acknowledged.

**Table 331. I<sup>2</sup>C Transmit register (I2C\_TX - address 0x2008 C300) bit description**

Bit	Symbol	Description	Reset Value
7:0	TX Data	Transmit data.	-
8	START	When 1, issue a START condition before transmitting this byte.	-
9	STOP	When 1, issue a STOP condition after transmitting this byte.	-
31:10	-	Reserved. Read value is undefined, only zero should be written.	-

### 15.8.12 I<sup>2</sup>C Status Register (I2C\_STS - 0x2008 C304)

The I2C\_STS register provides status information on the TX and RX blocks as well as the current state of the external buses. Individual bits are enabled as interrupts by the I2C\_CTL register and routed to the I2C\_USB\_INT bit in USBIntSt.

**Table 332. I<sup>2</sup>C status register (I2C\_STS - address 0x2008 C304) bit description**

Bit	Symbol	Value	Description	Reset Value
0	TDI		Transaction Done Interrupt. This flag is set if a transaction completes successfully. It is cleared by writing a one to bit 0 of the status register. It is unaffected by slave transactions.	0
		0	Transaction has not completed.	
		1	Transaction completed.	
1	AFI		Arbitration Failure Interrupt. When transmitting, if the SDA is low when SDAOUT is high, then this I <sup>2</sup> C has lost the arbitration to another device on the bus. The Arbitration Failure bit is set when this happens. It is cleared by writing a one to bit 1 of the status register.	0
		0	No arbitration failure on last transmission.	
		1	Arbitration failure occurred on last transmission.	
2	NAI		No Acknowledge Interrupt. After every byte of data is sent, the transmitter expects an acknowledge from the receiver. This bit is set if the acknowledge is not received. It is cleared when a byte is written to the master TX FIFO.	0
		0	Last transmission received an acknowledge.	
		1	Last transmission did not receive an acknowledge.	
3	DRMI		Master Data Request Interrupt. Once a transmission is started, the transmitter must have data to transmit as long as it isn't followed by a stop condition or it will hold SCL low until more data is available. The Master Data Request bit is set when the master transmitter is data-starved. If the master TX FIFO is empty and the last byte did not have a STOP condition flag, then SCL is held low until the CPU writes another byte to transmit. This bit is cleared when a byte is written to the master TX FIFO.	0
		0	Master transmitter does not need data.	
		1	Master transmitter needs data.	

Table 332. I<sup>2</sup>C status register (I2C\_STS - address 0x2008 C304) bit description

Bit	Symbol	Value	Description	Reset Value
4	DRSI		Slave Data Request Interrupt. Once a transmission is started, the transmitter must have data to transmit as long as it isn't followed by a STOP condition or it will hold SCL low until more data is available. The Slave Data Request bit is set when the slave transmitter is data-starved. If the slave TX FIFO is empty and the last byte transmitted was acknowledged, then SCL is held low until the CPU writes another byte to transmit. This bit is cleared when a byte is written to the slave Tx FIFO.	0
		0	Slave transmitter does not need data.	
		1	Slave transmitter needs data.	
5	Active		Indicates whether the bus is busy. This bit is set when a START condition has been seen. It is cleared when a STOP condition is seen..	0
6	SCL		The current value of the SCL signal.	-
7	SDA		The current value of the SDA signal.	-
8	RFF		Receive FIFO Full (RFF). This bit is set when the RX FIFO is full and cannot accept any more data. It is cleared when the RX FIFO is not full. If a byte arrives when the Receive FIFO is full, the SCL is held low until the CPU reads the RX FIFO and makes room for it.	0
		0	RX FIFO is not full	
		1	RX FIFO is full	
9	RFE		Receive FIFO Empty. RFE is set when the RX FIFO is empty and is cleared when the RX FIFO contains valid data.	1
		0	RX FIFO contains data.	
		1	RX FIFO is empty	
10	TFF		Transmit FIFO Full. TFF is set when the TX FIFO is full and is cleared when the TX FIFO is not full.	0
		0	TX FIFO is not full.	
		1	TX FIFO is full	
11	TFE		Transmit FIFO Empty. TFE is set when the TX FIFO is empty and is cleared when the TX FIFO contains valid data.	1
		0	TX FIFO contains valid data.	
		1	TX FIFO is empty	
31:12	-		Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.13 I<sup>2</sup>C Control Register (I2C\_CTL - 0x2008 C308)

The I2C\_CTL register is used to enable interrupts and reset the I<sup>2</sup>C state machine. Enabled interrupts cause the USB\_I2C\_INT interrupt output line to be asserted when set.

Table 333. I<sup>2</sup>C Control register (I2C\_CTL - address 0x2008 C308) bit description

Bit	Symbol	Value	Description	Reset Value
0	TDIE		Transmit Done Interrupt Enable. This enables the TDI interrupt signalling that this I <sup>2</sup> C issued a STOP condition.	0
		0	Disable the TDI interrupt.	
		1	Enable the TDI interrupt.	

Table 333. I<sup>2</sup>C Control register (I2C\_CTL - address 0x2008 C308) bit description

Bit	Symbol	Value	Description	Reset Value
1	AFIE		Transmitter Arbitration Failure Interrupt Enable. This enables the AFI interrupt which is asserted during transmission when trying to set SDA high, but the bus is driven low by another device.	0
		0	Disable the AFI.	
		1	Enable the AFI.	
2	NAIE		Transmitter No Acknowledge Interrupt Enable. This enables the NAI interrupt signalling that transmitted byte was not acknowledged.	0
		0	Disable the NAI.	
		1	Enable the NAI.	
3	DRMIE		Master Transmitter Data Request Interrupt Enable. This enables the DRMI interrupt which signals that the master transmitter has run out of data, has not issued a STOP, and is holding the SCL line low.	0
		0	Disable the DRMI interrupt.	
		1	Enable the DRMI interrupt.	
4	DRSIE		Slave Transmitter Data Request Interrupt Enable. This enables the DRSI interrupt which signals that the slave transmitter has run out of data and the last byte was acknowledged, so the SCL line is being held low.	0
		0	Disable the DRSI interrupt.	
		1	Enable the DRSI interrupt.	
5	REFIE		Receive FIFO Full Interrupt Enable. This enables the Receive FIFO Full interrupt to indicate that the receive FIFO cannot accept any more data.	0
		0	Disable the RFFI.	
		1	Enable the RFFI.	
6	RFDAIE		Receive Data Available Interrupt Enable. This enables the DAI interrupt to indicate that data is available in the receive FIFO (i.e. not empty).	0
		0	Disable the DAI.	
		1	Enable the DAI.	
7	TFFIE		Transmit FIFO Not Full Interrupt Enable. This enables the Transmit FIFO Not Full interrupt to indicate that the more data can be written to the transmit FIFO. Note that this is not full. It is intended help the CPU to write to the I <sup>2</sup> C block only when there is room in the FIFO and do this without polling the status register.	0
		0	Disable the TFFI.	
		1	Enable the TFFI.	
8	SRST		Soft reset. This is only needed in unusual circumstances. If a device issues a start condition without issuing a stop condition. A system timer may be used to reset the I <sup>2</sup> C if the bus remains busy longer than the time-out period. On a soft reset, the Tx and Rx FIFOs are flushed, I2C_STS register is cleared, and all internal state machines are reset to appear idle. The I2C_CLKHI, I2C_CLKLO and I2C_CTL (except Soft Reset Bit) are NOT modified by a soft reset.	0
		0	See the text.	
		1	Reset the I <sup>2</sup> C to idle state. Self clearing.	
31:9	-		Reserved. Read value is undefined, only zero should be written.	NA

### 15.8.14 I<sup>2</sup>C Clock High Register (I2C\_CLKHI - 0x2008 C30C)

The CLK register holds a terminal count for counting 48 MHz clock cycles to create the high period of the slower I<sup>2</sup>C serial clock, SCL.

Table 334. I<sup>2</sup>C\_CLKHI register (I2C\_CLKHI - address 0x2008 C30C) bit description

Bit	Symbol	Description	Reset Value
7:0	CDHI	Clock divisor high. This value is the number of 48 MHz clocks the serial clock (SCL) will be high.	0xB9

### 15.8.15 I<sup>2</sup>C Clock Low Register (I2C\_CLKLO - 0x2008 C310)

The CLK register holds a terminal count for counting 48 MHz clock cycles to create the low period of the slower I<sup>2</sup>C serial clock, SCL.

Table 335. I<sup>2</sup>C\_CLKLO register (I2C\_CLKLO - address 0x2008 C310) bit description

Bit	Symbol	Description	Reset Value
7:0	CDLO	Clock divisor low. This value is the number of 48 MHz clocks the serial clock (SCL) will be low.	0xB9

### 15.8.16 Interrupt handling

The interrupts set in the OTGIntSt register are set and cleared during HNP switching. All OTG related interrupts, if enabled, are routed to the USB\_OTG\_INT bit in the USBIntSt register.

I<sup>2</sup>C related interrupts are set in the I2C\_STS register and routed, if enabled by I2C\_CTL, to the USB\_I2C\_INT bit.

For more details on the interrupts created by device controller, see the USB device chapter. For interrupts created by the host controllers, see the OHCI specification.

The EN\_USB\_INTS bit in the USBIntSt register enables the routing of any of the USB related interrupts to the NVIC controller (see [Figure 52](#)).

**Remark:** During the HNP switching between host and device with the OTG stack active, an action may raise several levels of interrupts. It is advised to let the OTG stack initiate any actions based on interrupts and ignore device and host level interrupts. This means that during HNP switching, the OTG stack provides the communication to the host and device controllers.

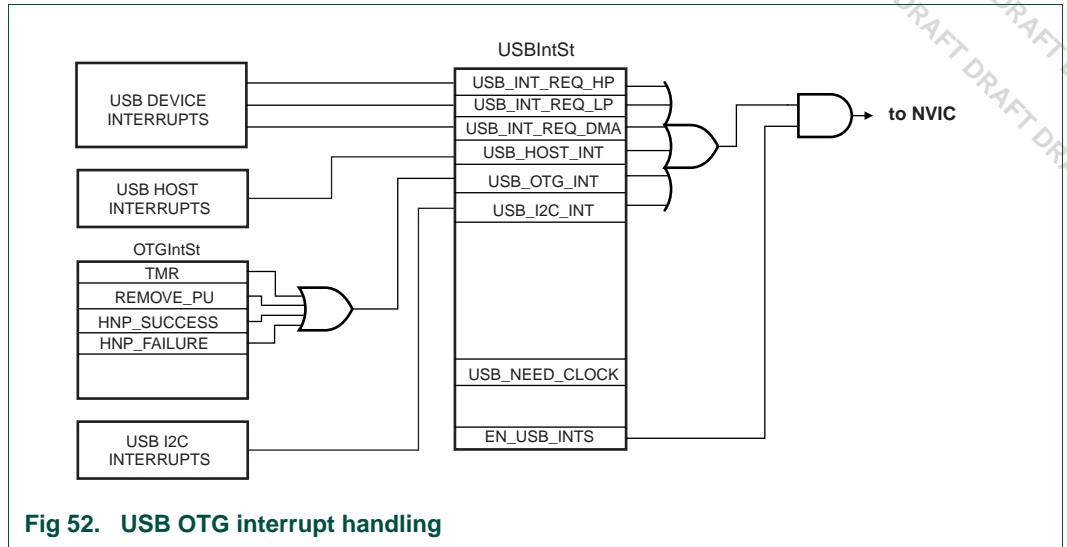


Fig 52. USB OTG interrupt handling

## 15.9 HNP support

This section describes the hardware support for the Host Negotiation Protocol (HNP) provided by the OTG controller.

When two dual-role OTG devices are connected to each other, the plug inserted into the mini-AB receptacle determines the default role of each device. The device with the mini-A plug inserted becomes the default Host (A-device), and the device with the mini-B plug inserted becomes the default Peripheral (B-device).

Once connected, the default Host (A-device) and the default Peripheral (B-device) can switch Host and Peripheral roles using HNP.

The context of the OTG controller operation is shown in [Figure 53](#). Each controller (Host, Device, or OTG) communicates with its software stack through a set of status and control registers and interrupts. In addition, the OTG software stack communicates with the external OTG transceiver through the I<sup>2</sup>C interface and the external transceiver interrupt signal.

The OTG software stack is responsible for implementing the HNP state machines as described in the On-The-Go Supplement to the USB 2.0 Specification.

The OTG controller hardware provides support for some of the state transitions in the HNP state machines as described in the following subsections.

The USB state machines, the HNP switching, and the communications between the USB controllers are described in more detail in the following documentation:

- USB OHCI specification
- USB OTG supplement, version 1.2
- USB 2.0 specification
- ISP1302 data sheet and user manual



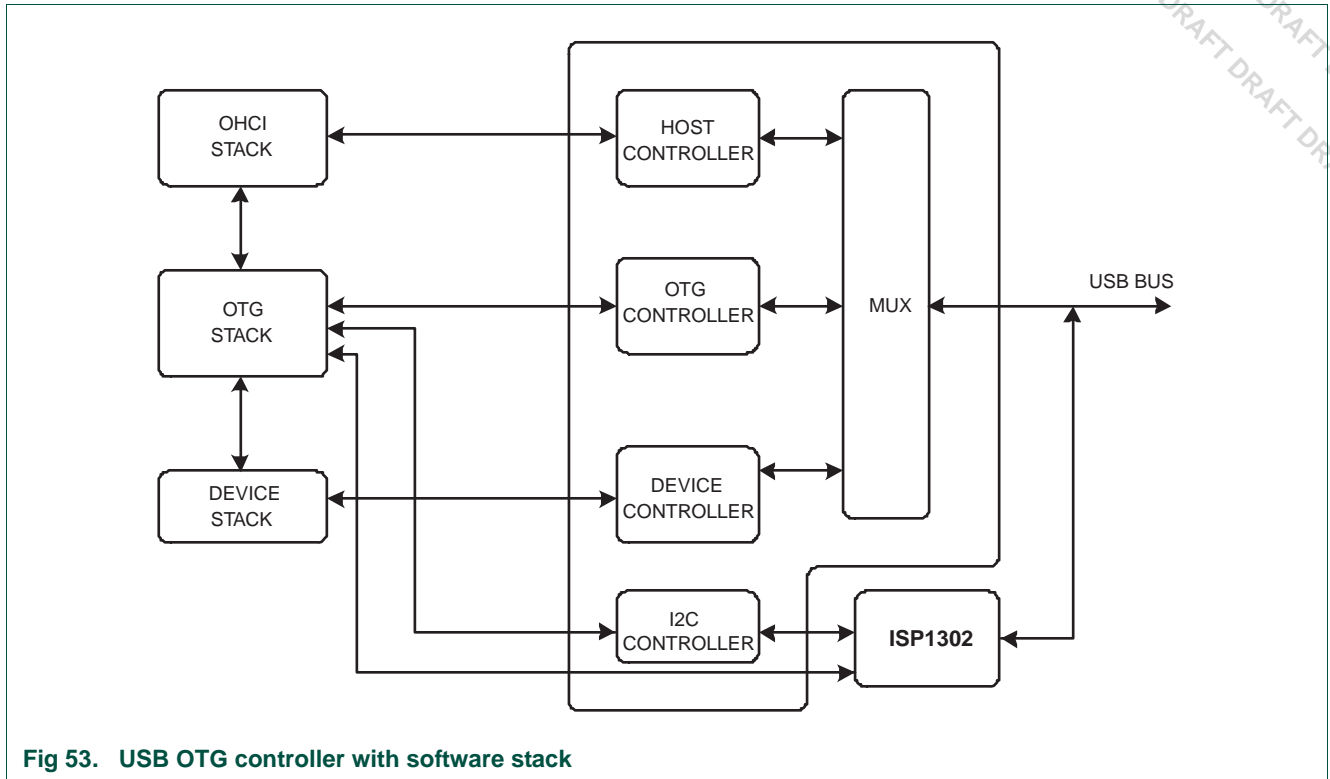


Fig 53. USB OTG controller with software stack

### 15.9.1 B-device: peripheral to host switching

In this case, the default role of the OTG controller is peripheral (B-device), and it switches roles from Peripheral to Host.

The On-The-Go Supplement defines the behavior of a dual-role B-device during HNP using a state machine diagram. The OTG software stack is responsible for implementing all of the states in the Dual-Role B-Device State Diagram.

The OTG controller hardware provides support for the state transitions between the states `b_peripheral`, `b_wait_acon`, and `b_host` in the Dual-Role B-Device state diagram. Setting `B_HNP_TRACK` in the `OTGStCtrl` register enables hardware support for the B-device switching from peripheral to host. The hardware actions after setting this bit are shown in [Figure 54](#).

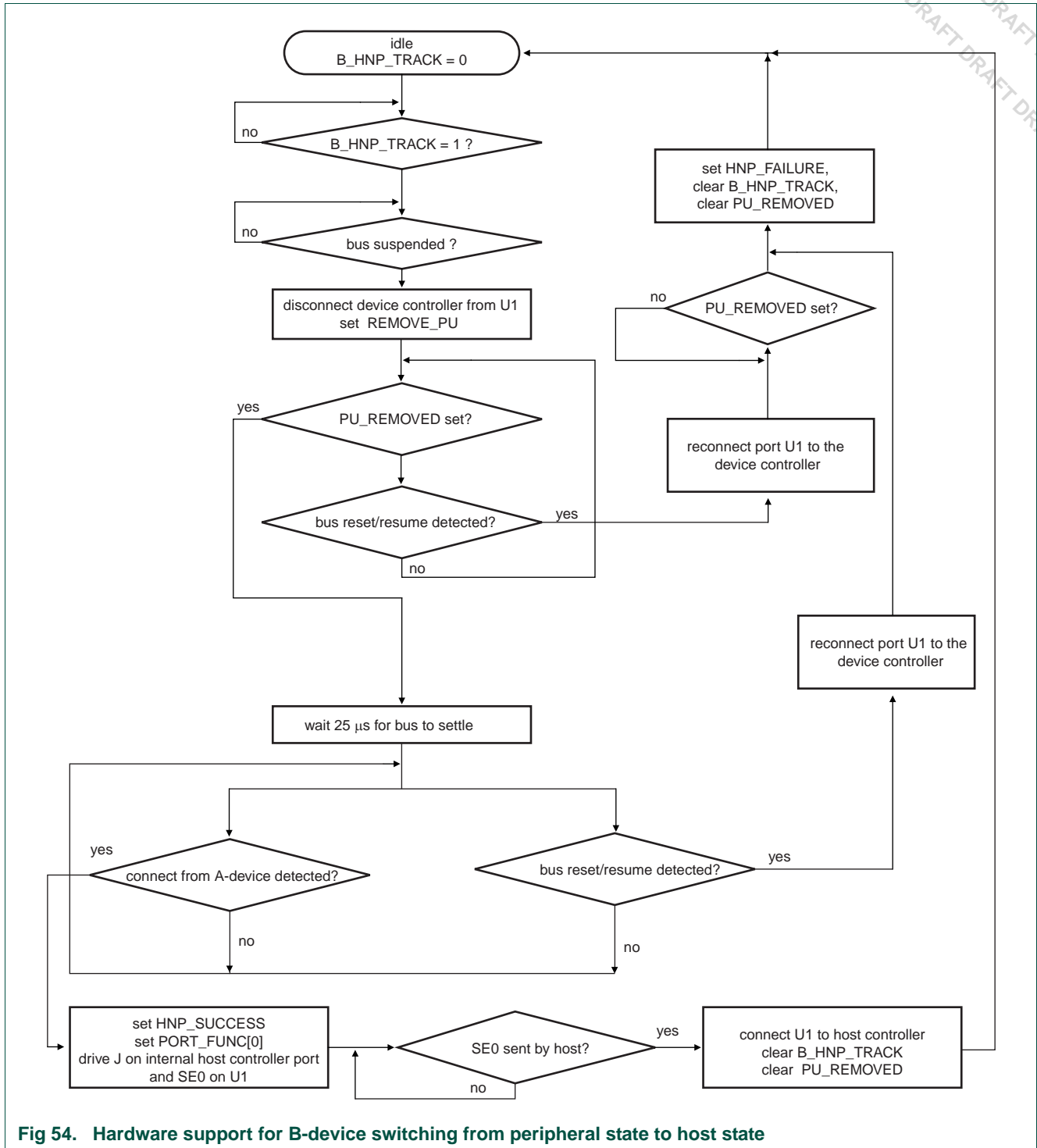


Fig 54. Hardware support for B-device switching from peripheral state to host state

Figure 55 shows the actions that the OTG software stack should take in response to the hardware actions setting REMOVE\_PU, HNP\_SUCCESS, AND HNP\_FAILURE. The relationship of the software actions to the Dual-Role B-Device states is also shown. B-device states are in bold font with a circle around them.

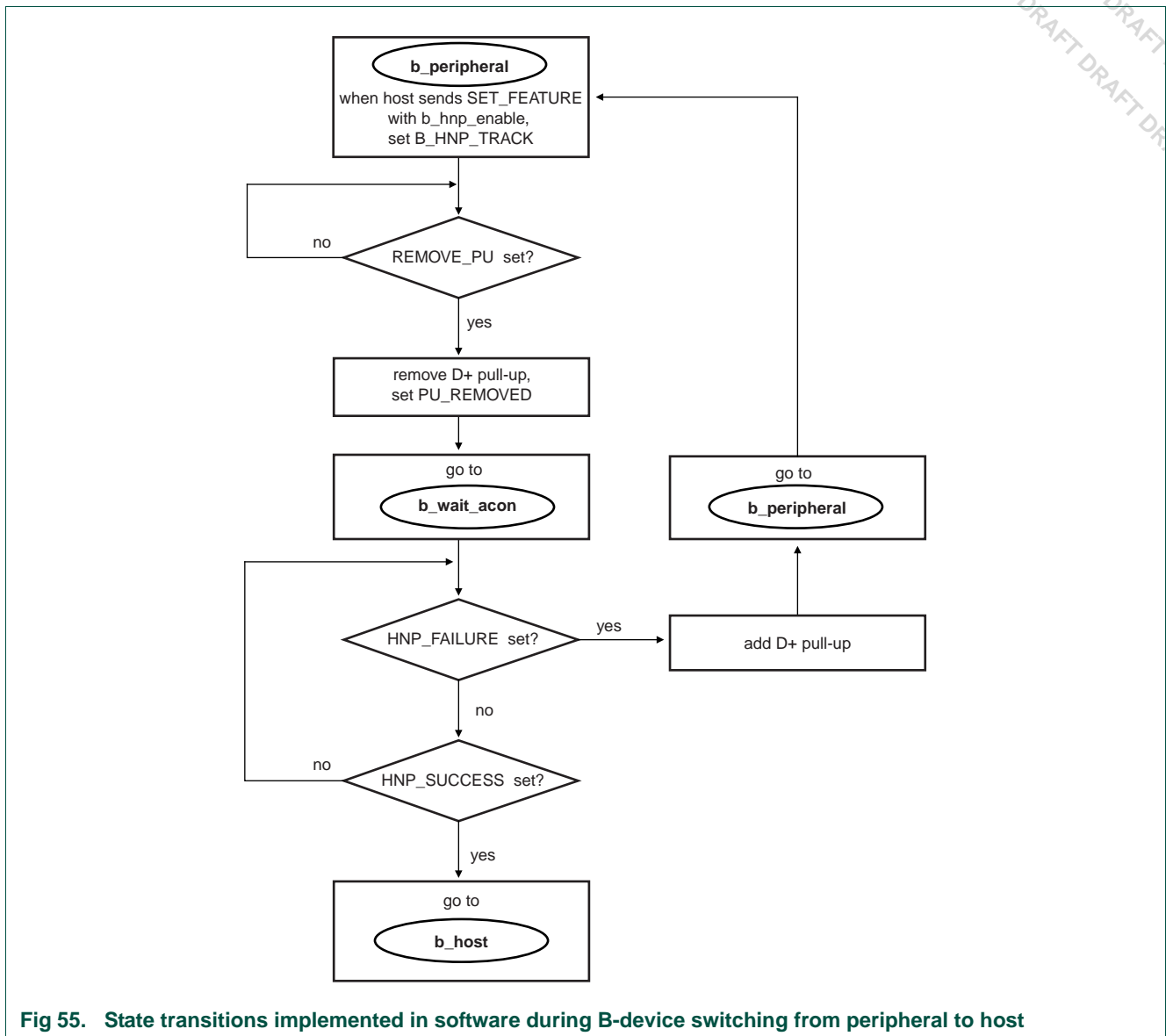


Fig 55. State transitions implemented in software during B-device switching from peripheral to host

Note that only the subset of B-device HNP states and state transitions supported by hardware are shown. Software is responsible for implementing all of the HNP states.

Figure 55 may appear to imply that the interrupt bits such as REMOVE\_PU should be polled, but this is not necessary if the corresponding interrupt is enabled.

Following are code examples that show how the actions in Figure 55 are accomplished. The examples assume that ISP1302 is being used as the external OTG transceiver.

**Remove D+ pull-up**

```

/* Remove D+ pull-up through ISP1302 */
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0
OTG_I2C_TX = 0x007; // Send OTG Control (Clear) register address
OTG_I2C_TX = 0x201; // Clear DP_PULLUP bit, send STOP condition
  
```

```
/* Wait for TDI to be set */  
while (!(OTG_I2C_STS & TDI));
```

```
/* Clear TDI */  
OTG_I2C_STS = TDI;
```

### Add D+ pull-up

```
/* Add D+ pull-up through ISP1302 */  
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0  
OTG_I2C_TX = 0x006; // Send OTG Control (Set) register address  
OTG_I2C_TX = 0x201; // Set DP_PULLUP bit, send STOP condition
```

```
/* Wait for TDI to be set */  
while (!(OTG_I2C_STS & TDI));
```

```
/* Clear TDI */  
OTG_I2C_STS = TDI;
```

## 15.9.2 A-device: host to peripheral HNP switching

In this case, the role of the OTG controller is host (A-device), and the A-device switches roles from host to peripheral.

The On-The-Go Supplement defines the behavior of a dual-role A-device during HNP using a state machine diagram. The OTG software stack is responsible for implementing all of the states in the Dual-Role A-Device State Diagram.

The OTG controller hardware provides support for the state transitions between a\_host, a\_suspend, a\_wait\_vfall, and a\_peripheral in the Dual-Role A-Device state diagram. Setting A\_HNP\_TRACK in the OTGStCtrl register enables hardware support for switching the A-device from the host state to the device state. The hardware actions after setting this bit are shown in [Figure 56](#).

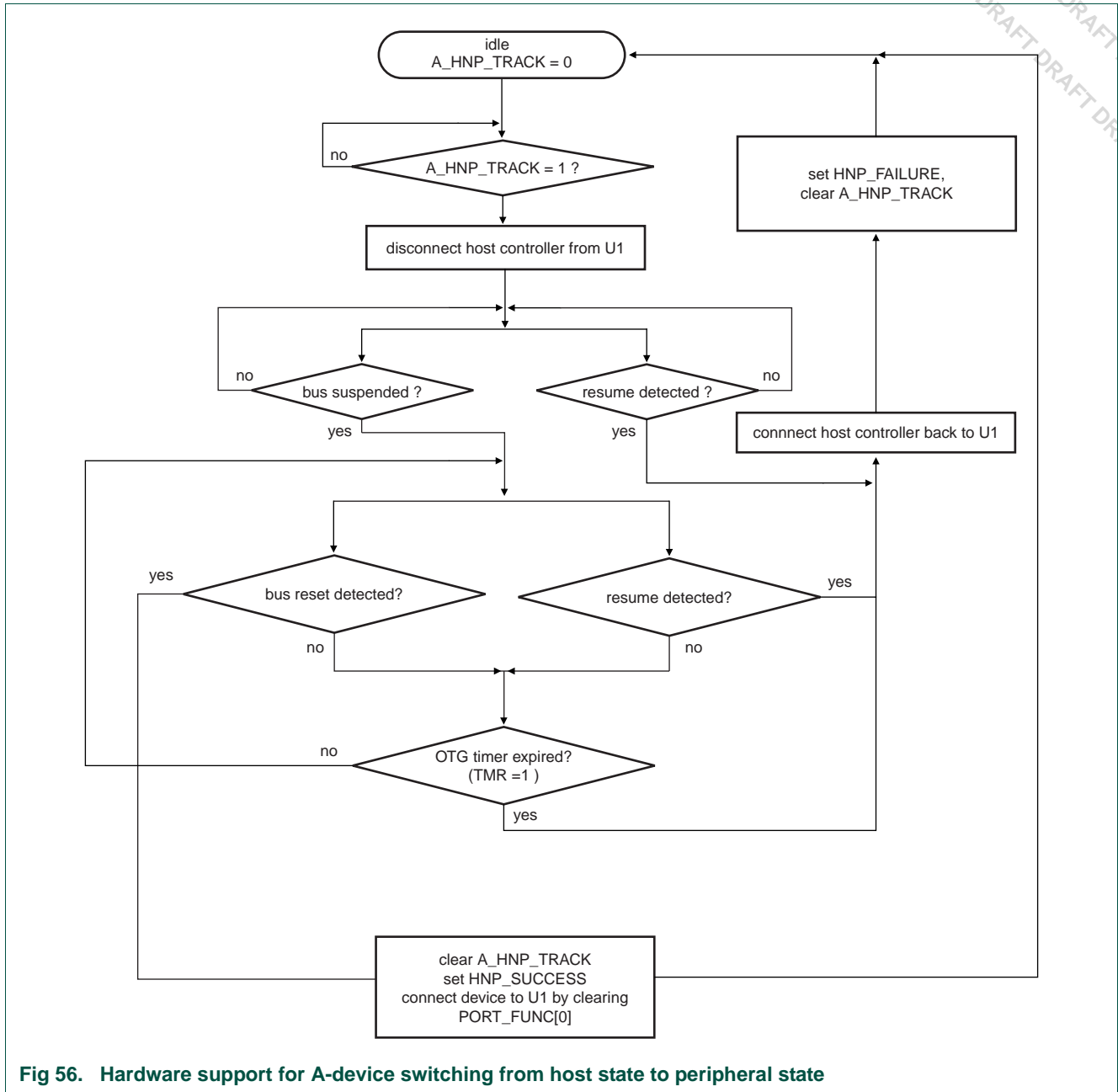


Fig 56. Hardware support for A-device switching from host state to peripheral state

Figure 57 shows the actions that the OTG software stack should take in response to the hardware actions setting TMR, HNP\_SUCCESS, and HNP\_FAILURE. The relationship of the software actions to the Dual-Role A-Device states is also shown. A-device states are shown in bold font with a circle around them.

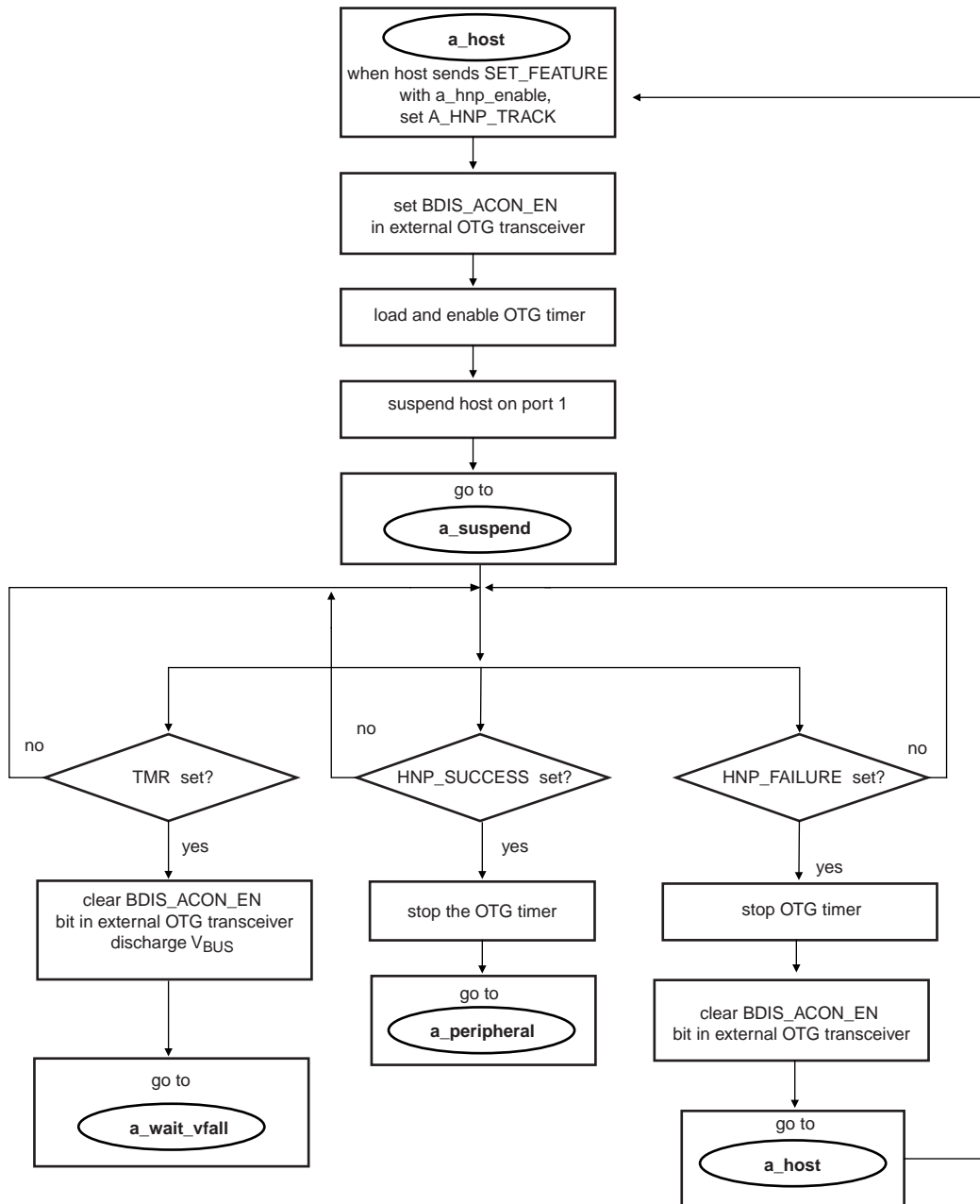


Fig 57. State transitions implemented in software during A-device switching from host to peripheral

Note that only the subset of A-device HNP states and state transitions supported by hardware are shown. Software is responsible for implementing all of the HNP states.

Figure 57 may appear to imply that the interrupt bits such as TMR should be polled, but this is not necessary if the corresponding interrupt is enabled.

Following are code examples that show how the actions in Figure 57 are accomplished. The examples assume that ISP1302 is being used as the external OTG transceiver.

**Set BDIS\_ACON\_EN in external OTG transceiver**

```

/* Set BDIS_ACON_EN in ISP1302 */
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0
OTG_I2C_TX = 0x004; // Send Mode Control 1 (Set) register address
OTG_I2C_TX = 0x210; // Set BDIS_ACON_EN bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;

```

#### Clear BDIS\_ACON\_EN in external OTG transceiver

```

/* Set BDIS_ACON_EN in ISP1302 */
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0
OTG_I2C_TX = 0x005; // Send Mode Control 1 (Clear) register address
OTG_I2C_TX = 0x210; // Clear BDIS_ACON_EN bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;

```

#### Discharge V<sub>BUS</sub>

```

/* Clear the VBUS_DRV bit in ISP1302 */
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0
OTG_I2C_TX = 0x007; // Send OTG Control (Clear) register address
OTG_I2C_TX = 0x220; // Clear VBUS_DRV bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;

/* Set the VBUS_DISCHRG bit in ISP1302 */
OTG_I2C_TX = 0x15A; // Send ISP1302 address, R/W=0
OTG_I2C_TX = 0x006; // Send OTG Control (Set) register address
OTG_I2C_TX = 0x240; // Set VBUS_DISCHRG bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;

```

### Load and enable OTG timer

```
/* The following assumes that the OTG timer has previously been */
/* configured for a time scale of 1 ms (TMR_SCALE = "10") */
/* and monoshot mode (TMR_MODE = 0) */
```

```
/* Load the timeout value to implement the a_aidl_bdis_tmr timer */
/* the minimum value is 200 ms */
OTG_TIMER = 200;
```

```
/* Enable the timer */
OTG_STAT_CTRL |= TMR_EN;
```

### Stop OTG timer

```
/* Disable the timer - causes TMR_CNT to be reset to 0 */
OTG_STAT_CTRL &= ~TMR_EN;
```

```
/* Clear TMR interrupt */
OTG_INT_CLR = TMR;
```

### Suspend host on port 1

```
/* Write to PortSuspendStatus bit to suspend host port 1 - */
/* this example demonstrates the low-level action software needs to take. */
/* The host stack code where this is done will be somewhat more involved. */
HC_RH_PORT_STAT1 = PSS;
```



## 15.10 Clocking and power management

The OTG controller clocking is shown in [Figure 58](#).

A clock switch controls each clock with the exception of `ahb_slave_clk`. When the enable of the clock switch is asserted, its clock output is turned on and its `CLK_ON` output is asserted. The `CLK_ON` signals are observable in the `OTGCkSt` register.

To conserve power, the clocks to the Device, Host, OTG, and I<sup>2</sup>C controllers can be disabled when not in use by clearing the respective `CLK_EN` bit in the `OTGCkCtrl` register. When the entire USB block is not in use, all of its clocks can be disabled by clearing the `PCUSB` bit in the `PCONP` register.

When software wishes to access registers in one of the controllers, it should first ensure that the respective controller's 48 MHz clock is enabled by setting its `CLK_EN` bit in the `OTGCkCtrl` register and then poll the corresponding `CLK_ON` bit in `OTGCkSt` until set. Once set, the controller's clock will remain enabled until `CLK_EN` is cleared by software. Accessing the register of a controller when its 48 MHz clock is not enabled will result in a data abort exception.

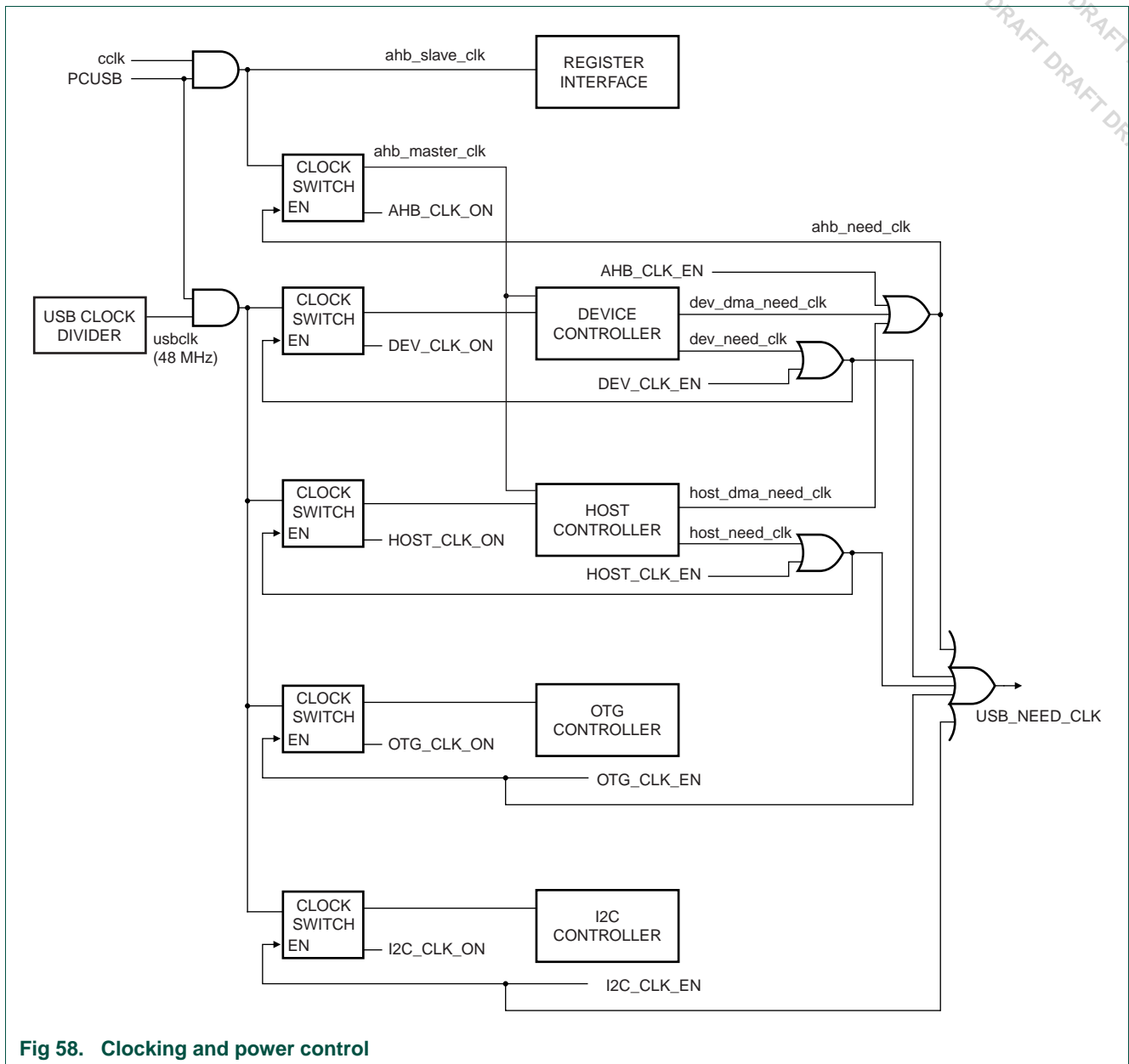


Fig 58. Clocking and power control

### 15.10.1 Device clock request signals

The Device controller has two clock request signals, `dev_need_clk` and `dev_dma_need_clk`. When asserted, these signals turn on the device's 48 MHz clock and `ahb_master_clk` respectively.

The `dev_need_clk` signal is asserted while the device is not in the suspend state, or if the device is in the suspend state and activity is detected on the USB bus. The `dev_need_clk` signal is de-asserted if a disconnect is detected (CON bit is cleared in the SIE Get Device Status register – [Section 13.10.7](#)). This signal allows `DEV_CLK_EN` to be cleared during normal operation when software does not need to access the Device controller registers – the Device will continue to function normally and automatically shut off its clock when it is suspended or disconnected.

The `dev_dma_need_clk` signal is asserted on any Device controller DMA access to memory. Once asserted, it remains active for 2 ms (2 frames), to help assure that DMA throughput is not affected by any latency associated with re-enabling `ahb_master_clk`. 2 ms after the last DMA access, `dev_dma_need_clk` is de-asserted to help conserve power. This signal allows `AHB_CLK_EN` to be cleared during normal operation.

### 15.10.1.1 Host clock request signals

The Host controller has two clock request signals, `host_need_clk` and `host_dma_need_clk`. When asserted, these signals turn on the host's 48 MHz clock and `ahb_master_clk` respectively.

The `host_need_clk` signal is asserted while the Host controller functional state is not `UsbSuspend`, or if the functional state is `UsbSuspend` and resume signaling or a disconnect is detected on the USB bus. This signal allows `HOST_CLK_EN` to be cleared during normal operation when software does not need to access the Host controller registers – the Host will continue to function normally and automatically shut off its clock when it goes into the `UsbSuspend` state.

The `host_dma_need_clk` signal is asserted on any Host controller DMA access to memory. Once asserted, it remains active for 2 ms (2 frames), to help assure that DMA throughput is not affected by any latency associated with re-enabling `ahb_master_clk`. 2 ms after the last DMA access, `host_dma_need_clk` is de-asserted to help conserve power. This signal allows `AHB_CLK_EN` to be cleared during normal operation.

### 15.10.2 Power-down mode support

The LPC178x/177x can be configured to wake up from Power-down mode on any USB bus activity. When the chip is in Power-down mode and the USB interrupt is enabled, the assertion of `USB_NEED_CLK` causes the chip to wake up from Power-down mode.

Before Power-down mode can be entered when the USB activity interrupt is enabled, `USB_NEED_CLK` must be de-asserted. This is accomplished by clearing all of the `CLK_EN` bits in `OTGClkCtrl` and putting the Host controller into the `UsbSuspend` functional state. If it is necessary to wait for either of the `dma_need_clk` signals or the `dev_need_clk` to be de-asserted, the status of `USB_NEED_CLK` can be polled in the `USBIntSt` register to determine when they have all been de-asserted.

## 15.11 USB OTG controller initialization

The LPC178x/177x OTG device controller initialization includes the following steps:

1. Enable the device controller by setting the PCUSB bit of PCONP.
2. Configure and enable the Alt PLL (PLL1) or Main PLL (PLL0) to provide 48 MHz for usbclock and the desired frequency for cclk. For the procedure for determining the PLL setting and configuration, see [Section 4.5.10 “Procedure for determining PLL settings”](#) or [Section 4.5.10 “Procedure for determining PLL settings”](#).
3. Enable the desired controller clocks by setting their respective CLK\_EN bits in the USBClkCtrl register. Poll the corresponding CLK\_ON bits in the USBClkSt register until they are set.
4. Enable the desired USB pin functions by writing to the corresponding IOCON registers.
5. Follow the appropriate steps in [Section 13.13 “USB device controller initialization”](#) to initialize the device controller.
6. Follow the guidelines given in the OpenHCI specification for initializing the host controller.

### 16.1 Basic configuration

---

The SD card interface (also known as MCI or Multimedia card interface) is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PC\_SD.  
**Remark:** On reset, the SD card interface is disabled (PCSD = 0).
2. Peripheral clock: The SD card interface operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select SD card interface pins and their modes through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 16.2 Introduction

---

The Secure Digital card interface is an interface between the Advanced Peripheral Bus (APB) system bus and multimedia and/or secure digital memory cards. It consists of two parts:

- The SD card interface provides all functions specific to the Secure Digital memory card, such as the clock generation unit, power management control, command and data transfer. The interface also supports the Multimedia Card Interface.
- The APB interface accesses SD card interface registers, and generates interrupt and DMA request signals.

### 16.3 Features

---

The following features are provided by the SD card interface:

- Conformance to *Secure Digital Memory Card Physical Layer Specification, v0.96*.
- Conformance to *Multimedia Card Specification v2.11*.
- Use as a multimedia card bus or a secure digital memory card bus host. It can be connected to several multimedia cards, or a single secure digital memory card.
- DMA supported through the General Purpose DMA Controller.

## 16.4 Pin description

Table 336. SD/MMC card interface pin description

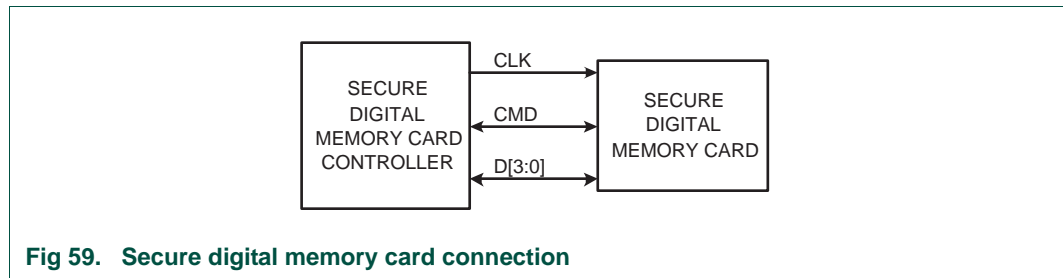
Pin Name	Type	Description
SD_CLK	Output	Clock output
SD_CMD	Input	Command input/output.
SD_DAT[3:0]	Output	Data lines. Only SD_DAT[0] is used for Multimedia cards.
SD_PWR	Output	Power Supply Enable for external card power supply.

## 16.5 Functional overview

The SD card interface may be used as a secure digital memory card bus host (see [Section 16.5.1 “Secure digital memory card”](#)) or as a multimedia card bus host (see [Section 16.5.2 “Multimedia card”](#)). A single secure digital memory card or up to 4 multimedia cards (depending on board loading) may be connected.

### 16.5.1 Secure digital memory card

[Figure 59](#) shows the secure digital memory card connection.



#### 16.5.1.1 Secure digital memory card bus signals

The following signals are used on the secure digital memory card bus:

- SD\_CLK Host to card clock signal
- SD\_CMD Bidirectional command/response signal
- SD\_DAT[3:0] Bidirectional data signals

### 16.5.2 Multimedia card

[Figure 60](#) shows the multimedia card system.

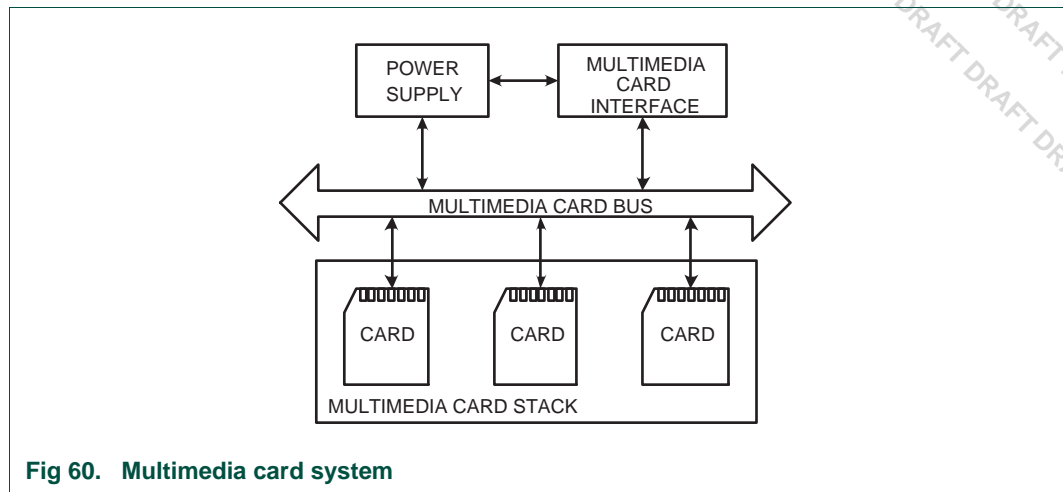


Fig 60. Multimedia card system

Multimedia cards are grouped into three types according to their function:

- Read Only Memory (ROM) cards, containing pre-programmed data
- Read/Write (R/W) cards, used for mass storage
- Input/Output (I/O) cards, used for communication

The multimedia card system transfers commands and data using three signal lines:

- CLK: One bit is transferred on both command and data lines with each clock cycle. The clock frequency varies between 0 MHz and 20 MHz (for a multimedia card) or 0 MHz and 25 MHz (for a secure digital memory card).
- CMD: Bidirectional command channel that initializes a card and transfers commands. CMD has two operational modes:
  - Open-drain for initialization
  - Push-pull for command transfer
- DAT: Bidirectional data channel, operating in push-pull mode

### 16.5.3 SD card interface details

[Figure 61](#) shows a simplified block diagram of the SD card interface.

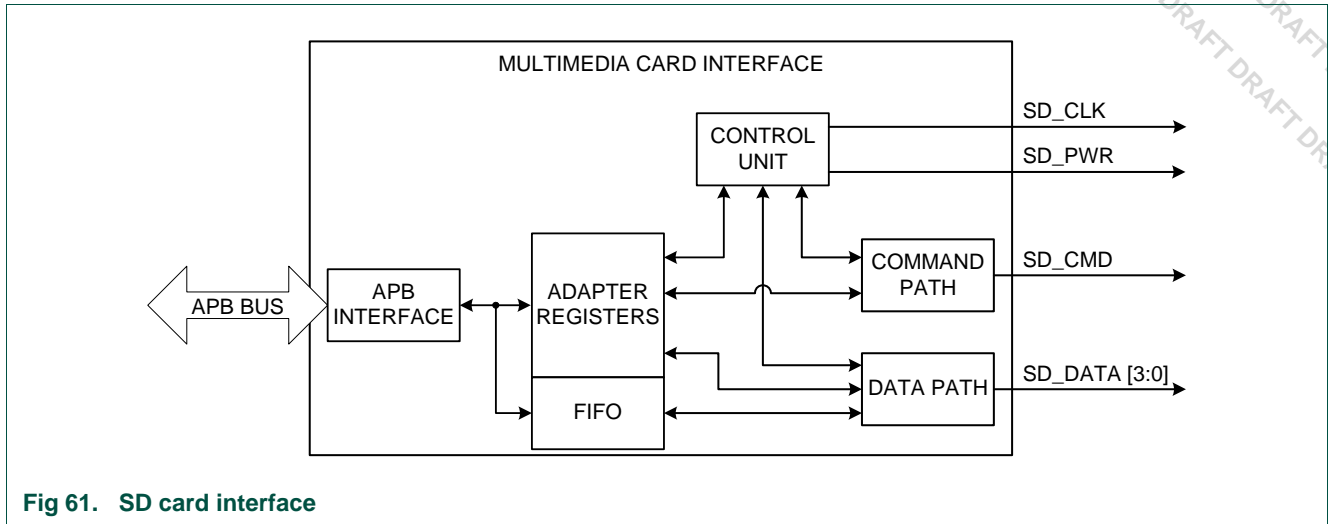


Fig 61. SD card interface

The SD card interface is a secure digital/multimedia memory card bus master that provides an interface to a multimedia card stack or to a secure digital memory card. It consists of five subunits:

- Adapter register block
- Control unit
- Command path
- Data path
- Data FIFO

### 16.5.3.1 Adapter register block

The adapter register block contains all system registers. This block also generates the signals that clear the static flags in the multimedia card. The clear signals are generated when 1 is written into the corresponding bit location of the MCIClear register.

### 16.5.3.2 Control unit

The control unit contains the power management functions and the clock divider for the memory card clock.

There are three power phases:

- Power-off
- Power-up
- Power-on

The power management logic controls an external power supply unit, and disables the card bus output signals during the power-off or power-up phases. The power-up phase is a transition phase between the power-off and power-on phases, and allows an external power supply to reach the card bus operating voltage. A device driver is used to ensure that the interface remains in the power-up phase until the external power supply reaches the operating voltage.



The clock management logic generates and controls the SD\_CLK signal. The SD\_CLK output can use either a clock divide or clock bypass mode. The clock output is inactive:

- after reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the IDLE state (eight clock periods after both the command and data path subunits enter the IDLE phase)

### 16.5.3.3 Command path

The command path subunit sends commands to and receives responses from the cards.

### 16.5.3.4 Command path state machine

When the command register is written to and the enable bit is set, command transfer starts. When the command has been sent, the Command Path State Machine (CPSM) sets the status flags and enters the IDLE state if a response is not required. If a response is required, it waits for the response (see [Figure 62](#)). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

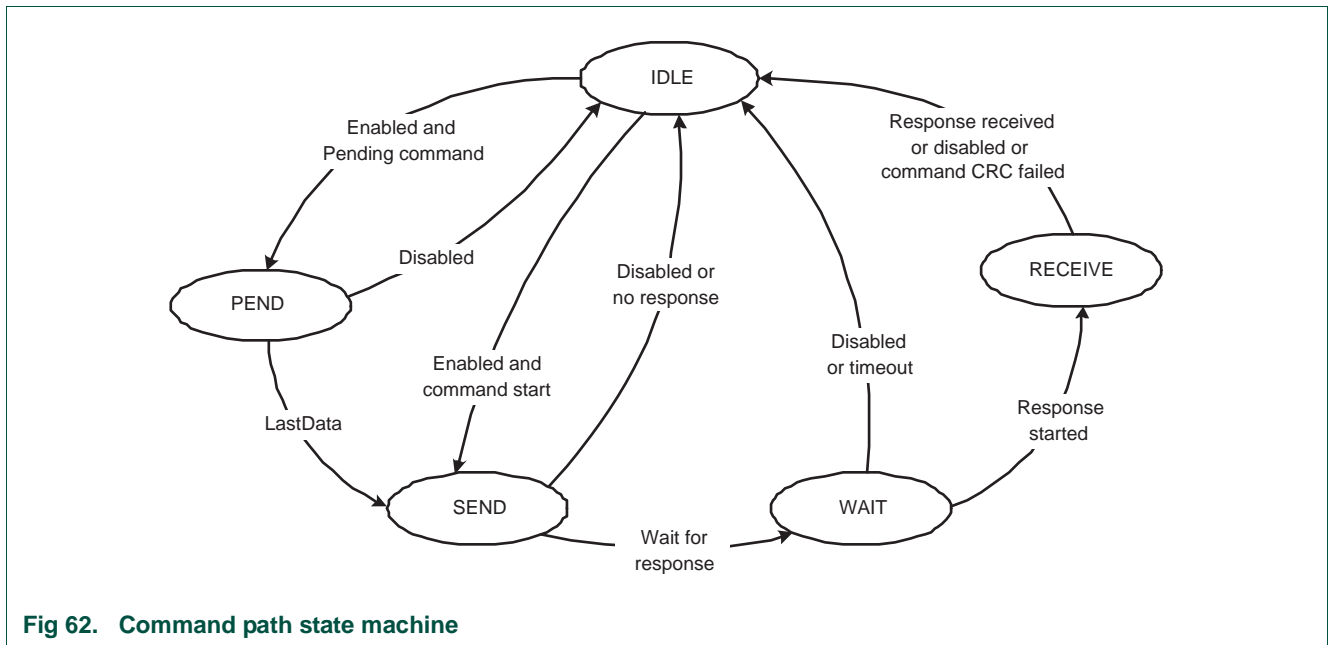


Fig 62. Command path state machine

When the WAIT state is entered, the command timer starts running. If the timeout<sup>1</sup> is reached before the CPSM moves to the RECEIVE state, the timeout flag is set and the IDLE<sup>2</sup> state is entered.

If the interrupt bit is set in the command register, the timer is disabled and the CPSM waits for an interrupt request from one of the cards. If a pending bit is set in the command register, the CPSM enters the PEND state, and waits for a CmdPend signal from the data path subunit. When CmdPend is detected, the CPSM moves to the SEND state. This enables the data counter to trigger the stop command transmission.

1. The timeout period has a fixed value of 64 SD\_CLK clocks period.  
 2. The CPSM remains in the IDLE state for at least eight SD\_CLK periods to meet Ncc and Nrc timing constraints.

Figure 63 shows the command transfer.

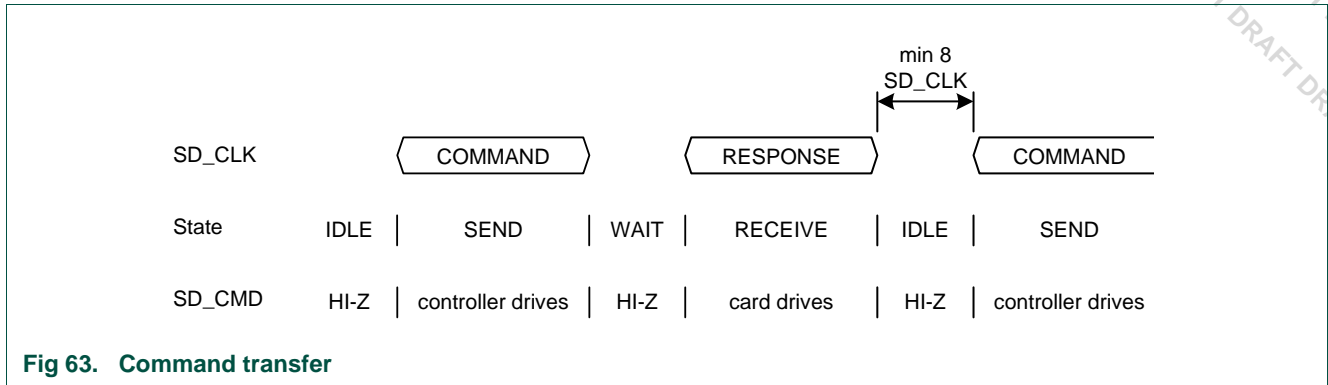


Fig 63. Command transfer

### 16.5.3.5 Command format

The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the CPSM is not in the SEND state, the SD\_CMD output is in HI-Z state, as shown in Figure 63. Data on SD\_CMD is synchronous to the rising SD\_CLK edge. All commands have a fixed length of 48 bits. Table 337 shows the command format.

Table 337. Command format

Bit Position	Width	Value	Description
0	1	1	End bit.
7:1	7	-	CRC7
39:8	32	-	Argument.
45:40	6	-	Command index.
46	1	1	Transmission bit.
47	1	0	Stat bit.

The SD card interface supports two response types. Both use CRC error checking:

- 48 bit short response (see Table 338)
- 136 bit long response (see Table 339)

Note: If the response does not contain CRC (CMD1 response), the device driver must ignore the CRC failed status.

Table 338. Simple response format

Bit Position	Width	Value	Description
0	1	1	End bit.
7:1	7	-	CRC7 (or 1111111).
39:8	32	-	Argument.
45:40	6	-	Command index.
46	1	0	Transmission bit.
47	1	0	Start bit.

**Table 339. Long response format**

Bit Position	Width	Value	Description
0	1	1	End bit.
127:1	127	-	CID or CSD (including internal CRC7).
133:128	6	111111	Reserved.
134	1	1	Transmission bit.
135	1	0	Start bit.

The command register contains the command index (six bits sent to a card) and the command type. These determine whether the command requires a response, and whether the response is 48 or 136 bits long (see [Section 16.6.4 “Command Register \(MCICommand - 0x400C 000C\)”](#) for more information). The command path implements the status flags shown in [Table 340](#) (see [Section 16.6.11 “Status Register \(MCISatus - 0x400C 0034\)”](#) for more information).

**Table 340. Command path status flags**

Flag	Description
CmdRespEnd	Set if response CRC is OK.
CmdCrcFail	Set if response CRC fails.
CmdSent	Set when command (that does not require response) is sent.
CmdTimeOut	Response timeout.
CmdActive	Command transfer in progress.

The CRC generator calculates the CRC checksum for all bits before the CRC code. This includes the start bit, transmitter bit, command index, and command argument (or card status). The CRC checksum is calculated for the first 120 bits of CID or CSD for the long response format. Note that the start bit, transmitter bit and the six reserved bits are not used in the CRC calculation.

The CRC checksum is a 7 bit value:

$$\text{CRC}[6:0] = \text{Remainder} [(M(x) \times x_7) / G(x)]$$

$$G(x) = x_7 + x_3 + 1$$

$$M(x) = (\text{start bit}) \times x_{39} + \dots + (\text{last bit before CRC}) \times x_0, \text{ or}$$

$$M(x) = (\text{start bit}) \times x_{119} + \dots + (\text{last bit before CRC}) \times x_0$$

### 16.5.3.6 Data path

The card data bus width can be programmed using the clock control register. If the wide bus mode is enabled, data is transferred at four bits per clock cycle over all four data signals (SD\_DAT[3:0]). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over SD\_DAT[0].

Depending on the transfer direction (send or receive), the Data Path State Machine (DPSM) moves to the WAIT\_S or WAIT\_R state when it is enabled:

- **Send:** The DPSM moves to the WAIT\_S state. If there is data in the send FIFO, the DPSM moves to the SEND state, and the data path subunit starts sending data to a card.

- Receive: The DPSM moves to the WAIT\_R state and waits for a start bit. When it receives a start bit, the DPSM moves to the RECEIVE state, and the data path subunit starts receiving data from a card.

16.5.3.7 Data path state machine

The DPSM operates at SD\_CLK frequency. Data on the card bus signals is synchronous to the rising edge of SD\_CLK. The DPSM has six states, as shown in Figure 64.

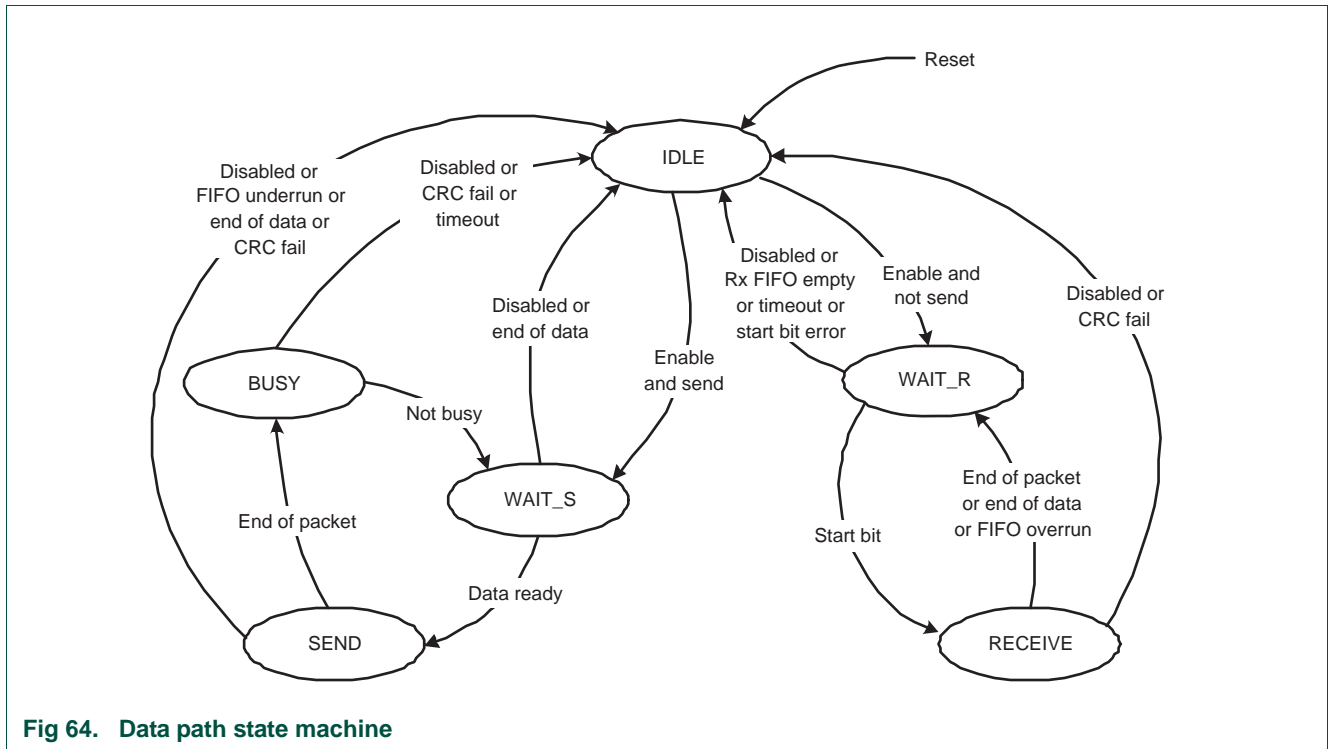


Fig 64. Data path state machine

- IDLE: The data path is inactive, and the SD\_DAT[3:0] outputs are in HI-Z. When the data control register is written and the enable bit is set, the DPSM loads the data counter with a new value and, depending on the data direction bit, moves to either the WAIT\_S or WAIT\_R state.

WAIT\_R: If the data counter equals zero, the DPSM moves to the IDLE state when the receive FIFO is empty. If the data counter is not zero, the DPSM waits for a start bit on SD\_DAT.

The DPSM moves to the RECEIVE state if it receives a start bit before a timeout, and loads the data block counter. If it reaches a timeout before it detects a start bit, or a start bit error occurs, it moves to the IDLE state and sets the timeout status flag.

- RECEIVE: Serial data received from a card is packed in bytes and written to the data FIFO. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM waits until it receives the CRC code. If the received code matches the internally generated CRC code, the DPSM moves to the WAIT\_R state. If not, the CRC fail status flag is set and the DPSM moves to the IDLE state.

- In stream mode, the DPSM receives data while the data counter is not zero. When the counter is zero, the remaining data in the shift register is written to the data FIFO, and the DPSM moves to the WAIT\_R state.

If a FIFO overrun error occurs, the DPSM sets the FIFO error flag and moves to the WAIT\_R state.

- WAIT\_S: The DPSM moves to the IDLE state if the data counter is zero. If not, it waits until the data FIFO empty flag is deasserted, and moves to the SEND state.

Note: The DPSM remains in the WAIT\_S state for at least two clock periods to meet Nwr timing constraints.

- SEND: The DPSM starts sending data to a card. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
  - In block mode, when the data block counter reaches zero, the DPSM sends an internally generated CRC code and end bit, and moves to the BUSY state.
  - In stream mode, the DPSM sends data to a card while the enable bit is HIGH and the data counter is not zero. It then moves to the IDLE state.

If a FIFO underrun error occurs, the DPSM sets the FIFO error flag and moves to the IDLE state.

- BUSY: The DPSM waits for the CRC status flag:
  - If it does not receive a positive CRC status, it moves to the IDLE state and sets the CRC fail status flag.
  - If it receives a positive CRC status, it moves to the WAIT\_S state if SD\_DAT[0] is not LOW (the card is not busy).

If a timeout occurs while the DPSM is in the BUSY state, it sets the data timeout flag and moves to the IDLE state.

The data timer is enabled when the DPSM is in the WAIT\_R or BUSY state, and generates the data timeout error:

- When transmitting data, the timeout occurs if the DPSM stays in the BUSY state for longer than the programmed timeout period
- When receiving data, the timeout occurs if the end of the data is not true, and if the DPSM stays in the WAIT\_R state for longer than the programmed timeout period.

### 16.5.3.8 Data counter

The data counter has two functions:

- To stop a data transfer when it reaches zero. This is the end of the data condition.
- To start transferring a pending command (see [Figure 65](#)). This is used to send the stop command for a stream data transfer.

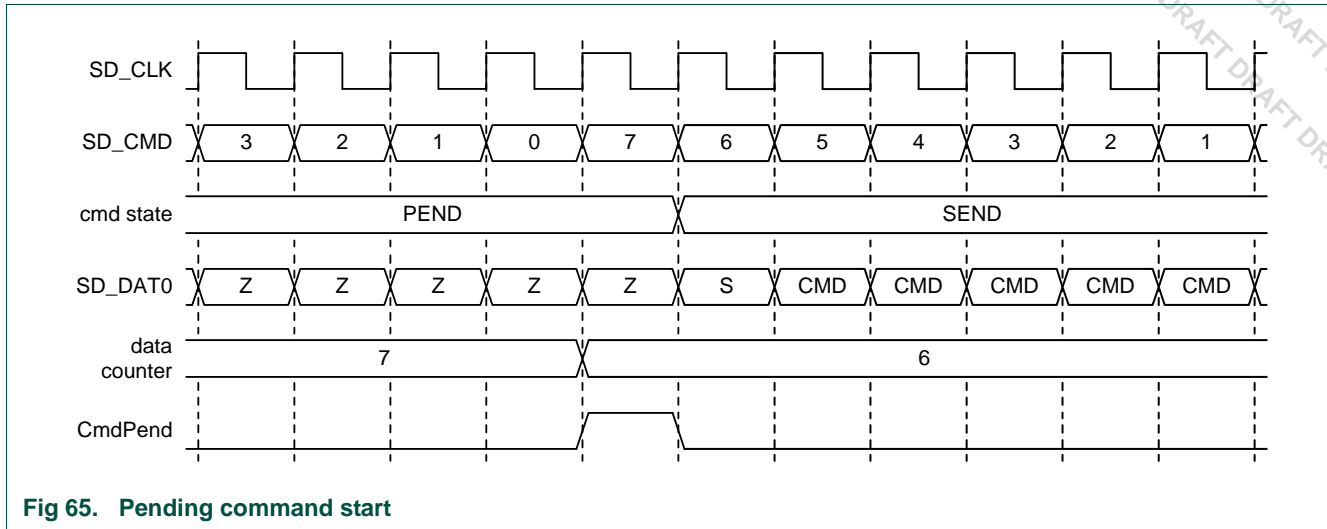


Fig 65. Pending command start

The data block counter determines the end of a data block. If the counter is zero, the end-of-data condition is TRUE (see [Section 16.6.9 “Data Control Register \(MCIDataCtrl - 0x400C 002C\)”](#) for more information).

### 16.5.3.9 Bus mode

In wide bus mode, all four data signals (SD\_DAT[3:0]) are used to transfer data, and the CRC code is calculated separately for each data signal. While transmitting data blocks to a card, only SD\_DAT[0] is used for the CRC token and busy signalling. The start bit must be transmitted on all four data signals at the same time (during the same clock period). If the start bit is not detected on all data signals on the same clock edge while receiving data, the DPSM sets the start bit error flag and moves to the IDLE state.

The data path also operates in half-duplex mode, where data is either sent to a card or received from a card. While not being transferred, SD\_DAT[3:0] are in the HI-Z state.

Data on these signals is synchronous to the rising edge of the clock period.

If standard bus mode is selected the SD\_DAT[3:1] outputs are always in HI-Z state and only the SD\_DAT[0] output is driven LOW when data is transmitted.

Design note: If wide mode is selected, all data outputs enabled at the same time. If not, the SD\_DAT[3:1] outputs are always off, and only the SD\_DAT[0] output is driven LOW when data is transmitted.

### 16.5.3.10 CRC Token status

The CRC token status follows each write data block, and determines whether a card has received the data block correctly. When the token has been received, the card asserts a busy signal by driving SD\_DAT[0] LOW. [Table 341](#) shows the CRC token status values.

Table 341. CRC token status

Token	Description
010	Card has received error-free data block.
101	Card has detected a CRC error.

### 16.5.3.11 Status flags

[Table 342](#) lists the data path status flags (see [Section 16.6.11 “Status Register \(MCISStatus - 0x400C 0034\)”](#) on page 423 for more information).

**Table 342. Data path status flags**

Flag	Description
TxFifoFull	Transmit FIFO is full.
TxFifoEmpty	Transmit FIFO is empty.
TxFifoHalfEmpty	Transmit FIFO is half full.
TxDataAvlbl	Transmit FIFO data available.
TxUnderrun	Transmit FIFO underrun error.
RxFifoFull	Receive FIFO is full.
RxFifoEmpty	Receive FIFO is empty.
RxFifoHalfFull	Receive FIFO is half full.
RxDataAvlbl	Receive FIFO data available.
RxOverrun	Receive FIFO overrun error.
DataBlockEnd	Data block sent/received.
StartBitErr	Start bit not detected on all data signals in wide bus mode.
DataCrcFail	Data packet CRC failed.
DataEnd	Data end (data counter is zero).
DataTimeOut	Data timeout.
TxActive	Data transmission in progress.
RxActive	Data reception in progress.

### 16.5.3.12 CRC generator

The CRC generator calculates the CRC checksum only for the data bits in a single block, and is bypassed in data stream mode. The checksum is a 16 bit value:

$$\text{CRC}[15:0] = \text{Remainder} [(M(x) \times x^{15}) / G(x)]$$

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

$$M(x) = (\text{first data bit}) \times x^n + \dots + (\text{last data bit}) \times x^0$$

### 16.5.3.13 Data FIFO

The data FIFO (first-in-first-out) subunit is a data buffer with transmit and receive logic.

The FIFO contains a 32 bit wide, 16-word deep data buffer, and transmit and receive logic. Because the data FIFO operates in the APB clock domain (PCLK), all signals from the subunits in the SD card interface clock domain (MCLK) are re-synchronized.

Depending on TxActive and RxActive, the FIFO can be disabled, transmit enabled, or receive enabled. TxActive and RxActive are driven by the data path subunit and are mutually exclusive:

- The transmit FIFO refers to the transmit logic and data buffer when TxActive is asserted (see [Section 16.5.3.14 “Transmit FIFO”](#))

- The receive FIFO refers to the receive logic and data buffer when RxActive is asserted (see [Section 16.5.3.15 “Receive FIFO”](#)).

### 16.5.3.14 Transmit FIFO

Data can be written to the transmit FIFO through the APB interface once the SD card interface is enabled for transmission.

The transmit FIFO is accessible via 16 sequential addresses (see [Section 16.6.15 “Data FIFO Register \(MCIFIFO - 0x400C 0080 to 0x400C 00BC\)”](#)). The transmit FIFO contains a data output register that holds the data word pointed to by the read pointer. When the data path subunit has loaded its shift register, it increments the read pointer and drives new data out.

If the transmit FIFO is disabled, all status flags are deasserted. The data path subunit asserts TxActive when it transmits data. [Table 343](#) lists the transmit FIFO status flags.

**Table 343. Transmit FIFO status flags**

Flag	Description
TxFifoFull	Set to HIGH when all 16 transmit FIFO words contain valid data.
TxFifoEmpty	Set to HIGH when the transmit FIFO does not contain valid data.
TxHalfEmpty	Set to HIGH when 8 or more transmit FIFO words are empty. This flag can be used as a DMA request.
TxDataAvlbl	Set to HIGH when the transmit FIFO contains valid data. This flag is the inverse of the TxFifoEmpty flag.
TxUnderrun	Set to HIGH when an underrun error occurs. This flag is cleared by writing to the MCIClear register.

### 16.5.3.15 Receive FIFO

When the data path subunit receives a word of data, it drives data on the write data bus and asserts the write enable signal. This signal is synchronized to the PCLK domain. The write pointer is incremented after the write is completed, and the receive FIFO control logic asserts RxWrDone, that then deasserts the write enable signal.

On the read side, the content of the FIFO word pointed to by the current value of the read pointer is driven on the read data bus. The read pointer is incremented when the APB bus interface asserts RxRdPrtInc.

If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts RxActive when it receives data. [Table 353](#) lists the receive FIFO status flags.

The receive FIFO is accessible via 16 sequential addresses (see [Section 16.6.15 “Data FIFO Register \(MCIFIFO - 0x400C 0080 to 0x400C 00BC\)”](#)).

If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts RxActive when it receives data. [Table 344](#) lists the receive FIFO status flags.



**Table 344. Receive FIFO status flags**

Symbol	Description
RxFifoFull	Set to HIGH when all 16 receive FIFO words contain valid data.
RxFifoEmpty	Set to HIGH when the receive FIFO does not contain valid data.
RxHalfFull	Set to HIGH when 8 or more receive FIFO words contain valid data. This flag can be used as a DMA request.
RxDataAvlbl	Set to HIGH when the receive FIFO is not empty. This flag is the inverse of the RxFifoEmpty flag.
RxOverrun	Set to HIGH when an overrun error occurs. This flag is cleared by writing to the MCIClear register.

### 16.5.3.16 APB interfaces

The APB interface generates the interrupt and DMA requests, and accesses the SD card interface registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic. DMA is controlled by the General Purpose DMA controller, see that chapter for details.

### 16.5.3.17 Interrupt logic

The interrupt logic generates an interrupt request signal that is asserted when at least one of the selected status flags is HIGH. A mask register is provided to allow selection of the conditions that will generate an interrupt. A status flag generates the interrupt request if a corresponding mask flag is set.

## 16.6 Register description

The SD card interface registers are shown in [Table 345](#).

**Table 345. Summary of SD card interface registers**

Name	Description	Access	Width	Reset Value <sup>[1]</sup>	Address	Table
MCIPower	Power control register.	R/W	8	0	0x400C 0000	<a href="#">346</a>
MCIClock	Clock control register.	R/W	12	0	0x400C 0004	<a href="#">347</a>
MCIArgument	Argument register.	R/W	32	0	0x400C 0008	<a href="#">348</a>
MMCCCommand	Command register.	R/W	11	0	0x400C 000C	<a href="#">349</a>
MCIRspCmd	Response command register.	RO	6	0	0x400C 0010	<a href="#">351</a>
MCIResponse0	Response register.	RO	32	0	0x400C 0014	<a href="#">352</a>
MCIResponse1	Response register.	RO	32	0	0x400C 0018	<a href="#">352</a>
MCIResponse2	Response register.	RO	32	0	0x400C 001C	<a href="#">352</a>
MCIResponse3	Response register.	RO	31	0	0x400C 0020	<a href="#">352</a>
MCIDataTimer	Data Timer.	R/W	32	0	0x400C 0024	<a href="#">354</a>
MCIDataLength	Data length register.	R/W	16	0	0x400C 0028	<a href="#">355</a>
MCIDataCtrl	Data control register.	R/W	8	0	0x400C 002C	<a href="#">356</a>
MCIDataCnt	Data counter.	RO	16	0	0x400C 0030	<a href="#">358</a>
MCIStatus	Status register.	RO	22	0	0x400C 0034	<a href="#">359</a>
MCIClear	Clear register.	WO	11	-	0x400C 0038	<a href="#">360</a>
MCIMask0	Interrupt 0 mask register.	R/W	22	0	0x400C 003C	<a href="#">361</a>
MCIFifoCnt	FIFO Counter.	RO	15	0	0x400C 0048	<a href="#">362</a>
MCIFIFO	Data FIFO Register.	R/W	32	0	0x400C 0080 to 0x400C 00BC	<a href="#">363</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 16.6.1 Power Control Register (MCIPWR - 0x400C 0000)

The MCIPWR register controls an external power supply. Power can be switched on and off, and adjust the output voltage. [Table 346](#) shows the bit assignment of the MCIPower register.

The active level of the SD\_PWR pin can be selected by bit 3 of the SCS register (see [Section 3.8.1 “System Controls and Status register \(SCS - 0x400F C1A0\)” on page 32](#) for details).

**Table 346: Power Control register (MCIPWR - address 0x400C 0000) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Ctrl	00	Power-off	00
		01	Reserved	
		10	Power-up	
		11	Power-on	
5:2	-		Reserved. Read value is undefined, only zero should be written.	NA

**Table 346: Power Control register (MCIPWR - address 0x400C 0000) bit description**

Bit	Symbol	Value	Description	Reset Value
6	OpenDrain		SD_CMD output control.	0
7	Rod		Rod control.	0
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

When the external power supply is switched on, the software first enters the power-up phase, and waits until the supply output is stable before moving to the power-on phase. During the power-up phase, SD\_PWR is set HIGH. The card bus outlets are disabled during both phases.

**Note:** After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

### 16.6.2 Clock Control Register (MCIClock - 0x400C 0004)

The MCIClock register controls the SD\_CLK output. [Table 347](#) shows the bit assignment of the clock control register.

**Table 347: Clock Control register (MCIClock - address 0x400C 0004) bit description**

Bit	Symbol	Value	Description	Reset Value
7:0	ClkDiv		Bus clock period: SD_CLK frequency = $MCLK / [2 \times (ClkDiv + 1)]$ .	0
8	Enable	0	Clock disabled.	0
		1	Clock enabled.	
9	PwrSave	0	Always enabled.	0
		1	Clock enabled when bus is active.	
10	Bypass	0	Disable bypass.	0
		1	Enable bypass. MCLK driven to card bus output (SD_CLK).	
11	WideBus	0	Standard bus mode (only SD_DAT[0] used).	0
		1	Wide bus mode (SD_DAT[3:0] used)	
31:12	-		Reserved. Read value is undefined, only zero should be written.	NA

While the SD card interface is in identification mode, the SD\_CLK frequency must be less than 400 kHz. The clock frequency can be changed to the maximum card bus frequency when relative card addresses are assigned to all cards.

**Note:** After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

### 16.6.3 Argument Register (MCIArument - 0x400C 0008)

The MCIArument register contains a 32 bit command argument, which is sent to a card as part of a command message. [Table 348](#) shows the bit assignment of the MCIArument register.

**Table 348: Argument register (MCIArument - address 0x400C 0008) bit description**

Bit	Symbol	Description	Reset Value
31:0	CmdArg	Command argument	0x0000 0000

If a command contains an argument, it must be loaded into the argument register before writing a command to the command register.

### 16.6.4 Command Register (MCICommand - 0x400C 000C)

The MCICommand register contains the command index and command type bits:

- The command index is sent to a card as part of a command message.
- The command type bits control the Command Path State Machine (CPSM). Writing 1 to the enable bit starts the command send operation, while clearing the bit disables the CPSM.

[Table 349](#) shows the bit assignment of the MCICommand register.

**Table 349: Command register (MCICommand - address 0x400C 000C) bit description**

Bit	Symbol	Description	Reset Value
5:0	CmdIndex	Command index.	0
6	Response	If set, CPSM waits for a response.	0
7	LongRsp	If set, CPSM receives a 136 bit long response.	0
8	Interrupt	If set, CPSM disables command timer and waits for interrupt request.	0
9	Pending	If set, CPSM waits for CmdPend before it starts sending a command.	0
10	Enable	If set, CPSM is enabled.	0
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA

**Note:** After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

[Table 350](#) shows the response types.

**Table 350: Command Response Types**

Response	Long Response	Description
0	0	No response, expect CmdSent flag.
0	1	No response, expect CmdSent flag.
1	0	Short response, expect CmdRespEnd or CmdCrcFail flag.
1	1	Long response, expect CmdRespEnd or CmdCrcFail flag.

### 16.6.5 Command Response Register (MCIRspCommand - 0x400C 0010)

The MCIRspCommand register contains the command index field of the last command response received. [Table 349](#) shows the bit assignment of the MCIRspCommand register.

**Table 351: Command Response register (MCIRspCommand - address 0x400C 0010) bit description**

Bit	Symbol	Description	Reset Value
5:0	RespCmd	Response command index	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA

If the command response transmission does not contain the command index field (long response), the RespCmd field is unknown, although it must contain 111111 (the value of the reserved field from the response).

### 16.6.6 Response Registers (MCIResponse0-3 - 0x400C 0014, 0x400C 0018, 0x400C 001C and 0x400C 0020)

The MCIResponse0-3 registers contain the status of a card, which is part of the received response. [Table 352](#) shows the bit assignment of the MCIResponse0-3 registers.

**Table 352: Response registers (MCIResponse0-3 - addresses 0x400C 0014, 0x400C 0018, 0x400C 001C and 0x400C 0020) bit description**

Bit	Symbol	Description	Reset Value
31:0	Status	Card status	0

The card status size can be 32 or 127 bits, depending on the response type (see [Table 353](#)).

**Table 353: Response Register Type**

Description	Short Response	Long Response
MCIResponse0	Card status [31:0]	Card status [127:96]
MCIResponse1	Unused	Card status [95:64]
MCIResponse2	Unused	Card status [63:32]
MCIResponse3	Unused	Card status [31:1]

The most significant bit of the card status is received first. The MCIResponse3 register LSBit is always 0.

### 16.6.7 Data Timer Register (MCIDataTimer - 0x400C 0024)

The MCIDataTimer register contains the data timeout period, in card bus clock periods. [Table 354](#) shows the bit assignment of the MCIDataTimer register.

**Table 354: Data Timer register (MCIDataTimer - address 0x400C 0024) bit description**

Bit	Symbol	Description	Reset Value
31:0	DataTime	Data timeout period.	0

A counter loads the value from the data timer register, and starts decrementing when the Data Path State Machine (DPSM) enters the WAIT\_R or BUSY state. If the timer reaches 0 while the DPSM is in either of these states, the timeout status flag is set.

A data transfer must be written to the data timer register and the data length register before being written to the data control register.

### 16.6.8 Data Length Register (MCIDataLength - 0x400C 0028)

The MCIDataLength register contains the number of data bytes to be transferred. The value is loaded into the data counter when data transfer starts. [Table 355](#) shows the bit assignment of the MCIDataLength register.

**Table 355: Data Length register (MCIDataLength - address 0x400C 0028) bit description**

Bit	Symbol	Description	Reset Value
15:0	DataLength	Data length value	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

For a block data transfer, the value in the data length register must be a multiple of the block size (see [Section 16.6.9 “Data Control Register \(MCIDataCtrl - 0x400C 002C\)”](#)).

To initiate a data transfer, write to the data timer register and the data length register before writing to the data control register.

### 16.6.9 Data Control Register (MCIDataCtrl - 0x400C 002C)

The MCIDataCtrl register controls the DPSM. [Table 356](#) shows the bit assignment of the MCIDataCtrl register.

**Table 356: Data Control register (MCIDataCtrl - address 0x400C 002C) bit description**

Bit	Symbol	Value	Description	Reset Value
0	Enable		Data transfer enable.	0
1	Direction		Data transfer direction:	0
		0	From controller to card.	
		1	From card to controller.	
2	Mode		Data transfer mode:	0
		0	Block data transfer.	
		1	Stream data transfer.	
3	DMAEnable		Enable DMA:	0
		0	DMA disabled.	
		1	DMA enabled.	
7:4	BlockSize		Data block length	0
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

**Note:** After a data write, data cannot be written to this register for three MCLK clock periods plus two PCLK clock periods.

Data transfer starts if 1 is written to the enable bit. Depending on the direction bit, the DPSM moves to the WAIT\_S or WAIT\_R state. It is not necessary to clear the enable bit after the data transfer. BlockSize controls the data block length if Mode is 0, as shown in [Table 357](#).

**Table 357: Data Block Length**

Block Size	Block Length
0	2 <sup>0</sup> = 1 byte.
1	2 <sup>1</sup> = 2 bytes.
...	-
11	2 <sup>11</sup> = 2048 bytes.
12:15	Reserved.

### 16.6.10 Data Counter Register (MCIDataCnt - 0x400C 0030)

The MCIDataCnt register loads the value from the data length register (see [Section 16.6.8 “Data Length Register \(MCIDataLength - 0x400C 0028\)”](#)) when the DPSM moves from the IDLE state to the WAIT\_R or WAIT\_S state. As data is transferred, the counter decrements the value until it reaches 0. The DPSM then moves to the IDLE state and the data status end flag is set. [Table 358](#) shows the bit assignment of the MCIDataCnt register.

**Table 358: Data Counter register (MCIDataCnt - address 0x400C 0030) bit description**

Bit	Symbol	Description	Reset Value
15:0	DataCount	Remaining data	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

**Note:** This register should be read only when the data transfer is complete.

### 16.6.11 Status Register (MCIStatus - 0x400C 0034)

The MCIStatus register is a read-only register. It contains two types of flag:

- Static [10:0]: These remain asserted until they are cleared by writing to the Clear register (see [Section 16.6.12 “Clear Register \(MCIClear - 0x400C 0038\)”](#)).
- Dynamic [21:11]: These change state depending on the state of the underlying logic (for example, FIFO full and empty flags are asserted and deasserted as data while written to the FIFO).

[Table 359](#) shows the bit assignment of the MCIStatus register.

**Table 359: Status register (MCIStatus - address 0x400C 0034) bit description**

Bit	Symbol	Description	Reset Value
0	CmdCrcFail	Command response received (CRC check failed).	0
1	DataCrcFail	Data block sent/received (CRC check failed).	0
2	CmdTimeOut	Command response timeout.	0
3	DataTimeOut	Data timeout.	0
4	TxUnderrun	Transmit FIFO underrun error.	0
5	RxOverrun	Receive FIFO overrun error.	0
6	CmdRespEnd	Command response received (CRC check passed).	0
7	CmdSent	Command sent (no response required).	0
8	DataEnd	Data end (data counter is zero).	0

**Table 359: Status register (MCISStatus - address 0x400C 0034) bit description**

Bit	Symbol	Description	Reset Value
9	StartBitErr	Start bit not detected on all data signals in wide bus mode.	0
10	DataBlockEnd	Data block sent/received (CRC check passed).	0
11	CmdActive	Command transfer in progress.	0
12	TxActive	Data transmit in progress.	0
13	RxActive	Data receive in progress.	0
14	TxFifoHalfEmpty	Transmit FIFO half empty.	0
15	RxFifoHalfFull	Receive FIFO half full.	0
16	TxFifoFull	Transmit FIFO full.	0
17	RxFifoFull	Receive FIFO full.	0
18	TxFifoEmpty	Transmit FIFO empty.	0
19	RxFifoEmpty	Receive FIFO empty.	0
20	TxDataAvlbl	Data available in transmit FIFO.	0
21	RxDataAvlbl	Data available in receive FIFO.	0
31:22	-	Reserved. The value read from a reserved bit is not defined.	NA

### 16.6.12 Clear Register (MCIClear - 0x400C 0038)

The MCIClear register is a write-only register. The corresponding static status flags can be cleared by writing a 1 to the corresponding bit in the register. [Table 360](#) shows the bit assignment of the MCIClear register.

**Table 360: Clear register (MCIClear - address 0x400C 0038) bit description**

Bit	Symbol	Description	Reset Value
0	CmdCrcFailClr	Clears CmdCrcFail flag.	-
1	DataCrcFailClr	Clears DataCrcFail flag.	-
2	CmdTimeOutClr	Clears CmdTimeOut flag.	-
3	DataTimeOutClr	Clears DataTimeOut flag.	-
4	TxUnderrunClr	Clears TxUnderrun flag.	-
5	RxOverrunClr	Clears RxOverrun flag.	-
6	CmdRespEndClr	Clears CmdRespEnd flag.	-
7	CmdSentClr	Clears CmdSent flag.	-
8	DataEndClr	Clears DataEnd flag.	-
9	StartBitErrClr	Clears StartBitErr flag.	-
10	DataBlockEndClr	Clears DataBlockEnd flag.	-
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA

### 16.6.13 Interrupt Mask Registers (MCIMask0 - 0x400C 003C)

The interrupt mask registers determine which status flags generate an interrupt request by setting the corresponding bit to 1. [Table 361](#) shows the bit assignment of the MCIMaskx registers.



**Table 361: Interrupt Mask registers (MCIMask0 - address 0x400C 003C) bit description**

Bit	Symbol	Description	Reset Value
0	Mask0	Mask CmdCrcFail flag.	0
1	Mask1	Mask DataCrcFail flag.	0
2	Mask2	Mask CmdTimeOut flag.	0
3	Mask3	Mask DataTimeOut flag.	0
4	Mask4	Mask TxUnderrun flag.	0
5	Mask5	Mask RxOverrun flag.	0
6	Mask6	Mask CmdRespEnd flag.	0
7	Mask7	Mask CmdSent flag.	0
8	Mask8	Mask DataEnd flag.	0
9	Mask9	Mask StartBitErr flag.	0
10	Mask10	Mask DataBlockEnd flag.	0
11	Mask11	Mask CmdActive flag.	0
12	Mask12	Mask TxActive flag.	0
13	Mask13	Mask RxActive flag.	0
14	Mask14	Mask TxFifoHalfEmpty flag.	0
15	Mask15	Mask RxFifoHalfFull flag.	0
16	Mask16	Mask TxFifoFull flag.	0
17	Mask17	Mask RxFifoFull flag.	0
18	Mask18	Mask TxFifoEmpty flag.	0
19	Mask19	Mask RxFifoEmpty flag.	0
20	Mask20	Mask TxDataAvlbl flag.	0
21	Mask21	Mask RxDataAvlbl flag.	0
31:22	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 16.6.14 FIFO Counter Register (MCIFifoCnt - 0x400C 0048)

The MCIFifoCnt register contains the remaining number of words to be written to or read from the FIFO. The FIFO counter loads the value from the data length register (see [Section 16.6.8 “Data Length Register \(MCIDataLength - 0x400C 0028\)”](#)) when the Enable bit is set in the data control register. If the data length is not word aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word. [Table 362](#) shows the bit assignment of the MCIFifoCnt register.

**Table 362: FIFO Counter register (MCIFifoCnt - address 0x400C 0048) bit description**

Bit	Symbol	Description	Reset Value
14:0	DataCount	Remaining data	0
31:15	-	Reserved. Read value is undefined, only zero should be written.	NA

### 16.6.15 Data FIFO Register (MCIFIFO - 0x400C 0080 to 0x400C 00BC)

The receive and transmit FIFOs can be read or written as 32 bit wide registers. The FIFOs contain 16 entries on 16 sequential addresses. This allows the microprocessor to use its load and store multiple operands to read/write to the FIFO. [Table 363](#) shows the bit assignment of the MCIFIFO register.

**Table 363: Data FIFO register (MCIFIFO - address 0x400C 0080 to 0x400C 00BC) bit description**

Bit	Symbol	Description	Reset Value
31:0	Data	FIFO data.	0

### 17.1 Basic configuration

The UART1 peripheral is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bits PCUART1.  
**Remark:** On reset, UART1 is enabled (PCUART1 = 1).
2. Peripheral clock: UART1 operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Baud rate: In register U1LCR ([Table 374](#)), set bit DLAB =1. This enables access to registers DLL ([Table 368](#)) and DLM ([Table 369](#)) for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register ([Table 381](#)).
4. UART FIFO: Use bit FIFO enable (bit 0) in register U1FCR ([Table 373](#)) to enable the FIFOs.
5. Pins: Select UART pins and pin modes through the in the relevant IOCON registers ([Section 8.4.1](#)).  
**Remark:** UART receive pins should not have pull-down resistors enabled.
6. Interrupts: To enable UART interrupts set bit DLAB =0 in register U1LCR ([Table 374](#)). This enables access to U1IER ([Table 370](#)). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
7. DMA: UART1 transmit and receive functions can operated with the GPDMA controller (see [Table 655](#)).

### 17.2 Features

- Full modem control handshaking available
- Data sizes of 5, 6, 7, and 8 bits.
- Parity generation and checking: odd, even mark, space or none.
- One or two stop bits.
- 16 byte Receive and Transmit FIFOs.
- Built-in baud rate generator, including a fractional rate divider for great versatility.
- Supports DMA for both transmit and receive.
- Auto-baud capability
- Break generation and detection.
- Multiprocessor addressing mode.
- RS-485/EIA-485 support.

## 17.3 Architecture

The architecture of the UART1 is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART1.

The UART1 receiver block, U1RX, monitors the serial input line, RXD1, for valid input. The UART1 RX Shift Register (U1RSR) accepts valid characters via RXD1. After a valid character is assembled in the U1RSR, it is passed to the UART1 RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART1 transmitter block, U1TX, accepts data written by the CPU or host and buffers the data in the UART1 TX Holding Register FIFO (U1THR). The UART1 TX Shift Register (U1TSR) reads the data stored in the U1THR and assembles the data to transmit via the serial output pin, TXD1.

The UART1 Baud Rate Generator block, U1BRG, generates the timing enables used by the UART1 TX block. The U1BRG clock input source is the APB clock (PCLK). The main clock is divided down per the divisor specified in the U1DLL and U1DLM registers. This divided down clock is the 16x oversample clock.

The modem interface contains registers U1MCR and U1MSR. This interface is responsible for handshaking between a modem peripheral and the UART1.

The interrupt interface contains registers U1IER and U1IIR. The interrupt interface receives several one clock wide enables from the U1TX and U1RX blocks.

Status information from the U1TX and U1RX is stored in the U1LSR. Control information for the U1TX and U1RX is stored in the U1LCR.

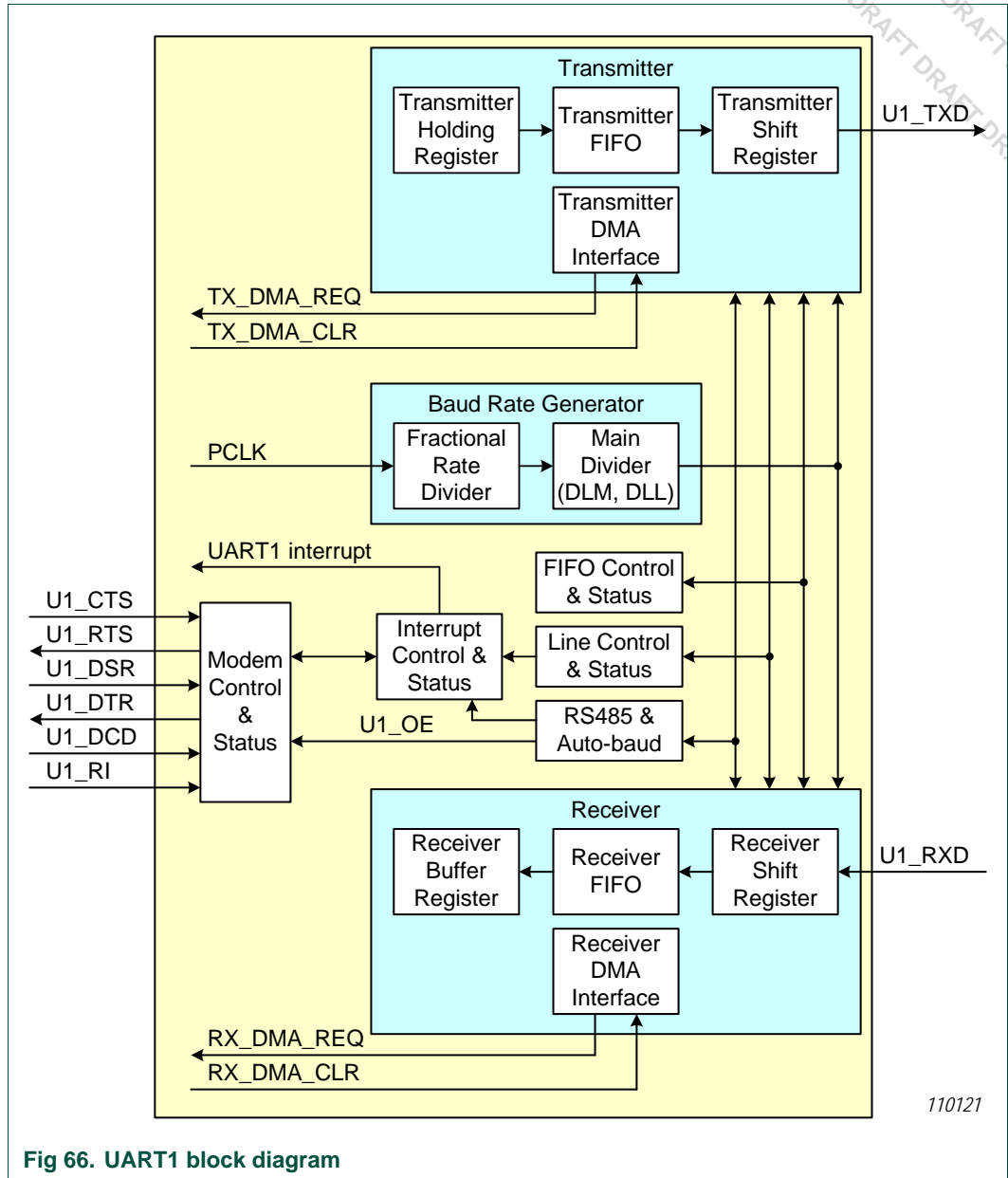


Fig 66. UART1 block diagram

## 17.4 Pin description

Table 364: UART1 Pin Description

Pin	Type	Description
U1_RXD	Input	<b>Serial Input.</b> Serial receive data.
U1_TXD	Output	<b>Serial Output.</b> Serial transmit data.
U1_CTS	Input	<p><b>Clear To Send.</b> Active low signal indicates if the external modem is ready to accept transmitted data via TXD1 from the UART1. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[4]. State change information is stored in U1MSR[0] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).</p> <p>Clear to send. CTS1 is an asynchronous, active low modem status signal. Its condition can be checked by reading bit 4 (CTS) of the modem status register. Bit 0 (DCTS) of the Modem Status Register (MSR) indicates that CTS1 has changed states since the last read from the MSR. If the modem status interrupt is enabled when CTS1 changes levels and the auto-cts mode is not enabled, an interrupt is generated. CTS1 is also used in the auto-cts mode to control the transmitter.</p>
U1_DCD	Input	<p><b>Data Carrier Detect.</b> Active low signal indicates if the external modem has established a communication link with the UART1 and data may be exchanged. In normal operation of the modem interface (U1MCR[4]=0), the complement value of this signal is stored in U1MSR[7]. State change information is stored in U1MSR3 and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).</p>
U1_DSR	Input	<p><b>Data Set Ready.</b> Active low signal indicates if the external modem is ready to establish a communications link with the UART1. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[5]. State change information is stored in U1MSR[1] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).</p>
U1_DTR	Output	<p>Data Terminal Ready. Active low signal indicates that the UART1 is ready to establish connection with external modem. The complement value of this signal is stored in U1MCR[0].</p> <p>The DTR pin can also be used as an RS-485/EIA-485 output enable signal.</p>
U1_RI	Input	<p><b>Ring Indicator.</b> Active low signal indicates that a telephone ringing signal has been detected by the modem. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[6]. State change information is stored in U1MSR[2] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).</p>
U1_RTS	Output	<p><b>Request To Send.</b> Active low signal indicates that the UART1 would like to transmit data to the external modem. The complement value of this signal is stored in U1MCR[1].</p> <p>In auto-rts mode, RTS1 is used to control the transmitter FIFO threshold logic.</p> <p>Request to send. RTS1 is an active low signal informing the modem or data set that the UART is ready to receive data. RTS1 is set to the active (low) level by setting the RTS modem control register bit and is set to the inactive (high) level either as a result of a system reset or during loop-back mode operations or by clearing bit 1 (RTS) of the MCR. In the auto-rts mode, RTS1 is controlled by the transmitter FIFO threshold logic.</p> <p>The RTS pin can also be used as an RS-485/EIA-485 output enable signal.</p>

## 17.5 Register description

UART1 contains registers organized as shown in [Table 365](#). The Divisor Latch Access Bit (DLAB) is contained in U1LCR[7] and enables access to the Divisor Latches.

**Table 365: UART1 register map**

Name	Description	Access	Reset Value <sup>[1]</sup>	Address	Table
U1RBR (DLAB =0)	Receiver Buffer Register. Contains the next received character to be read.	RO	NA	0x4001 0000	<a href="#">366</a>
U1THR (DLAB =0)	Transmit Holding Register. The next character to be transmitted is written here.	WO	NA	0x4001 0000	<a href="#">367</a>
U1DLL (DLAB =1)	Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x01	0x4001 0000	<a href="#">368</a>
U1DLM (DLAB =1)	Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0	0x4001 0004	<a href="#">369</a>
U1IER (DLAB =0)	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART1 interrupts.	R/W	0	0x4001 0004	<a href="#">370</a>
U1IIR	Interrupt ID Register. Identifies which interrupt(s) are pending.	RO	0x01	0x4001 0008	<a href="#">371</a>
U1FCR	FIFO Control Register. Controls UART1 FIFO usage and modes.	WO	0	0x4001 0008	<a href="#">373</a>
U1LCR	Line Control Register. Contains controls for frame formatting and break generation.	R/W	0	0x4001 000C	<a href="#">374</a>
U1MCR	Modem Control Register. Contains controls for flow control handshaking and loopback mode.	R/W	0	0x4001 0010	<a href="#">375</a>
U1LSR	Line Status Register. Contains flags for transmit and receive status, including line errors.	RO	0x60	0x4001 0014	<a href="#">377</a>
U1MSR	Modem Status Register. Contains handshake signal status flags.	RO	0	0x4001 0018	<a href="#">378</a>
U1SCR	Scratch Pad Register. 8-bit temporary storage for software.	R/W	0	0x4001 001C	<a href="#">379</a>
U1ACR	Auto-baud Control Register. Contains controls for the auto-baud feature.	R/W	0	0x4001 0020	<a href="#">380</a>
U1FDR	Fractional Divider Register. Generates a clock input for the baud rate divider.	R/W	0x10	0x4001 0028	<a href="#">381</a>
U1TER	Transmit Enable Register. Turns off UART transmitter for use with software flow control.	R/W	0x80	0x4001 0030	<a href="#">383</a>
U1RS485CTRL	RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes.	R/W	0	0x4001 004C	<a href="#">384</a>
U1ADRMATCH	RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode.	R/W	0	0x4001 0050	<a href="#">385</a>
U1RS485DLY	RS-485/EIA-485 direction control delay.	R/W	0	0x4001 0054	<a href="#">386</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 17.5.1 UART1 Receiver Buffer Register (U1RBR - 0x4001 0000, when DLAB = 0)

The U1RBR is the top byte of the UART1 RX FIFO. The top byte of the RX FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in U1LCR must be zero in order to access the U1RBR. The U1RBR is always read-only.

Since PE, FE and BI bits correspond to the byte sitting on the top of the RBR FIFO (i.e. the one that will be read in the next read from the RBR), the right approach for fetching the valid pair of received byte and its status bits is first to read the content of the U1LSR register, and then to read a byte from the U1RBR.

**Table 366: UART1 Receiver Buffer Register (U1RBR - address 0x4001 0000 when DLAB = 0) bit description**

Bit	Symbol	Description	Reset Value
7:0	RBR	The UART1 Receiver Buffer Register contains the oldest received byte in the UART1 RX FIFO.	undefined
31:8	-	Reserved, the value read from a reserved bit is not defined.	NA

### 17.5.2 UART1 Transmitter Holding Register (U1THR - 0x4001 0000 when DLAB = 0)

The write-only U1THR is the top byte of the UART1 TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

The Divisor Latch Access Bit (DLAB) in U1LCR must be zero in order to access the U1THR. The U1THR is write-only.

**Table 367: UART1 Transmitter Holding Register (U1THR - address 0x4001 0000 when DLAB = 0) bit description**

Bit	Symbol	Description	Reset Value
7:0	THR	Writing to the UART1 Transmit Holding Register causes the data to be stored in the UART1 transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.3 UART1 Divisor Latch LSB and MSB Registers (U1DLL - 0x4001 0000 and U1DLM - 0x4001 0004, when DLAB = 1)

The UART1 Divisor Latch is part of the UART1 Baud Rate Generator and holds the value used, along with the Fractional Divider, to divide the APB clock (PCLK) in order to produce the baud rate clock, which must be 16x the desired baud rate. The U1DLL and U1DLM registers together form a 16-bit divisor where U1DLL contains the lower 8 bits of the divisor and U1DLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed. The Divisor Latch Access Bit (DLAB) in U1LCR must be one in order to access the UART1 Divisor Latches. Details on how to select the right value for U1DLL and U1DLM can be found later in this chapter, see [Section 17.5.16](#).



**Table 368: UART1 Divisor Latch LSB Register (U1DLL - address 0x4001 0000 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLLSB	The UART1 Divisor Latch LSB Register, along with the U1DLM register, determines the baud rate of the UART1.	0x01
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

**Table 369: UART1 Divisor Latch MSB Register (U1DLM - address 0x4001 0004 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLMSB	The UART1 Divisor Latch MSB Register, along with the U1DLL register, determines the baud rate of the UART1.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.4 UART1 Interrupt Enable Register (U1IER - 0x4001 0004, when DLAB = 0)

The U1IER is used to enable the four UART1 interrupt sources.

**Table 370: UART1 Interrupt Enable Register (U1IER - address 0x4001 0004 when DLAB = 0) bit description**

Bit	Symbol	Value	Description	Reset Value
0	RBR Interrupt Enable		enables the Receive Data Available interrupt for UART1. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	
1	THRE Interrupt Enable		enables the THRE interrupt for UART1. The status of this interrupt can be read from U1LSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Interrupt Enable		enables the UART1 RX line status interrupts. The status of this interrupt can be read from U1LSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
3	Modem Status Interrupt Enable		enables the modem interrupt. The status of this interrupt can be read from U1MSR[3:0].	0
		0	Disable the modem interrupt.	
		1	Enable the modem interrupt.	
6:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7	CTS Interrupt Enable		If auto-cts mode is enabled this bit enables/disables the modem status interrupt generation on a CTS1 signal transition. If auto-cts mode is disabled a CTS1 transition will generate an interrupt if Modem Status Interrupt Enable (U1IER[3]) is set.  In normal operation a CTS1 signal transition will generate a Modem Status Interrupt unless the interrupt has been disabled by clearing the U1IER[3] bit in the U1IER register. In auto-cts mode a transition on the CTS1 bit will trigger an interrupt only if both the U1IER[3] and U1IER[7] bits are set.	0
		0	Disable the CTS interrupt.	
		1	Enable the CTS interrupt.	

**Table 370: UART1 Interrupt Enable Register (U1IER - address 0x4001 0004 when DLAB = 0) bit description**

Bit	Symbol	Value	Description	Reset Value
8	ABEOIntEn		Enables the end of auto-baud interrupt.	0
		0	Disable end of auto-baud Interrupt.	
		1	Enable end of auto-baud Interrupt.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable auto-baud time-out Interrupt.	
		1	Enable auto-baud time-out Interrupt.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.5 UART1 Interrupt Identification Register (U1IIR - 0x4001 0008)

The U1IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an U1IIR access. If an interrupt occurs during an U1IIR access, the interrupt is recorded for the next U1IIR access.

**Table 371: UART1 Interrupt Identification Register (U1IIR - address 0x4001 0008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	IntStatus		Interrupt status. Note that U1IIR[0] is active low. The pending interrupt can be determined by evaluating U1IIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	IntId		Interrupt identification. U1IER[3:1] identifies an interrupt corresponding to the UART1 Rx or TX FIFO. All other combinations of U1IER[3:1] not listed below are reserved (100,101,111).	0
		011	1 - Receive Line Status (RLS).	
		010	2a - Receive Data Available (RDA).	
		110	2b - Character Time-out Indicator (CTI).	
		001	3 - THRE Interrupt.	
		000	4 - Modem Interrupt.	
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	FIFO Enable		Copies of U1FCR[0].	0
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

Bit U1IIR[9:8] are set by the auto-baud function and signal a time-out or end of auto-baud condition. The auto-baud interrupt conditions are cleared by setting the corresponding Clear bits in the Auto-baud Control Register.

If the IntStatus bit is 1 no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in [Table 372](#). Given the status of U1IIR[3:0], an

interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The U1IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UART1 RLS interrupt (U1IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART1RX input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART1 Rx error condition that set the interrupt can be observed via U1LSR[4:1]. The interrupt is cleared upon an U1LSR read.

The UART1 RDA interrupt (U1IIR[3:1] = 010) shares the second level priority with the CTI interrupt (U1IIR[3:1] = 110). The RDA is activated when the UART1 Rx FIFO reaches the trigger level defined in U1FCR7:6 and is reset when the UART1 Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (U1IIR[3:1] = 110) is a second level interrupt and is set when the UART1 Rx FIFO contains at least one character and no UART1 Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART1 Rx FIFO activity (read or write of UART1 RSR) will clear the interrupt. This interrupt is intended to flush the UART1 RBR after a message has been received that is not a multiple of the trigger level size. For example, if a peripheral wished to send a 105 character message and the trigger level was 10 characters, the CPU would receive 10 RDA interrupts resulting in the transfer of 100 characters and 1 to 5 CTI interrupts (depending on the service routine) resulting in the transfer of the remaining 5 characters.

**Table 372: UART1 Interrupt Handling**

U1IIR[3:0] value <sup>[1]</sup>	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE <sup>[2]</sup> or PE <sup>[2]</sup> or FE <sup>[2]</sup> or BI <sup>[2]</sup>	U1LSR Read <sup>[2]</sup>
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (U1FCR0=1)	U1RBR Read <sup>[3]</sup> or UART1 FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the RX FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).  The exact time will be: [(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs	U1RBR Read <sup>[3]</sup>
0010	Third	THRE	THRE <sup>[2]</sup>	U1IIR Read <sup>[4]</sup> (if source of interrupt) or THR write
0000	Fourth	Modem Status	CTS or DSR or RI or DCD	MSR Read

[1] Values "0000", "0011", "0101", "0111", "1000", "1001", "1010", "1011", "1101", "1110", "1111" are reserved.

[2] For details see [Section 17.5.10 "UART1 Line Status Register \(U1LSR - 0x4001 0014\)"](#)

[3] For details see [Section 17.5.1 "UART1 Receiver Buffer Register \(U1RBR - 0x4001 0000, when DLAB = 0\)"](#)

[4] For details see [Section 17.5.5 "UART1 Interrupt Identification Register \(U1IIR - 0x4001 0008\)"](#) and [Section 17.5.2 "UART1 Transmitter Holding Register \(U1THR - 0x4001 0000 when DLAB = 0\)"](#)

The UART1 THRE interrupt (U1IIR[3:1] = 001) is a third level interrupt and is activated when the UART1 THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART1 THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE = 1 and there have not been at least two characters in the U1THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to U1THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UART1 THR FIFO has held two or more characters at one time and currently, the U1THR is empty. The THRE interrupt is reset when a U1THR write occurs or a read of the U1IIR occurs and the THRE is the highest interrupt (U1IIR[3:1] = 001).

It is the lowest priority interrupt and is activated whenever there is any state change on modem inputs pins, DCD, DSR or CTS. In addition, a low to high transition on modem input RI will generate a modem interrupt. The source of the modem interrupt can be determined by examining U1MSR[3:0]. A U1MSR read will clear the modem interrupt.

### 17.5.6 UART1 FIFO Control Register (U1FCR - 0x4001 0008)

The write-only U1FCR controls the operation of the UART1 RX and TX FIFOs.

**Table 373: UART1 FIFO Control Register (U1FCR - address 0x4001 0008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	FIFO Enable	0	UART1 FIFOs are disabled. Must not be used in the application.	0
		1	Active high enable for both UART1 Rx and TX FIFOs and U1FCR[7:1] access. This bit must be set for proper UART1 operation. Any transition on this bit will automatically clear the UART1 FIFOs.	
1	RX FIFO Reset	0	No impact on either of UART1 FIFOs.	0
		1	Writing a logic 1 to U1FCR[1] will clear all bytes in UART1 Rx FIFO, reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UART1 FIFOs.	0
		1	Writing a logic 1 to U1FCR[2] will clear all bytes in UART1 TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	DMA Mode Select		When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode. See <a href="#">Section 17.5.6.1</a> .	0
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UART1 FIFO characters must be written before an interrupt is activated.	0
		00	Trigger level 0 (1 character or 0x01).	
		01	Trigger level 1 (4 characters or 0x04).	
		10	Trigger level 2 (8 characters or 0x08).	
		11	Trigger level 3 (14 characters or 0x0E).	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

#### 17.5.6.1 DMA Operation

The user can optionally operate the UART transmit and/or receive using DMA. The DMA mode is determined by the DMA Mode Select bit in the FCR register. This bit only has an affect when the FIFOs are enabled via the FIFO Enable bit in the FCR register.

**UART receiver DMA**

In DMA mode, the receiver DMA request is asserted on the event of the receiver FIFO level becoming equal to or greater than trigger level, or if a character timeout occurs. See the description of the RX Trigger Level above. The receiver DMA request is cleared by the DMA controller.

**UART transmitter DMA**

In DMA mode, the transmitter DMA request is asserted on the event of the transmitter FIFO transitioning to not full. The transmitter DMA request is cleared by the DMA controller.

**17.5.7 UART1 Line Control Register (U1LCR - 0x4001 000C)**

The U1LCR determines the format of the data character that is to be transmitted or received.

**Table 374: UART1 Line Control Register (U1LCR - address 0x4001 000C) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Word Length Select	00	5-bit character length.	0
		01	6-bit character length.	
		10	7-bit character length.	
		11	8-bit character length.	
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if U1LCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	0
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		10	Forced "1" stick parity.	
		11	Forced "0" stick parity.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UART1 TXD is forced to logic 0 when U1LCR[6] is active high.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

**17.5.8 UART1 Modem Control Register (U1MCR - 0x4001 0010)**

The U1MCR enables the modem loopback mode and controls the modem output signals.

**Table 375: UART1 Modem Control Register (U1MCR - address 0x4001 0010) bit description**

Bit	Symbol	Value	Description	Reset value
0	DTR Control		Source for modem output pin, DTR. This bit reads as 0 when modem loopback mode is active.	0
1	RTS Control		Source for modem output pin RTS. This bit reads as 0 when modem loopback mode is active.	0
3-2	-		Reserved. Read value is undefined, only zero should be written.	0
4	Loopback Mode Select		The modem loopback mode provides a mechanism to perform diagnostic loopback testing. Serial data from the transmitter is connected internally to serial input of the receiver. Input pin, RXD1, has no effect on loopback and output pin, TXD1 is held in marking state. The 4 modem inputs (CTS, DSR, RI and DCD) are disconnected externally. Externally, the modem outputs (RTS, DTR) are set inactive. Internally, the 4 modem outputs are connected to the 4 modem inputs. As a result of these connections, the upper 4 bits of the U1MSR will be driven by the lower 4 bits of the U1MCR rather than the 4 modem inputs in normal mode. This permits modem status interrupts to be generated in loopback mode by writing the lower 4 bits of U1MCR.	0
		0	Disable modem loopback mode.	
		1	Enable modem loopback mode.	
5	-		Reserved. Read value is undefined, only zero should be written.	0
6	RTSen	0	Disable auto-rts flow control.	0
		1	Enable auto-rts flow control.	
7	CTSen	0	Disable auto-cts flow control.	0
		1	Enable auto-cts flow control.	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.9 Auto-flow control

If auto-RTS mode is enabled the UART1’s receiver FIFO hardware controls the RTS1 output of the UART1. If the auto-CTS mode is enabled the UART1’s U1TSR hardware will only start transmitting if the CTS1 input signal is asserted.

#### 17.5.9.1 Auto-RTS

The auto-RTS function is enabled by setting the RTSen bit. Auto-RTS data flow control originates in the U1RBR module and is linked to the programmed receiver FIFO trigger level. If auto-RTS is enabled, the data-flow is controlled as follows:

When the receiver FIFO level reaches the programmed trigger level, RTS1 is de-asserted (to a high value). It is possible that the sending UART sends an additional byte after the trigger level is reached (assuming the sending UART has another byte to send) because it might not recognize the de-assertion of RTS1 until after it has begun sending the additional byte. RTS1 is automatically reasserted (to a low value) once the receiver FIFO has reached the previous trigger level. The re-assertion of RTS1 signals to the sending UART to continue transmitting data.

If Auto-RTS mode is disabled, the RTSen bit controls the RTS1 output of the UART1. If Auto-RTS mode is enabled, hardware controls the RTS1 output, and the actual value of RTS1 will be copied in the RTS Control bit of the UART1. As long as Auto-RTS is enabled, the value of the RTS Control bit is read-only for software.

**Example:** Suppose the UART1 operating in '550 mode has trigger level in U1FCR set to 0x2 then if Auto-RTS is enabled the UART1 will de-assert the RTS1 output as soon as the receive FIFO contains 8 bytes (Table 373 on page 436). The RTS1 output will be reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

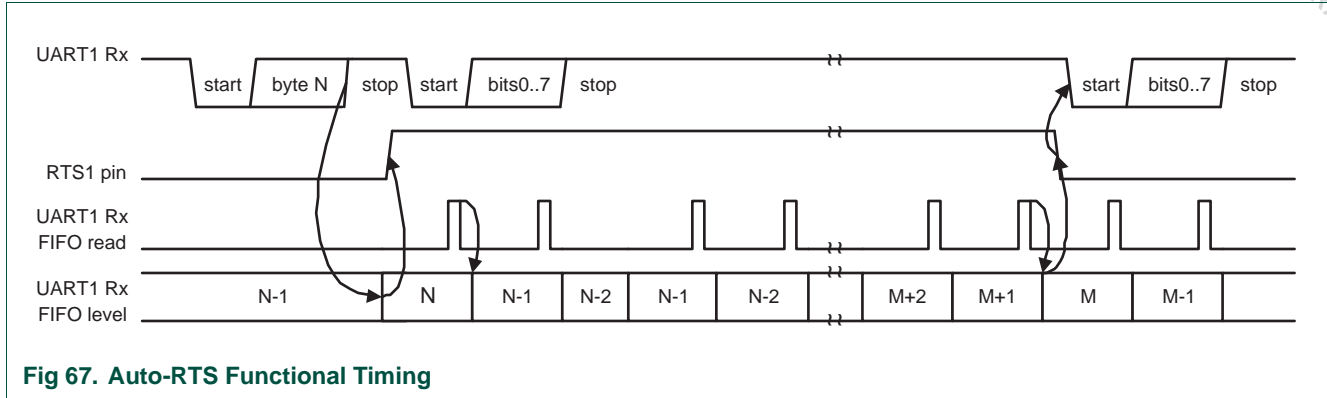


Fig 67. Auto-RTS Functional Timing

### 17.5.9.2 Auto-CTS

The Auto-CTS function is enabled by setting the CTSen bit. If Auto-CTS is enabled the transmitter circuitry in the U1TSR module checks CTS1 input before sending the next data byte. When CTS1 is active (low), the transmitter sends the next byte. To stop the transmitter from sending the following byte, CTS1 must be released before the middle of the last stop bit that is currently being sent. In Auto-CTS mode a change of the CTS1 signal does not trigger a modem status interrupt unless the CTS Interrupt Enable bit is set, Delta CTS bit in the U1MSR will be set though. Table 376 lists the conditions for generating a Modem Status interrupt.

Table 376: Modem status interrupt generation

Enable Modem Status Interrupt (U1ER[3])	CTSen (U1MCR[7])	CTS Interrupt Enable (U1IER[7])	Delta CTS (U1MSR[0])	Delta DCD or Trailing Edge RI or Delta DSR (U1MSR[3] or U1MSR[2] or U1MSR[1])	Modem Status Interrupt
0	x	x	x	x	No
1	0	x	0	0	No
1	0	x	1	x	Yes
1	0	x	x	1	Yes
1	1	0	x	0	No
1	1	0	x	1	Yes
1	1	1	0	0	No
1	1	1	1	x	Yes
1	1	1	x	1	Yes

The auto-CTS function reduces interrupts to the host system. When flow control is enabled, a CTS1 state change does not trigger host interrupts because the device automatically controls its own transmitter. Without Auto-CTS, the transmitter sends any data present in the transmit FIFO and a receiver overrun error can result. Figure 68 illustrates the Auto-CTS functional timing.

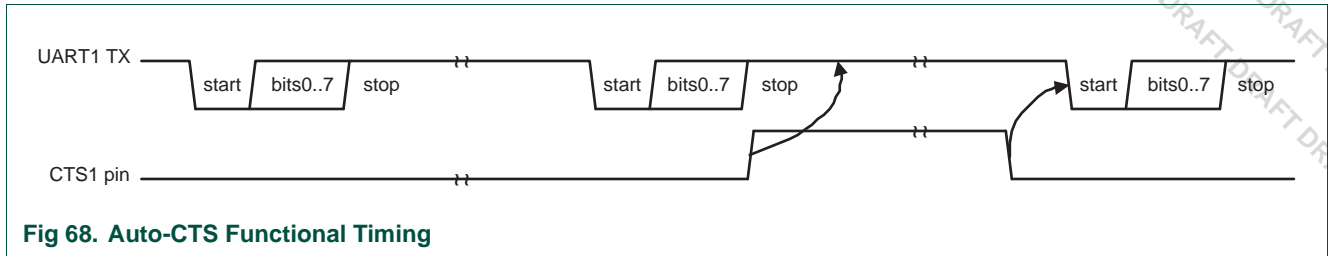


Fig 68. Auto-CTS Functional Timing

While starting transmission of the initial character the CTS1 signal is asserted. Transmission will stall as soon as the pending transmission has completed. The UART will continue transmitting a 1 bit as long as CTS1 is de-asserted (high). As soon as CTS1 gets de-asserted transmission resumes and a start bit is sent followed by the data bits of the next character.

### 17.5.10 UART1 Line Status Register (U1LSR - 0x4001 0014)

The U1LSR is a read-only register that provides status information on the UART1 TX and RX blocks.

Table 377: UART1 Line Status Register (U1LSR - address 0x4001 0014) bit description

Bit	Symbol	Value	Description	Reset Value
0	Receiver Data Ready (RDR)		U1LSR[0] is set when the U1RBR holds an unread character and is cleared when the UART1 RBR FIFO is empty.	0
		0	The UART1 receiver FIFO is empty.	
		1	The UART1 receiver FIFO is not empty.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An U1LSR read clears U1LSR[1]. U1LSR[1] is set when UART1 RSR has a new character assembled and the UART1 RBR FIFO is full. In this case, the UART1 RBR FIFO will not be overwritten and the character in the UART1 RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	Parity Error (PE)		When the parity bit of a received character is in the wrong state, a parity error occurs. An U1LSR read clears U1LSR[2]. Time of parity error detection is dependent on U1FCR[0]. <b>Note:</b> A parity error is associated with the character at the top of the UART1 RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An U1LSR read clears U1LSR[3]. The time of the framing error detection is dependent on U1FCR0. Upon detection of a framing error, the RX will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. <b>Note:</b> A framing error is associated with the character at the top of the UART1 RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	



Table 377: UART1 Line Status Register (U1LSR - address 0x4001 0014) bit description

Bit	Symbol	Value	Description	Reset Value
4	Break Interrupt (BI)		When RXD1 is held in the spacing state (all zeroes) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXD1 goes to marking state (all ones). An U1LSR read clears this status bit. The time of break detection is dependent on U1FCR[0].  <b>Note:</b> The break interrupt is associated with the character at the top of the UART1 RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE)		THRE is set immediately upon detection of an empty UART1 THR and is cleared on a U1THR write.	1
		0	U1THR contains valid data.	
		1	U1THR is empty.	
6	Transmitter Empty (TEMT)		TEMT is set when both U1THR and U1TSR are empty; TEMT is cleared when either the U1TSR or the U1THR contain valid data.	1
		0	U1THR and/or the U1TSR contains valid data.	
		1	U1THR and the U1TSR are empty.	
7	Error in RX FIFO (RXFE)		U1LSR[7] is set when a character with a RX error such as framing error, parity error or break interrupt, is loaded into the U1RBR. This bit is cleared when the U1LSR register is read and there are no subsequent errors in the UART1 FIFO.	0
		0	U1RBR contains no UART1 RX errors or U1FCR[0]=0.	
		1	UART1 RBR contains at least one UART1 RX error.	
31:8	-		Reserved, the value read from a reserved bit is not defined.	NA

### 17.5.11 UART1 Modem Status Register (U1MSR - 0x4001 0018)

The U1MSR is a read-only register that provides status information on the modem input signals. U1MSR[3:0] is cleared on U1MSR read. Note that modem signals have no direct effect on UART1 operation, they facilitate software implementation of modem signal operations.

Table 378: UART1 Modem Status Register (U1MSR - address 0x4001 0018) bit description

Bit	Symbol	Value	Description	Reset Value
0	Delta CTS		Set upon state change of input CTS. Cleared on an U1MSR read.	0
		0	No change detected on modem input, CTS.	
		1	State change detected on modem input, CTS.	
1	Delta DSR		Set upon state change of input DSR. Cleared on an U1MSR read.	0
		0	No change detected on modem input, DSR.	
		1	State change detected on modem input, DSR.	
2	Trailing Edge RI		Set upon low to high transition of input RI. Cleared on an U1MSR read.	0
		0	No change detected on modem input, RI.	
		1	Low-to-high transition detected on RI.	
3	Delta DCD		Set upon state change of input DCD. Cleared on an U1MSR read.	0
		0	No change detected on modem input, DCD.	
		1	State change detected on modem input, DCD.	

**Table 378: UART1 Modem Status Register (U1MSR - address 0x4001 0018) bit description**

Bit	Symbol	Value	Description	Reset Value
4	CTS		Clear To Send State. Complement of input signal CTS. This bit is connected to U1MCR[1] in modem loopback mode.	0
5	DSR		Data Set Ready State. Complement of input signal DSR. This bit is connected to U1MCR[0] in modem loopback mode.	0
6	RI		Ring Indicator State. Complement of input RI. This bit is connected to U1MCR[2] in modem loopback mode.	0
7	DCD		Data Carrier Detect State. Complement of input DCD. This bit is connected to U1MCR[3] in modem loopback mode.	0
31:8	-		Reserved, the value read from a reserved bit is not defined.	NA

### 17.5.12 UART1 Scratch Pad Register (U1SCR - 0x4001 001C)

The U1SCR has no effect on the UART1 operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the U1SCR has occurred.

**Table 379: UART1 Scratch Pad Register (U1SCR - address 0x4001 0014) bit description**

Bit	Symbol	Description	Reset Value
7:0	Pad	A readable, writable byte.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.13 UART1 Auto-baud Control Register (U1ACR - 0x4001 0020)

The UART1 Auto-baud Control Register (U1ACR) controls the process of measuring the incoming clock/data rate for the baud rate generation and can be read and written at user's discretion.

**Table 380: Auto-baud Control Register (U1ACR - address 0x4001 0020) bit description**

Bit	Symbol	Value	Description	Reset value
0	Start		This bit is automatically cleared after auto-baud completion.	0
		0	Auto-baud stop (auto-baud is not running).	
		1	Auto-baud start (auto-baud is running). Auto-baud run bit. This bit is automatically cleared after auto-baud completion.	
1	Mode		Auto-baud mode select bit.	0
		0	Mode 0.	
		1	Mode 1.	
2	AutoRestart	0	No restart	0
		1	Restart in case of time-out (counter restarts at next UART1 Rx falling edge)	0
7:3	-		Reserved. Read value is undefined, only zero should be written.	0
8	ABEOIntClr		End of auto-baud interrupt clear bit (write-only accessible).	0
		0	Writing a 0 has no impact.	
		1	Writing a 1 will clear the corresponding interrupt in the U1IIR.	
9	ABTOIntClr		Auto-baud time-out interrupt clear bit (write-only accessible).	0
		0	Writing a 0 has no impact.	
		1	Writing a 1 will clear the corresponding interrupt in the U1IIR.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	0

### 17.5.14 Auto-baud

The UART1 auto-baud function can be used to measure the incoming baud-rate based on the “AT” protocol (Hayes command). If enabled the auto-baud feature will measure the bit time of the receive data stream and set the divisor latch registers U1DLM and U1DLL accordingly.

**Remark:** the fractional rate divider is not connected during auto-baud operations, and therefore should not be used when the auto-baud feature is needed.

Auto-baud is started by setting the U1ACR Start bit. Auto-baud can be stopped by clearing the U1ACR Start bit. The Start bit will clear once auto-baud has finished and reading the bit will return the status of auto-baud (pending/finished).

Two auto-baud measuring modes are available which can be selected by the U1ACR Mode bit. In mode 0 the baud-rate is measured on two subsequent falling edges of the UART1 Rx pin (the falling edge of the start bit and the falling edge of the least significant bit). In mode 1 the baud-rate is measured between the falling edge and the subsequent rising edge of the UART1 Rx pin (the length of the start bit).

The U1ACR AutoRestart bit can be used to automatically restart baud-rate measurement if a time-out occurs (the rate measurement counter overflows). If this bit is set the rate measurement will restart at the next falling edge of the UART1 Rx pin.

The auto-baud function can generate two interrupts.

- The U1IIR ABTOInt interrupt will get set if the interrupt is enabled (U1IER ABTOIntEn is set and the auto-baud rate measurement counter overflows).
- The U1IIR ABEOInt interrupt will get set if the interrupt is enabled (U1IER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding U1ACR ABTOIntClr and ABEOIntEn bits.

Typically the fractional baud-rate generator is disabled (DIVADDVAL = 0) during auto-baud. However, if the fractional baud-rate generator is enabled (DIVADDVAL > 0), it is going to impact the measuring of UART1 Rx pin baud-rate, but the value of the U1FDR register is not going to be modified after rate measurement. Also, when auto-baud is used, any write to U1DLM and U1DLL registers should be done before U1ACR register write. The minimum and the maximum baud rates supported by UART1 are function of pclk, number of data bits, stop bits and parity bits.

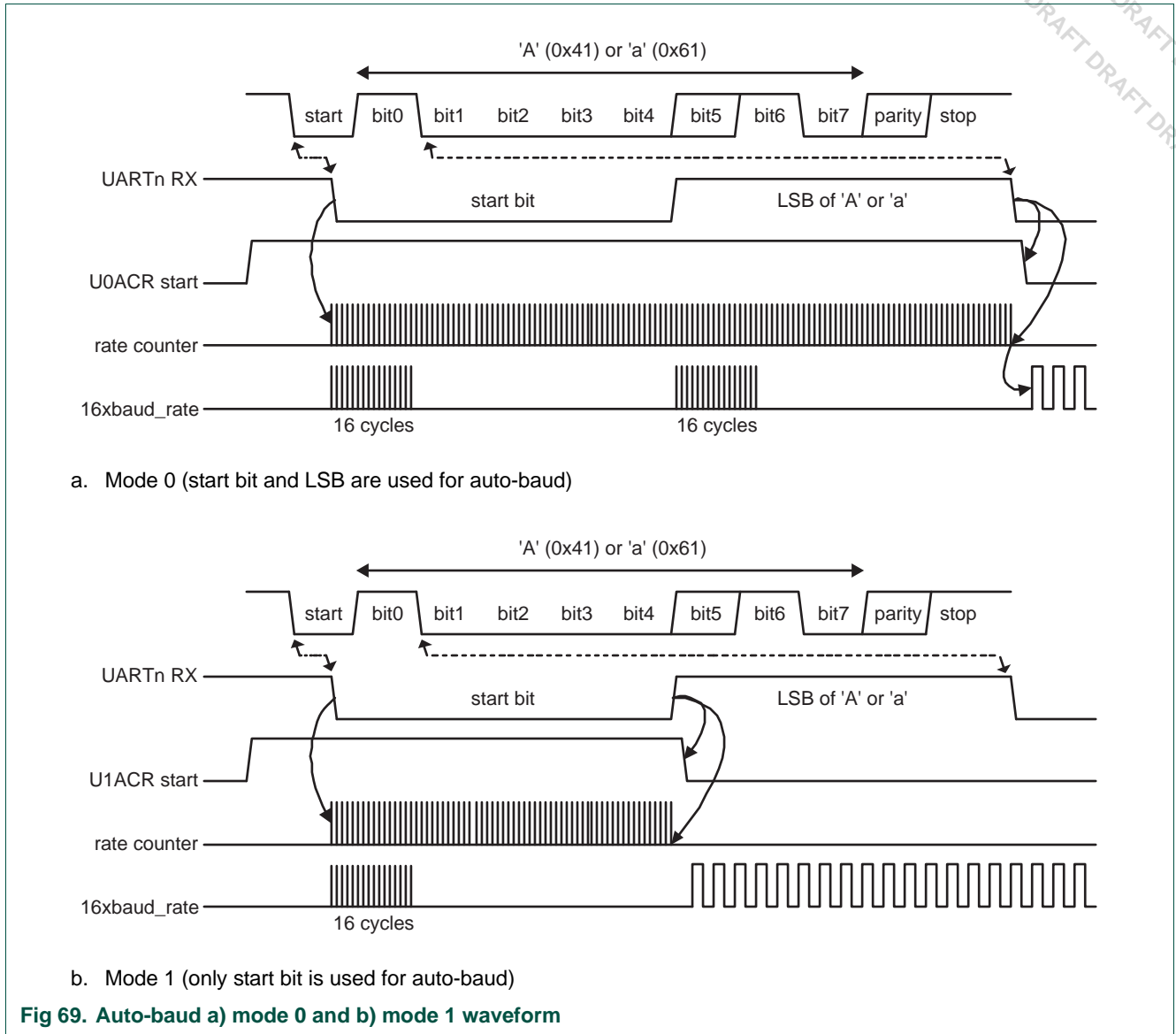
(1)

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART1_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

### 17.5.15 Auto-baud modes

When the software is expecting an “AT” command, it configures the UART1 with the expected character format and sets the U1ACR Start bit. The initial values in the divisor latches U1DLM and U1DLL don't care. Because of the “A” or “a” ASCII coding (“A” = 0x41, “a” = 0x61), the UART1 Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the U1ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On U1ACR Start bit setting, the baud-rate measurement counter is reset and the UART1 U1RSR is reset. The U1RSR baud rate is switch to the highest rate.
2. A falling edge on UART1 Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting pclk cycles optionally pre-scaled by the fractional baud-rate generator.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the (fractional baud-rate pre-scaled) UART1 input clock, guaranteeing the start bit is stored in the U1RSR.
4. During the receipt of the start bit (and the character LSB for mode = 0) the rate counter will continue incrementing with the pre-scaled UART1 input clock (pclk).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UART1 Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UART1 Rx pin.
6. The rate counter is loaded into U1DLM/U1DLL and the baud-rate will be switched to normal operation. After setting the U1DLM/U1DLL the end of auto-baud interrupt U1IIR ABEOInt will be set, if enabled. The U1RSR will now continue receiving the remaining bits of the “A/a” character.



### 17.5.16 UART1 Fractional Divider Register (U1FDR - 0x4001 0028)

The UART1 Fractional Divider Register (U1FDR) controls the clock pre-scaler for the baud rate generation and can be read and written at the user's discretion. This pre-scaler takes the APB clock and generates an output clock according to the specified fractional requirements.

**Important:** If the fractional divider is active (DIVADDDVAL > 0) and DLM = 0, the value of the DLL register must be greater than 2.

**Table 381: UART1 Fractional Divider Register (U1FDR - address 0x4001 0028) bit description**

Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-	-	Reserved. Read value is undefined, only zero should be written.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of UART1 disabled making sure that UART1 is fully software and hardware compatible with UARTs not equipped with this feature.

UART1 baud rate can be calculated as (n = 1):

(2)

$$UART1_{baudrate} = \frac{PCLK}{16 \times (256 \times U1DLM + U1DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Where PCLK is the peripheral clock, U1DLM and U1DLL are the standard UART1 baud rate divider registers, and DIVADDVAL and MULVAL are UART1 fractional baud rate generator specific parameters.

The value of MULVAL and DIVADDVAL should comply to the following conditions:

1.  $1 \leq MULVAL \leq 15$
2.  $0 \leq DIVADDVAL \leq 14$
3.  $DIVADDVAL < MULVAL$

The value of the U1FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the U1FDR register value does not comply to these two requests, then the fractional divider output is undefined. If DIVADDVAL is zero then the fractional divider is disabled, and the clock will not be divided.

### 17.5.16.1 Baud rate calculation

UART1 can operate with or without using the Fractional Divider. In real-life applications it is likely that the desired baud rate can be achieved using several different Fractional Divider settings. The following algorithm illustrates one way of finding a set of DLM, DLL, MULVAL, and DIVADDVAL values. Such set of parameters yields a baud rate with a relative error of less than 1.1% from the desired one.

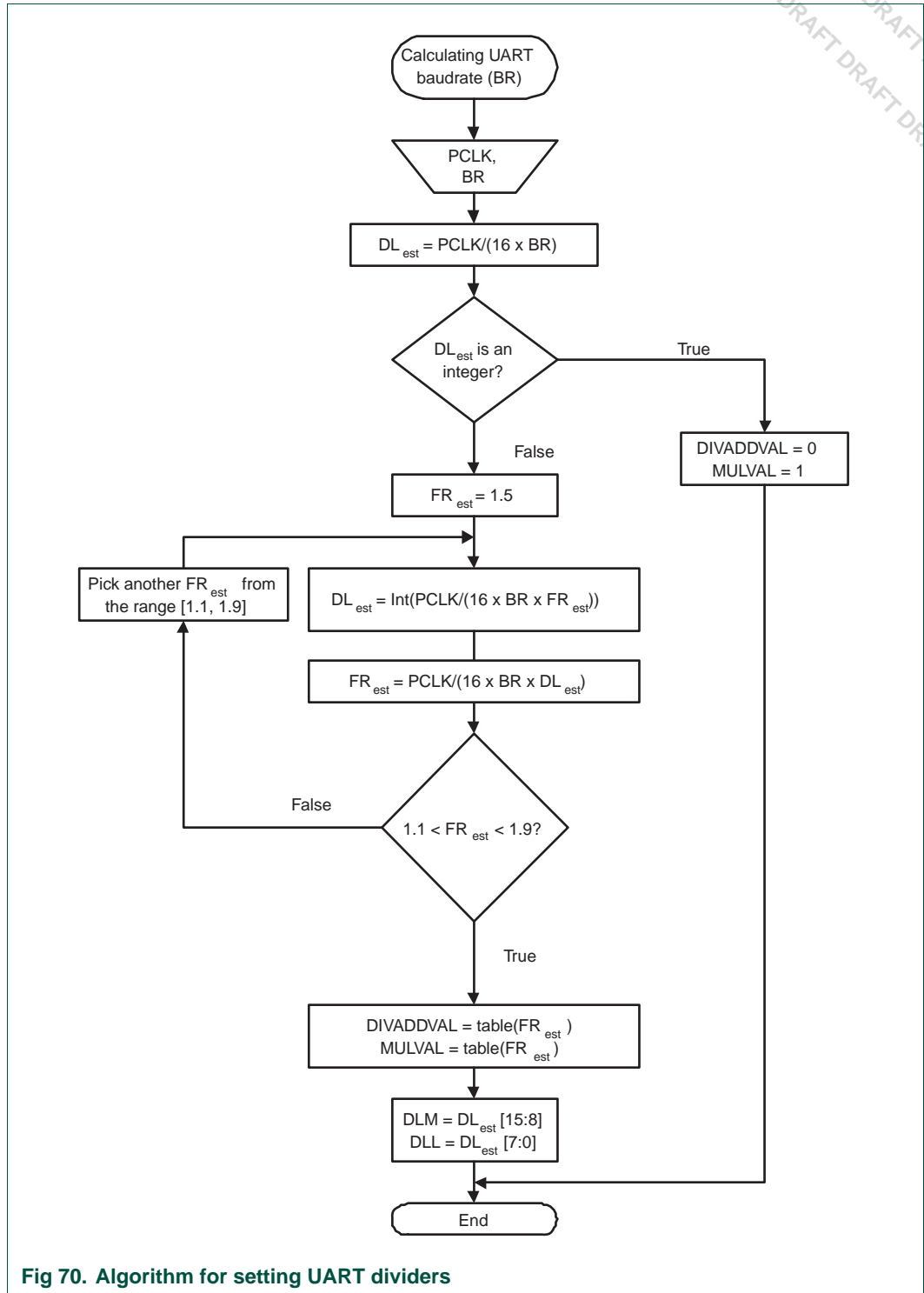


Fig 70. Algorithm for setting UART dividers

**Table 382. Fractional Divider setting look-up table**

FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
1.000	0/1	1.250	1/4	1.500	1/2	1.750	3/4
1.067	1/15	1.267	4/15	1.533	8/15	1.769	10/13
1.071	1/14	1.273	3/11	1.538	7/13	1.778	7/9
1.077	1/13	1.286	2/7	1.545	6/11	1.786	11/14
1.083	1/12	1.300	3/10	1.556	5/9	1.800	4/5
1.091	1/11	1.308	4/13	1.571	4/7	1.818	9/11
1.100	1/10	1.333	1/3	1.583	7/12	1.833	5/6
1.111	1/9	1.357	5/14	1.600	3/5	1.846	11/13
1.125	1/8	1.364	4/11	1.615	8/13	1.857	6/7
1.133	2/15	1.375	3/8	1.625	5/8	1.867	13/15
1.143	1/7	1.385	5/13	1.636	7/11	1.875	7/8
1.154	2/13	1.400	2/5	1.643	9/14	1.889	8/9
1.167	1/6	1.417	5/12	1.667	2/3	1.900	9/10
1.182	2/11	1.429	3/7	1.692	9/13	1.909	10/11
1.200	1/5	1.444	4/9	1.700	7/10	1.917	11/12
1.214	3/14	1.455	5/11	1.714	5/7	1.923	12/13
1.222	2/9	1.462	6/13	1.727	8/11	1.929	13/14
1.231	3/13	1.467	7/15	1.733	11/15	1.933	14/15

**17.5.16.1.1 Example 1: PCLK = 14.7456 MHz, BR = 9600**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 14.7456 \text{ MHz} / (16 \times 9600) = 96$ . Since this  $DL_{est}$  is an integer number,  $DIVADDVAL = 0$ ,  $MULVAL = 1$ ,  $DLM = 0$ , and  $DLL = 96$ .

**17.5.16.1.2 Example 2: PCLK = 12 MHz, BR = 115200**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 12 \text{ MHz} / (16 \times 115200) = 6.51$ . This  $DL_{est}$  is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of  $FR_{est} = 1.5$  a new  $DL_{est} = 4$  is calculated and  $FR_{est}$  is recalculated as  $FR_{est} = 1.628$ . Since  $FR_{est} = 1.628$  is within the specified range of 1.1 and 1.9,  $DIVADDVAL$  and  $MULVAL$  values can be obtained from the attached look-up table.

The closest value for  $FR_{est} = 1.628$  in the look-up [Table 382](#) is  $FR = 1.625$ . It is equivalent to  $DIVADDVAL = 5$  and  $MULVAL = 8$ .

Based on these findings, the suggested UART setup would be:  $DLM = 0$ ,  $DLL = 4$ ,  $DIVADDVAL = 5$ , and  $MULVAL = 8$ . According to [Equation 2](#) the UART rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

**17.5.17 UART1 Transmit Enable Register (U1TER - 0x4001 0030)**

In addition to being equipped with full hardware flow control (auto-cts and auto-rts mechanisms described above), U1TER enables implementation of software flow control, too. When  $TxEn=1$ , UART1 transmitter will keep sending data as long as they are available. As soon as  $TxEn$  becomes 0, UART1 transmission will stop.



[Table 383](#) describes how to use the TxEn bit in order to achieve hardware flow control. However, it is strongly suggested to let UART1 hardware implemented auto flow control features take for this purpose, and limit the scope of TxEn to software flow control.

U1TER enables implementation of software and hardware flow control. When TXEn=1, UART1 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART1 transmission will stop.

**Table 383: UART1 Transmit Enable Register (U1TER - address 0x4001 0030) bit description**

Bit	Symbol	Description	Reset Value
6:0	-	Reserved. Read value is undefined, only zero should be written.	NA
7	TXEN	When this bit is 1, as it is after a Reset, data written to the THR is output on the TXD pin as soon as any preceding data has been sent. If this bit cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again. In other words, a 0 in this bit blocks the transfer of characters from the THR or TX FIFO into the transmit shift register. Software can clear this bit when it detects that the a hardware-handshaking TX-permit signal (CTS) has gone false, or with software handshaking, when it receives an XOFF character (DC3). Software can set this bit again when it detects that the TX-permit signal has gone true, or when it receives an XON (DC1) character.	1
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.18 UART1 RS485 Control register (U1RS485CTRL - 0x4001 004C)

The U1RS485CTRL register controls the configuration of the UART in RS-485/EIA-485 mode.

**Table 384: UART1 RS485 Control register (U1RS485CTRL - address 0x4001 004C) bit description**

Bit	Symbol	Value	Description	Reset value
0	NMMEN	0	RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled.	0
		1	RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.	
1	RXDIS	0	The receiver is enabled.	0
		1	The receiver is disabled.	
2	AADEN	0	Auto Address Detect (AAD) is disabled.	0
		1	Auto Address Detect (AAD) is enabled.	
3	SEL	0	If direction control is enabled (bit DCTRL = 1), pin $\overline{\text{RTS}}$ is used for direction control.	0
		1	If direction control is enabled (bit DCTRL = 1), pin DTR is used for direction control.	
4	DCTRL	0	Disable Auto Direction Control.	0
		1	Enable Auto Direction Control.	
5	OINV		This bit reverses the polarity of the direction control signal on the RTS (or DTR) pin.	0
		0	The direction control pin will be driven to logic '0' when the transmitter has data to be sent. It will be driven to logic '1' after the last bit of data has been transmitted.	
		1	The direction control pin will be driven to logic '1' when the transmitter has data to be sent. It will be driven to logic '0' after the last bit of data has been transmitted.	
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA	

### 17.5.19 UART1 RS-485 Address Match register (U1RS485ADRMATCH - 0x4001 0050)

The U1RS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

**Table 385. UART1 RS-485 Address Match register (U1RS485ADRMATCH - address 0x4001 0050) bit description**

Bit	Symbol	Description	Reset value
7:0	ADRMATCH	Contains the address match value.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.20 UART1 RS-485 Delay value register (U1RS485DLY - 0x4001 0054)

The user may program the 8-bit RS485DLY register with a delay between the last stop bit leaving the TXFIFO and the de-assertion of RTS (or DTR). This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

**Table 386. UART1 RS-485 Delay value register (U1RS485DLY - address 0x4001 0054) bit description**

Bit	Symbol	Description	Reset value
7:0	DLY	Contains the direction control (RTS or DTR) delay value. This register works in conjunction with an 8-bit counter.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 17.5.21 RS-485/EIA-485 modes of operation

The RS-485/EIA-485 feature allows the UART to be configured as an addressable slave. The addressable slave is one of multiple slaves controlled by a single master.

The UART master transmitter will identify an address character by setting the parity (9th) bit to '1'. For data characters, the parity bit is set to '0'.

Each UART slave receiver can be assigned a unique address. The slave can be programmed to either manually or automatically reject data following an address which is not theirs.

#### RS-485/EIA-485 Normal Multidrop Mode (NMM)

Setting the RS485CTRL bit 0 enables this mode. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received data bytes will be ignored and will not be stored in the RXFIFO. When an address byte is detected (parity bit = '1') it will be placed into the RXFIFO and an Rx Data Ready Interrupt will be generated. The processor can then read the address byte and decide whether or not to enable the receiver to accept the following data.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all received bytes will be accepted and stored in the RXFIFO regardless of whether they are data or address. When an address character is received a parity error interrupt will be generated and the processor can decide whether or not to disable the receiver.

#### RS-485/EIA-485 Auto Address Detection (AAD) mode

When both RS485CTRL register bits 0 (9-bit mode enable) and 2 (AAD mode enable) are set, the UART is in auto address detect mode.

In this mode, the receiver will compare any address byte received (parity = '1') to the 8-bit value programmed into the RS485ADRMATCH register.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received byte will be discarded if it is either a data byte OR an address byte which fails to match the RS485ADRMATCH value.

When a matching address character is detected it will be pushed onto the RXFIFO along with the parity bit, and the receiver will be automatically enabled (RS485CTRL bit 1 will be cleared by hardware). The receiver will also generate a Rx Data Ready Interrupt.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all bytes received will be accepted and stored in the RXFIFO until an address byte which does not match the RS485ADRMATCH value is received. When this occurs, the receiver will be automatically disabled in hardware (RS485CTRL bit 1 will be set), The received non-matching address character will not be stored in the RXFIFO.

### RS-485/EIA-485 Auto Direction Control

RS485/EIA-485 Mode includes the option of allowing the transmitter to automatically control the state of either the  $\overline{\text{RTS}}$  pin or the  $\overline{\text{DTR}}$  pin as a direction control output signal.

Setting RS485CTRL bit 4 = '1' enables this feature.

Direction control, if enabled, will use the  $\overline{\text{RTS}}$  pin when RS485CTRL bit 3 = '0'. It will use the  $\overline{\text{DTR}}$  pin when RS485CTRL bit 3 = '1'.

When Auto Direction Control is enabled, the selected pin will be asserted (driven low) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven high) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

The RS485CTRL bit 4 takes precedence over all other mechanisms controlling  $\overline{\text{RTS}}$  (or  $\overline{\text{DTR}}$ ) with the exception of loopback mode.

### RS485/EIA-485 driver delay time

The driver delay time is the delay between the last stop bit leaving the TXFIFO and the de-assertion of RTS (or DTR). This delay time can be programmed in the 8-bit RS485DLY register. The delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

### RS485/EIA-485 output inversion

The polarity of the direction control signal on the  $\overline{\text{RTS}}$  (or  $\overline{\text{DTR}}$ ) pins can be reversed by programming bit 5 in the U1RS485CTRL register. When this bit is set, the direction control pin will be driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin will be driven to logic 0 after the last bit of data has been transmitted.

### 18.1 How to read this chapter

---

Most LPC178x/177x family devices include 5 UARTs. A few devices implement only 4 UARTs. Refer to a specific device data sheet for details. UARTs 0, 2, and 3 are essentially the same as UART1, but without modem/flow control signals. UART4, described in the next chapter, adds a synchronous mode and a Smart Card mode.

### 18.2 Basic configuration

---

The UART0/2/3 peripherals are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bits PCUART0/2/3.  
**Remark:** On reset, UART0 is enabled (PCUART0 = 1), and UART2/3 are disabled (PCUART2/3 = 0).
2. Peripheral clock: These UARTs operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Baud rate: In register U0/2/3LCR ([Table 397](#)), set bit DLAB = 1. This enables access to registers DLL ([Table 391](#)) and DLM ([Table 392](#)) for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register ([Table 401](#)).
4. UART FIFO: Use bit FIFO enable (bit 0) in register U0/2/3FCR ([Table 396](#)) to enable the FIFOs.
5. Pins: Select UART pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).  
**Remark:** UART receive pins should not have pull-down resistors enabled.
6. Interrupts: To enable UART interrupts set bit DLAB = 0 in register U0/2/3LCR ([Table 397](#)). This enables access to U0/2/3IER ([Table 393](#)). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
7. DMA: UART0/2/3 transmit and receive functions can operate with the GPDMA controller (see [Table 655](#)).

## 18.3 Features

- Data sizes of 5, 6, 7, and 8 bits.
- Parity generation and checking: odd, even mark, space or none.
- One or two stop bits.
- 16 byte Receive and Transmit FIFOs.
- Built-in baud rate generator, including a fractional rate divider for great versatility.
- Supports DMA for both transmit and receive.
- Auto-baud capability
- Break generation and detection.
- Multiprocessor addressing mode.
- Support for software flow control.
- RS-485/EIA-485 support.

## 18.4 Architecture

The architecture of the UARTs 0, 2, and 3 are shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART.

The UARTn receiver block, UnRX, monitors the serial input line, RXDn, for valid input. The UARTn RX Shift Register (UnRSR) accepts valid characters via RXDn. After a valid character is assembled in the UnRSR, it is passed to the UARTn RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UARTn transmitter block, UnTX, accepts data written by the CPU or host and buffers the data in the UARTn TX Holding Register FIFO (UnTHR). The UARTn TX Shift Register (UnTSR) reads the data stored in the UnTHR and assembles the data to transmit via the serial output pin, TXDn.

The UARTn Baud Rate Generator block, UnBRG, generates the timing enables used by the UARTn TX block. The UnBRG clock input source is the APB clock (PCLK). The main clock is divided down per the divisor specified in the UnDLL and UnDLM registers. This divided down clock is the 16x oversample clock.

The interrupt interface contains registers UnIER and UnIIR. The interrupt interface receives several one clock wide enables from the UnTX and UnRX blocks.

Status information from the UnTX and UnRX is stored in the UnLSR. Control information for the UnTX and UnRX is stored in the UnLCR.

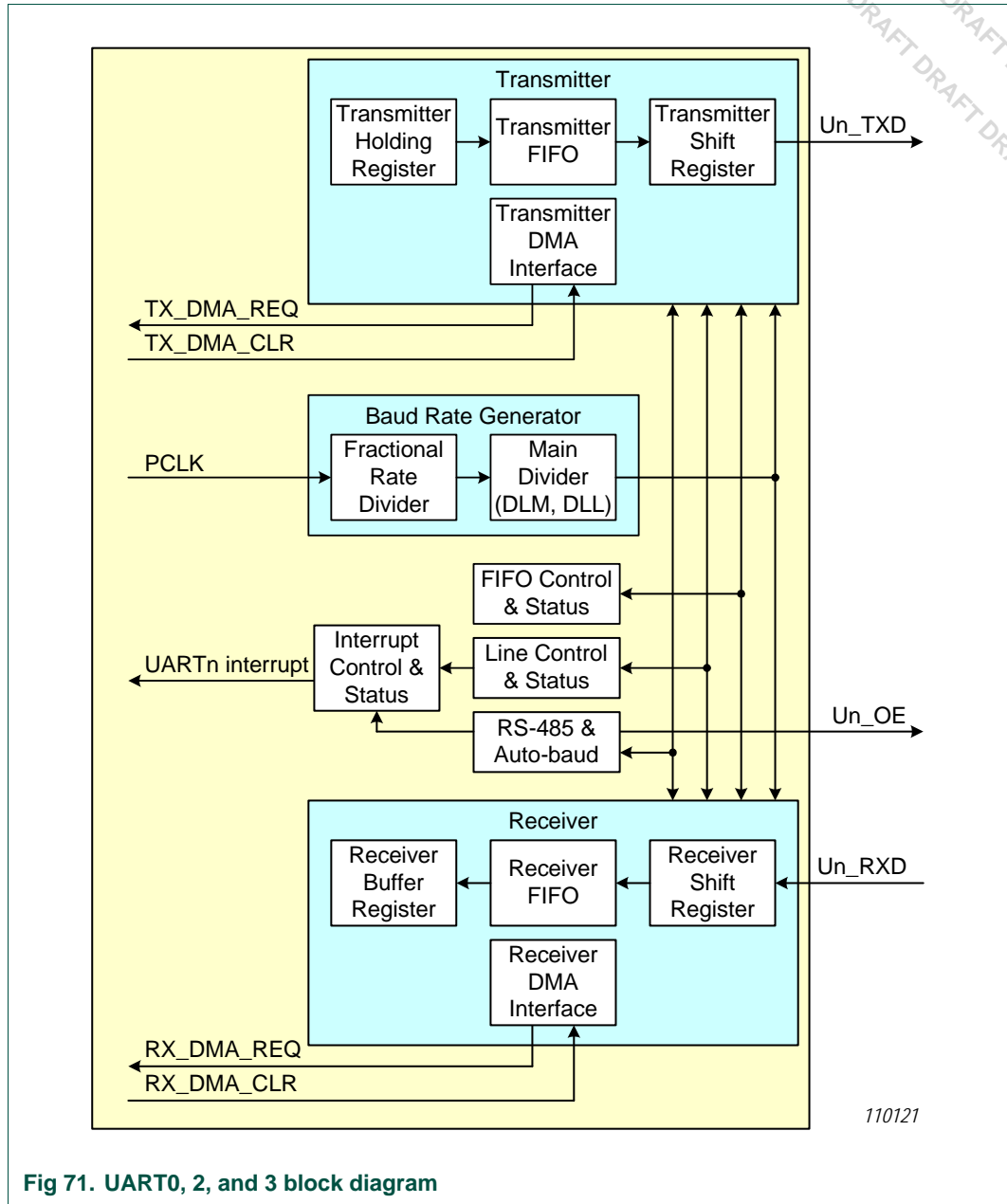


Fig 71. UART0, 2, and 3 block diagram

### 18.5 Pin description

Table 387: UARTn Pin description

Pin	Type	Description
U0_RXD, U2_RXD, U3_RXD	Input	<b>Serial Input.</b> Serial receive data.
U0_TXD, U2_TXD, U3_TXD	Output	<b>Serial Output.</b> Serial transmit data.
U0_OE, U2_OE, U3_OE	Output	<b>Output Enable.</b> RS-485/EIA-485 output enable.

## 18.6 Register description

Each UART contains registers as shown in [Table 388](#). The Divisor Latch Access Bit (DLAB) is contained in UnLCR7 and enables access to the Divisor Latches.

**Table 388. UART0/2/3 Register Map**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	UARTn Register Name & Address	Table
RBR (DLAB =0)	Receiver Buffer Register. Contains the next received character to be read.	RO	NA	U0RBR - 0x4000 C000 U2RBR - 0x4009 8000 U3RBR - 0x4009 C000	<a href="#">389</a>
THR (DLAB =0)	Transmit Holding Register. The next character to be transmitted is written here.	WO	NA	U0THR - 0x4000 C000 U2THR - 0x4009 8000 U3THR - 0x4009 C000	<a href="#">390</a>
DLL (DLAB =1)	Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x01	U0DLL - 0x4000 C000 U2DLL - 0x4009 8000 U3DLL - 0x4009 C000	<a href="#">391</a>
DLM (DLAB =1)	Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0	U0DLM - 0x4000 C004 U2DLM - 0x4009 8004 U3DLM - 0x4009 C004	<a href="#">392</a>
IER (DLAB =0)	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART interrupts.	R/W	0	U0IER - 0x4000 C004 U2IER - 0x4009 8004 U3IER - 0x4009 C004	<a href="#">393</a>
IIR	Interrupt ID Register. Identifies which interrupt(s) are pending.	RO	0x01	U0IIR - 0x4000 C008 U2IIR - 0x4009 8008 U3IIR - 0x4009 C008	<a href="#">394</a>
FCR	FIFO Control Register. Controls UART FIFO usage and modes.	WO	0	U0FCR - 0x4000 C008 U2FCR - 0x4009 8008 U3FCR - 0x4009 C008	<a href="#">396</a>
LCR	Line Control Register. Contains controls for frame formatting and break generation.	R/W	0	U0LCR - 0x4000 C00C U2LCR - 0x4009 800C U3LCR - 0x4009 C00C	<a href="#">397</a>
LSR	Line Status Register. Contains flags for transmit and receive status, including line errors.	RO	0x60	U0LSR - 0x4000 C014 U2LSR - 0x4009 8014 U3LSR - 0x4009 C014	<a href="#">398</a>
SCR	Scratch Pad Register. 8-bit temporary storage for software.	R/W	0	U0SCR - 0x4000 C01C U2SCR - 0x4009 801C U3SCR - 0x4009 C01C	<a href="#">399</a>
ACR	Auto-baud Control Register. Contains controls for the auto-baud feature.	R/W	0	U0ACR - 0x4000 C020 U2ACR - 0x4009 8020 U3ACR - 0x4009 C020	<a href="#">400</a>
FDR	Fractional Divider Register. Generates a clock input for the baud rate divider.	R/W	0x10	U0FDR - 0x4000 C028 U2FDR - 0x4009 8028 U3FDR - 0x4009 C028	<a href="#">401</a>
TER	Transmit Enable Register. Turns off UART transmitter for use with software flow control.	R/W	0x80	U0TER - 0x4000 C030 U2TER - 0x4009 8030 U3TER - 0x4009 C030	<a href="#">403</a>

Table 388. UART0/2/3 Register Map

Generic Name	Description	Access	Reset value <sup>[1]</sup>	UARTn Register Name & Address	Table
RS485CTRL	RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes.	R/W	0	U0RS485CTRL - 0x4000 C04C U2RS485CTRL - 0x4009 804C U3RS485CTRL - 0x4009 C04C	<a href="#">404</a>
ADRMATCH	RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode.	R/W	0	U0ADRMATCH - 0x4000 C050 U2ADRMATCH - 0x4009 8050 U3ADRMATCH - 0x4009 C050	<a href="#">405</a>
RS485DLY	RS-485/EIA-485 direction control delay.	R/W	0	U0RS485DLY - 0x4000 C054 U2RS485DLY - 0x4009 8054 U3RS485DLY - 0x4009 C054	<a href="#">406</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 18.6.1 UARTn Receiver Buffer Register (U0RBR - 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 0x4009 C000 when DLAB = 0)

The UnRBR is the top byte of the UARTn Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in LCR must be zero in order to access the UnRBR. The UnRBR is always read-only.

Since PE, FE and BI bits correspond to the byte sitting on the top of the RBR FIFO (i.e. the one that will be read in the next read from the RBR), the right approach for fetching the valid pair of received byte and its status bits is first to read the content of the U0LSR register, and then to read a byte from the UnRBR.

Table 389: UARTn Receiver Buffer Register (U0RBR - address 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 04009 C000 when DLAB = 0) bit description

Bit	Symbol	Description	Reset Value
7:0	RBR	The UARTn Receiver Buffer Register contains the oldest received byte in the UARTn Rx FIFO.	Undefined
31:8	-	Reserved, the value read from a reserved bit is not defined.	NA

### 18.6.2 UARTn Transmit Holding Register (U0THR - 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0)

The UnTHR is the top byte of the UARTn TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

The Divisor Latch Access Bit (DLAB) in UnLCR must be zero in order to access the UnTHR. The UnTHR is always write-only.



**Table 390: UARTn Transmit Holding Register (U0THR - address 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0) bit description**

Bit	Symbol	Description	Reset Value
7:0	THR	Writing to the UARTn Transmit Holding Register causes the data to be stored in the UARTn transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.3 UARTn Divisor Latch LSB register (U0DLL - 0x4000 C000, U2DLL - 0x4009 8000, U3DLL - 0x4009 C000 when DLAB = 1) and UARTn Divisor Latch MSB register (U0DLM - 0x4000 C004, U2DLM - 0x4009 8004, U3DLM - 0x4009 C004 when DLAB = 1)

The UARTn Divisor Latch is part of the UARTn Baud Rate Generator and holds the value used, along with the Fractional Divider, to divide the APB clock (PCLK) in order to produce the baud rate clock, which must be 16× the desired baud rate. The UnDLL and UnDLM registers together form a 16-bit divisor where UnDLL contains the lower 8 bits of the divisor and UnDLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed. The Divisor Latch Access Bit (DLAB) in UnLCR must be one in order to access the UARTn Divisor Latches. Details on how to select the right value for UnDLL and UnDLM can be found later in this chapter, see [Section 18.6.11](#).

**Table 391: UARTn Divisor Latch LSB register (U0DLL - address 0x4000 C000, U2DLL - 0x4009 8000, U3DLL - 0x4009 C000 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLLSB	The UARTn Divisor Latch LSB Register, along with the UnDLM register, determines the baud rate of the UARTn.	0x01
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

**Table 392: UARTn Divisor Latch MSB register (U0DLM - address 0x4000 C004, U2DLM - 0x4009 8004, U3DLM - 0x4009 C004 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLMSB	The UARTn Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UARTn.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.4 UARTn Interrupt Enable Register (U0IER - 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0)

The UnIER is used to enable the three UARTn interrupt sources.

**Table 393: UARTn Interrupt Enable Register (U0IER - address 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0) bit description**

Bit	Symbol	Value	Description	Reset Value
0	RBR Interrupt Enable		Enables the Receive Data Available interrupt for UARTn. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	

**Table 393: UARTn Interrupt Enable Register (U0IER - address 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0) bit description**

Bit	Symbol	Value	Description	Reset Value
1	THRE Interrupt Enable		Enables the THRE interrupt for UARTn. The status of this can be read from UnLSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Status Interrupt Enable		Enables the UARTn RX line status interrupts. The status of this interrupt can be read from UnLSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
7:3	-		Reserved. Read value is undefined, only zero should be written.	NA
8	ABEOIntEn		Enables the end of auto-baud interrupt.	0
		0	Disable end of auto-baud Interrupt.	
		1	Enable end of auto-baud Interrupt.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable auto-baud time-out Interrupt.	
		1	Enable auto-baud time-out Interrupt.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.5 UARTn Interrupt Identification Register (U0IIR - 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008)

The UnIIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an UnIIR access. If an interrupt occurs during an UnIIR access, the interrupt is recorded for the next UnIIR access.

**Table 394: UARTn Interrupt Identification Register (U0IIR - address 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	IntStatus		Interrupt status. Note that UnIIR[0] is active low. The pending interrupt can be determined by evaluating UnIIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	IntId		Interrupt identification. UnIER[3:1] identifies an interrupt corresponding to the UARTn Rx or TX FIFO. All other combinations of UnIER[3:1] not listed below are reserved (000,100,101,111).	0
		011	1 - Receive Line Status (RLS).	
		010	2a - Receive Data Available (RDA).	
		110	2b - Character Time-out Indicator (CTI).	
		001	3 - THRE Interrupt	
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	FIFO Enable		Copies of UnFCR[0].	0

**Table 394: UARTn Interrupt Identification Register (U0IIR - address 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

Bit UnIIR[9:8] are set by the auto-baud function and signal a time-out or end of auto-baud condition. The auto-baud interrupt conditions are cleared by setting the corresponding Clear bits in the Auto-baud Control Register.

If the IntStatus bit is 1 no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in [Table 395](#). Given the status of UnIIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The UnIIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UARTn RLS interrupt (UnIIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UARTn Rx input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UARTn Rx error condition that set the interrupt can be observed via UnLSR[4:1]. The interrupt is cleared upon an UnLSR read.

The UARTn RDA interrupt (UnIIR[3:1] = 010) shares the second level priority with the CTI interrupt (UnIIR[3:1] = 110). The RDA is activated when the UARTn Rx FIFO reaches the trigger level defined in UnFCR[7:6] and is reset when the UARTn Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (UnIIR[3:1] = 110) is a second level interrupt and is set when the UARTn Rx FIFO contains at least one character and no UARTn Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UARTn Rx FIFO activity (read or write of UARTn RSR) will clear the interrupt. This interrupt is intended to flush the UARTn RBR after a message has been received that is not a multiple of the trigger level size. For example, if a peripheral wished to send a 105 character message and the trigger level was 10 characters, the CPU would receive 10 RDA interrupts resulting in the transfer of 100 characters and 1 to 5 CTI interrupts (depending on the service routine) resulting in the transfer of the remaining 5 characters.

**Table 395: UARTn Interrupt Handling**

U0IIR[3:0] value <sup>[1]</sup>	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE <sup>[2]</sup> or PE <sup>[2]</sup> or FE <sup>[2]</sup> or BI <sup>[2]</sup>	UnLSR Read <sup>[2]</sup>

Table 395: UARTn Interrupt Handling

U0IIR[3:0] value <sup>[1]</sup>	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (UnFCR0=1)	UnRBR Read <sup>[3]</sup> or UARTn FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).  The exact time will be: [(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs	UnRBR Read <sup>[3]</sup>
0010	Third	THRE	THRE <sup>[2]</sup>	UnIIR Read (if source of interrupt) or THR write <sup>[4]</sup>

[1] Values "0000", "0011", "0101", "0111", "1000", "1001", "1010", "1011", "1101", "1110", "1111" are reserved.

[2] For details see [Section 18.6.8 "UARTn Line Status Register \(U0LSR - 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014\)"](#)

[3] For details see [Section 18.6.1 "UARTn Receiver Buffer Register \(U0RBR - 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 0x4009 C000 when DLAB = 0\)"](#)

[4] For details see [Section 18.6.5 "UARTn Interrupt Identification Register \(U0IIR - 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008\)"](#) and [Section 18.6.2 "UARTn Transmit Holding Register \(U0THR - 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0\)"](#)

The UARTn THRE interrupt (UnIIR[3:1] = 001) is a third level interrupt and is activated when the UARTn THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UARTn THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE = 1 and there have not been at least two characters in the UnTHR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to UnTHR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UARTn THR FIFO has held two or more characters at one time and currently, the UnTHR is empty. The THRE interrupt is reset when a UnTHR write occurs or a read of the UnIIR occurs and the THRE is the highest interrupt (UnIIR[3:1] = 001).

### 18.6.6 UARTn FIFO Control Register (U0FCR - 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4009 C008)

The write-only UnFCR controls the operation of the UARTn Rx and TX FIFOs.

Table 396: UARTn FIFO Control Register (U0FCR - address 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4007 C008) bit description

Bit	Symbol	Value	Description	Reset Value
0	FIFO Enable	0	UARTn FIFOs are disabled. Must not be used in the application.	0
		1	Active high enable for both UARTn Rx and TX FIFOs and UnFCR[7:1] access. This bit must be set for proper UART operation. Any transition on this bit will automatically clear the related UART FIFOs.	
1	RX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[1] will clear all bytes in UARTn Rx FIFO, reset the pointer logic. This bit is self-clearing.	

**Table 396: UARTn FIFO Control Register (U0FCR - address 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4007 C008) bit description**

Bit	Symbol	Value	Description	Reset Value
2	TX FIFO Reset	0	No impact on either of UARTn FIFOs.	0
		1	Writing a logic 1 to UnFCR[2] will clear all bytes in UARTn TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	DMA Mode Select		When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode. See <a href="#">Section 18.6.6.1</a> .	0
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UARTn FIFO characters must be written before an interrupt or DMA request is activated.	0
		00	Trigger level 0 (1 character or 0x01)	
		01	Trigger level 1 (4 characters or 0x04)	
		10	Trigger level 2 (8 characters or 0x08)	
		11	Trigger level 3 (14 characters or 0x0E)	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.6.1 DMA Operation

The user can optionally operate the UART transmit and/or receive using DMA. The DMA mode is determined by the DMA Mode Select bit in the FCR register. This bit only has an affect when the FIFOs are enabled via the FIFO Enable bit in the FCR register.

#### UART receiver DMA

In DMA mode, the receiver DMA request is asserted on the event of the receiver FIFO level becoming equal to or greater than trigger level, or if a character timeout occurs. See the description of the RX Trigger Level above. The receiver DMA request is cleared by the DMA controller.

#### UART transmitter DMA

In DMA mode, the transmitter DMA request is asserted on the event of the transmitter FIFO transitioning to not full. The transmitter DMA request is cleared by the DMA controller.

### 18.6.7 UARTn Line Control Register (U0LCR - 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C)

The UnLCR determines the format of the data character that is to be transmitted or received.

**Table 397: UARTn Line Control Register (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Word Length Select	00	5-bit character length	0
		01	6-bit character length	
		10	7-bit character length	
		11	8-bit character length	

**Table 397: UARTn Line Control Register (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) bit description**

Bit	Symbol	Value	Description	Reset Value
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if UnLCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	0
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		10	Forced "1" stick parity.	
		11	Forced "0" stick parity.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UARTn TXD is forced to logic 0 when UnLCR[6] is active high.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.8 UARTn Line Status Register (U0LSR - 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014)

The UnLSR is a read-only register that provides status information on the UARTn TX and RX blocks.

**Table 398: UARTn Line Status Register (U0LSR - address 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) bit description**

Bit	Symbol	Value	Description	Reset Value
0	Receiver Data Ready (RDR)		UnLSR[0] is set when the UnRBR holds an unread character and is cleared when the UARTn RBR FIFO is empty.	0
		0	The UARTn receiver FIFO is empty.	
		1	The UARTn receiver FIFO is not empty.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An UnLSR read clears UnLSR[1]. UnLSR[1] is set when UARTn RSR has a new character assembled and the UARTn RBR FIFO is full. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	Parity Error (PE)		When the parity bit of a received character is in the wrong state, a parity error occurs. An UnLSR read clears UnLSR[2]. Time of parity error detection is dependent on UnFCR[0]. <b>Note:</b> A parity error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	

**Table 398: UARTn Line Status Register (U0LSR - address 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) bit description**

Bit	Symbol	Value	Description	Reset Value
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An UnLSR read clears UnLSR[3]. The time of the framing error detection is dependent on UnFCR[0]. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. <b>Note:</b> A framing error is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXDn is held in the spacing state (all zeroes) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all ones). An UnLSR read clears this status bit. The time of break detection is dependent on UnFCR[0]. <b>Note:</b> The break interrupt is associated with the character at the top of the UARTn RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE))		THRE is set immediately upon detection of an empty UARTn THR and is cleared on a UnTHR write.	1
		0	UnTHR contains valid data.	
		1	UnTHR is empty.	
6	Transmitter Empty (TEMT)		TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when either the UnTSR or the UnTHR contain valid data.	1
		0	UnTHR and/or the UnTSR contains valid data.	
		1	UnTHR and the UnTSR are empty.	
7	Error in RX FIFO (RXFE)		UnLSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the UnRBR. This bit is cleared when the UnLSR register is read and there are no subsequent errors in the UARTn FIFO.	0
		0	UnRBR contains no UARTn RX errors or UnFCR[0]=0.	
		1	UARTn RBR contains at least one UARTn RX error.	
31:8	-		Reserved. The value read from a reserved bit is not defined.	NA

### 18.6.9 UARTn Scratch Pad Register (U0SCR - 0x4000 C01C, U2SCR - 0x4009 801C U3SCR - 0x4009 C01C)

The UnSCR has no effect on the UARTn operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the UnSCR has occurred.

**Table 399: UARTn Scratch Pad Register (U0SCR - address 0x4000 C01C, U2SCR - 0x4009 801C, U3SCR - 0x4009 C01C) bit description**

Bit	Symbol	Description	Reset Value
7:0	Pad	A readable, writable byte.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.10 UARTn Auto-baud Control Register (U0ACR - 0x4000 C020, U2ACR - 0x4009 8020, U3ACR - 0x4009 C020)

The UARTn Auto-baud Control Register (UnACR) controls the process of measuring the incoming clock/data rate for the baud rate generation and can be read and written at user's discretion.

**Table 400: UARTn Auto-baud Control Register (U0ACR - address 0x4000 C020, U2ACR - 0x4009 8020, U3ACR - 0x4009 C020) bit description**

Bit	Symbol	Value	Description	Reset value
0	Start		This bit is automatically cleared after auto-baud completion.	0
		0	Auto-baud stop (auto-baud is not running).	
		1	Auto-baud start (auto-baud is running). Auto-baud run bit. This bit is automatically cleared after auto-baud completion.	
1	Mode		Auto-baud mode select bit.	0
		0	Mode 0.	
		1	Mode 1.	
2	AutoRestart	0	No restart.	0
		1	Restart in case of time-out (counter restarts at next UARTn Rx falling edge)	0
7:3	-		Reserved. Read value is undefined, only zero should be written.	NA
8	ABEOIntClr		End of auto-baud interrupt clear bit (write-only accessible). Writing a 1 will clear the corresponding interrupt in the UnIIR. Writing a 0 has no impact.	0
9	ABTOIntClr		Auto-baud time-out interrupt clear bit (write-only accessible). Writing a 1 will clear the corresponding interrupt in the UnIIR. Writing a 0 has no impact.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

#### 18.6.10.1 Auto-baud

The UARTn auto-baud function can be used to measure the incoming baud-rate based on the "AT" protocol (Hayes command). If enabled the auto-baud feature will measure the bit time of the receive data stream and set the divisor latch registers UnDLM and UnDLL accordingly.

**Remark:** the fractional rate divider is not connected during auto-baud operations, and therefore should not be used when the auto-baud feature is needed.

Auto-baud is started by setting the UnACR Start bit. Auto-baud can be stopped by clearing the UnACR Start bit. The Start bit will clear once auto-baud has finished and reading the bit will return the status of auto-baud (pending/finished).

Two auto-baud measuring modes are available which can be selected by the UnACR Mode bit. In mode 0 the baud-rate is measured on two subsequent falling edges of the UARTn Rx pin (the falling edge of the start bit and the falling edge of the least significant bit). In mode 1 the baud-rate is measured between the falling edge and the subsequent rising edge of the UARTn Rx pin (the length of the start bit).

The UnACR AutoRestart bit can be used to automatically restart baud-rate measurement if a time-out occurs (the rate measurement counter overflows). If this bit is set the rate measurement will restart at the next falling edge of the UARTn Rx pin.

The auto-baud function can generate two interrupts.



- The UnIIR ABTOInt interrupt will get set if the interrupt is enabled (UnIER ABTOIntEn is set and the auto-baud rate measurement counter overflows).
- The UnIIR ABEOInt interrupt will get set if the interrupt is enabled (UnIER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding UnACR ABTOIntClr and ABEOIntEn bits.

Typically the fractional baud-rate generator is disabled (DIVADDVAL = 0) during auto-baud. However, if the fractional baud-rate generator is enabled (DIVADDVAL > 0), it is going to impact the measuring of UARTn Rx pin baud-rate, but the value of the UnFDR register is not going to be modified after rate measurement. Also, when auto-baud is used, any write to UnDLM and UnDLL registers should be done before UnACR register write. The minimum and the maximum baud rates supported by UARTn are function of pclk, number of data bits, stop bits and parity bits.

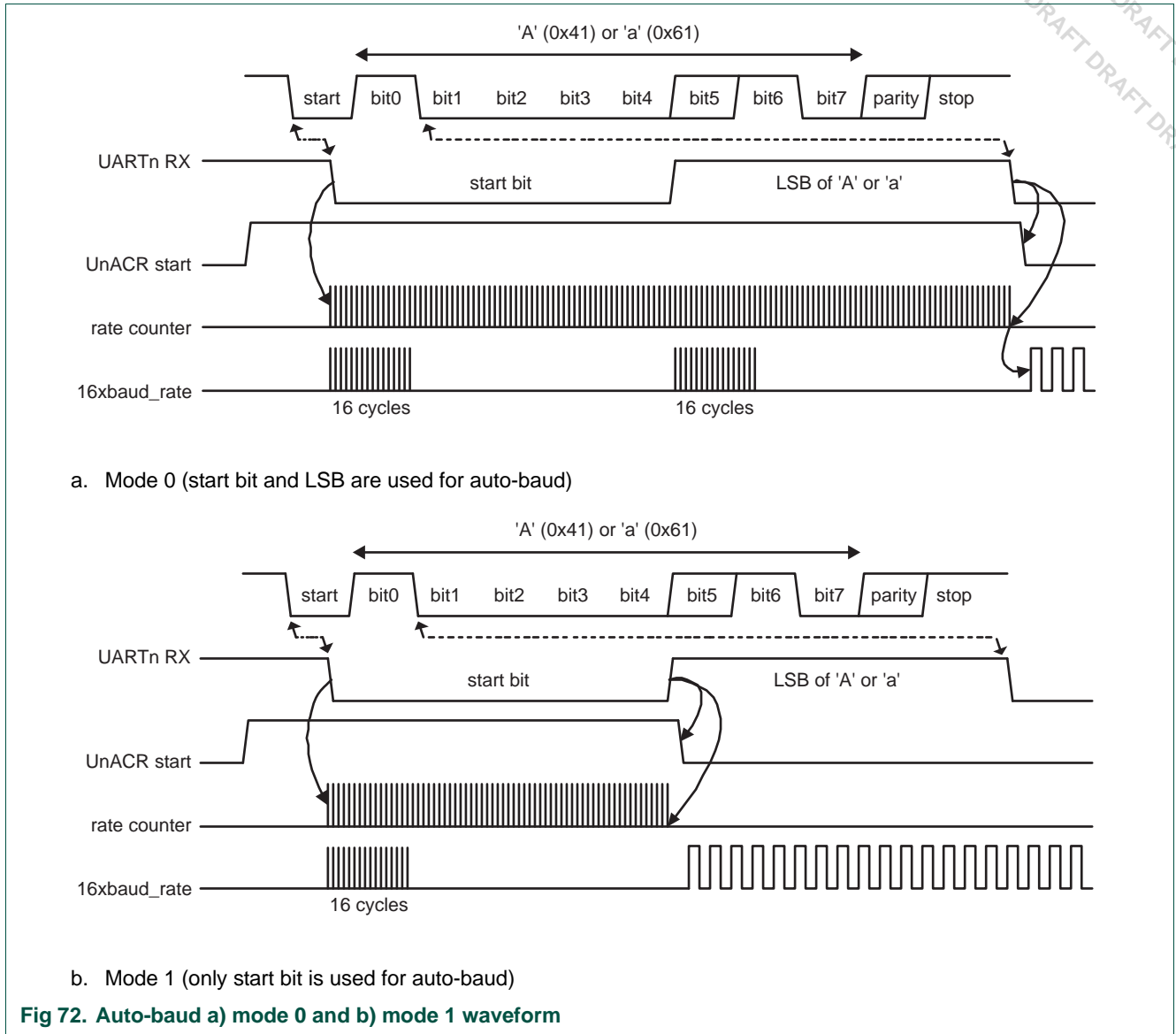
(3)

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UARTn_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

### 18.6.10.2 Auto-baud modes

When the software is expecting an “AT” command, it configures the UARTn with the expected character format and sets the UnACR Start bit. The initial values in the divisor latches UnDLM and UnDLL don’t care. Because of the “A” or “a” ASCII coding (“A” = 0x41, “a” = 0x61), the UARTn Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the UnACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On UnACR Start bit setting, the baud rate measurement counter is reset and the UARTn UnRSR is reset. The UnRSR baud rate is switch to the highest rate.
2. A falling edge on UARTn Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting pclk cycles optionally pre-scaled by the fractional baud-rate generator.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the (fractional baud-rate pre-scaled) UARTn input clock, guaranteeing the start bit is stored in the UnRSR.
4. During the receipt of the start bit (and the character LSB for mode = 0) the rate counter will continue incrementing with the pre-scaled UARTn input clock (pclk).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UARTn Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UARTn Rx pin.
6. The rate counter is loaded into UnDLM/UnDLL and the baud-rate will be switched to normal operation. After setting the UnDLM/UnDLL the end of auto-baud interrupt UnIIR ABEOInt will be set, if enabled. The UnRSR will now continue receiving the remaining bits of the “A/a” character.



### 18.6.11 UARTn Fractional Divider Register (U0FDR - 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028)

The UARTn Fractional Divider Register (UnFDR) controls the clock pre-scaler for the baud rate generation and can be read and written at the user's discretion. This pre-scaler takes the APB clock and generates an output clock according to the specified fractional requirements.

**Important:** If the fractional divider is active ( $DIVADDVAL > 0$ ) and  $DLM = 0$ , the value of the DLL register must be greater than 2.

**Table 401: UARTn Fractional Divider Register (U0FDR - address 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028) bit description**

Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-		Reserved. Read value is undefined, only zero should be written.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of the UART disabled, making sure that the UART is fully software and hardware compatible with UARTs not equipped with this feature.

The UART baud rate can be calculated as ( $n = 0/2/3$ ):

(4)

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Where PCLK is the peripheral clock, UnDLM and UnDLL are the standard UART baud rate divider registers, and DIVADDVAL and MULVAL are UART fractional baud rate generator specific parameters.

The value of MULVAL and DIVADDVAL should comply to the following conditions:

1.  $1 \leq MULVAL \leq 15$
2.  $0 \leq DIVADDVAL \leq 14$
3.  $DIVADDVAL < MULVAL$

The value of the UnFDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the UnFDR register value does not comply to these two requests, then the fractional divider output is undefined. If DIVADDVAL is zero then the fractional divider is disabled, and the clock will not be divided.

### 18.6.11.1 Baud rate calculation

UARTn can operate with or without using the Fractional Divider. In real-life applications it is likely that the desired baud rate can be achieved using several different Fractional Divider settings. The following algorithm illustrates one way of finding a set of DLM, DLL, MULVAL, and DIVADDVAL values. Such set of parameters yields a baud rate with a relative error of less than 1.1% from the desired one.

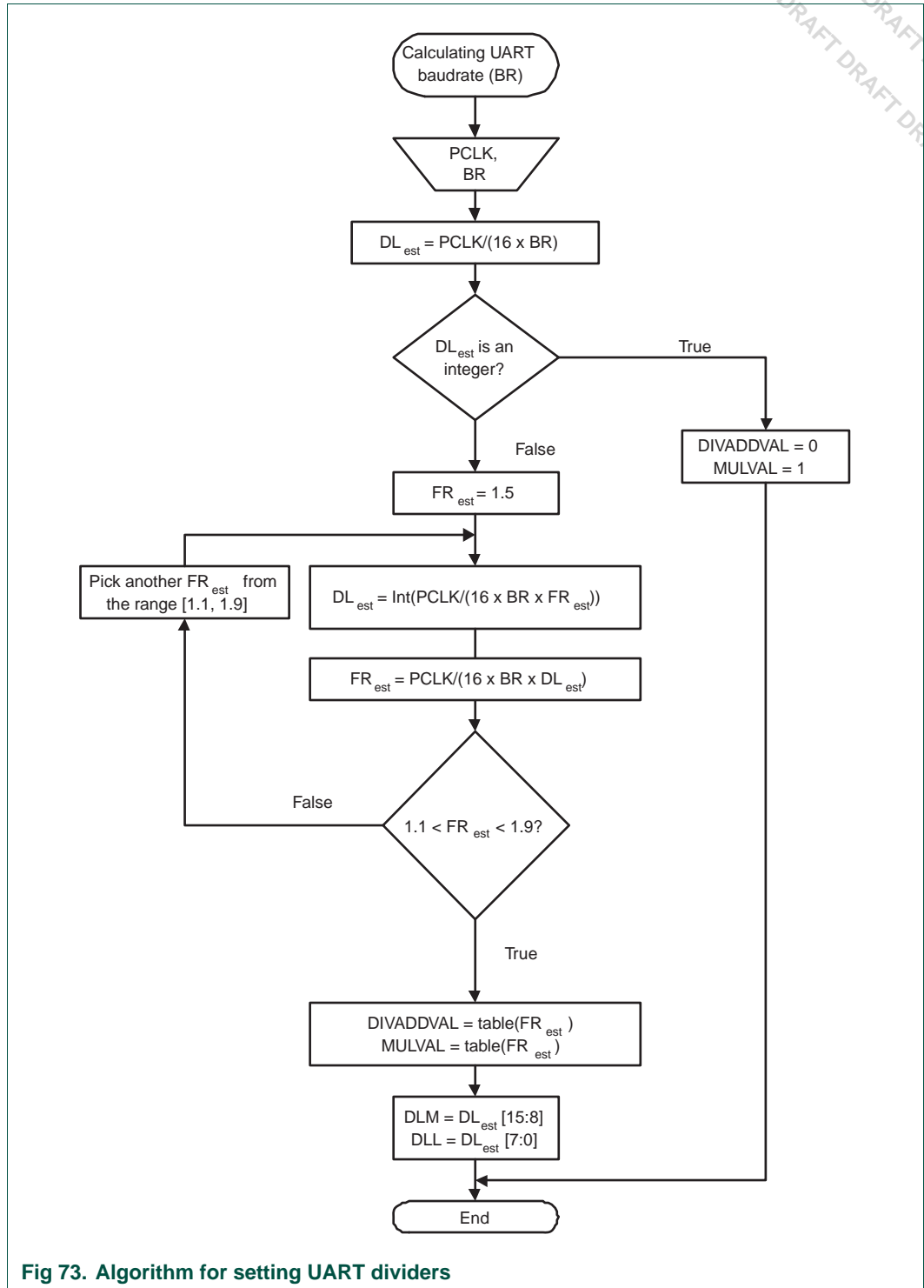


Fig 73. Algorithm for setting UART dividers

**Table 402. Fractional Divider setting look-up table**

FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
1.000	0/1	1.250	1/4	1.500	1/2	1.750	3/4
1.067	1/15	1.267	4/15	1.533	8/15	1.769	10/13
1.071	1/14	1.273	3/11	1.538	7/13	1.778	7/9
1.077	1/13	1.286	2/7	1.545	6/11	1.786	11/14
1.083	1/12	1.300	3/10	1.556	5/9	1.800	4/5
1.091	1/11	1.308	4/13	1.571	4/7	1.818	9/11
1.100	1/10	1.333	1/3	1.583	7/12	1.833	5/6
1.111	1/9	1.357	5/14	1.600	3/5	1.846	11/13
1.125	1/8	1.364	4/11	1.615	8/13	1.857	6/7
1.133	2/15	1.375	3/8	1.625	5/8	1.867	13/15
1.143	1/7	1.385	5/13	1.636	7/11	1.875	7/8
1.154	2/13	1.400	2/5	1.643	9/14	1.889	8/9
1.167	1/6	1.417	5/12	1.667	2/3	1.900	9/10
1.182	2/11	1.429	3/7	1.692	9/13	1.909	10/11
1.200	1/5	1.444	4/9	1.700	7/10	1.917	11/12
1.214	3/14	1.455	5/11	1.714	5/7	1.923	12/13
1.222	2/9	1.462	6/13	1.727	8/11	1.929	13/14
1.231	3/13	1.467	7/15	1.733	11/15	1.933	14/15

**18.6.11.1.1 Example 1: PCLK = 14.7456 MHz, BR = 9600**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 14.7456 \text{ MHz} / (16 \times 9600) = 96$ . Since this  $DL_{est}$  is an integer number,  $DIVADDVAL = 0$ ,  $MULVAL = 1$ ,  $DLM = 0$ , and  $DLL = 96$ .

**18.6.11.1.2 Example 2: PCLK = 12 MHz, BR = 115200**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 12 \text{ MHz} / (16 \times 115200) = 6.51$ . This  $DL_{est}$  is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of  $FR_{est} = 1.5$  a new  $DL_{est} = 4$  is calculated and  $FR_{est}$  is recalculated as  $FR_{est} = 1.628$ . Since  $FR_{est} = 1.628$  is within the specified range of 1.1 and 1.9,  $DIVADDVAL$  and  $MULVAL$  values can be obtained from the attached look-up table.

The closest value for  $FR_{est} = 1.628$  in the look-up [Table 402](#) is  $FR = 1.625$ . It is equivalent to  $DIVADDVAL = 5$  and  $MULVAL = 8$ .

Based on these findings, the suggested UART setup would be:  $DLM = 0$ ,  $DLL = 4$ ,  $DIVADDVAL = 5$ , and  $MULVAL = 8$ . According to [Equation 4](#) the UART rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

**18.6.12 UARTn Transmit Enable Register (U0TER - 0x4000 C030, U2TER - 0x4009 8030, U3TER - 0x4009 C030)**

The UnTER register enables implementation of software flow control. When TXEn=1, UARTn transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UARTn transmission will stop.

Table 403 describes how to use the TXEn bit in order to achieve software flow control.

**Table 403: UARTn Transmit Enable Register (U0TER - address 0x4000 C030, U2TER - 0x4009 8030, U3TER - 0x4009 C030) bit description**

Bit	Symbol	Description	Reset Value
6:0	-	Reserved. Read value is undefined, only zero should be written.	NA
7	TXEN	When this bit is 1, as it is after a Reset, data written to the THR is output on the TXD pin as soon as any preceding data has been sent. If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again. In other words, a 0 in this bit blocks the transfer of characters from the THR or TX FIFO into the transmit shift register. Software implementing software-handshaking can clear this bit when it receives an XOFF character (DC3). Software can set this bit again when it receives an XON (DC1) character.	1

### 18.6.13 UARTn RS485 Control register (U0RS485CTRL - 0x4000 C04C, U2RS485CTRL - 0x4009 804C, U3RS485CTRL - 0x4009 C04C)

The UnRS485CTRL register controls configuration of the RS-485/EIA-485 mode.

**Table 404: UARTn RS485 Control register (U0RS485CTRL - address 0x4000 C04C, U2RS485CTRL - 0x4009 804C, U3RS485CTRL - 0x4009 C04C) bit description**

Bit	Symbol	Value	Description	Reset value
0	NMMEN	0	RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled.	0
		1	RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.	
1	RXDIS	0	The receiver is enabled.	0
		1	The receiver is disabled.	
2	AADEN	0	Auto Address Detect (AAD) is disabled.	0
		1	Auto Address Detect (AAD) is enabled.	
3	-		Reserved. Read value is undefined, only zero should be written.	NA
4	DCTRL	0	Disable Auto Direction Control.	0
		1	Enable Auto Direction Control.	
5	OINV		This bit reverses the polarity of the direction control signal on the Un_OE pin.	0
		0	The direction control pin will be driven to logic '0' when the transmitter has data to be sent. It will be driven to logic '1' after the last bit of data has been transmitted.	
		1	The direction control pin will be driven to logic '1' when the transmitter has data to be sent. It will be driven to logic '0' after the last bit of data has been transmitted.	
31:6	-		Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.14 UARTn RS-485 Address Match register (U0RS485ADRMATCH - 0x4000 C050, U2RS485ADRMATCH - 0x4009 8050, U3RS485ADRMATCH - 0x4009 C050)

The UnRS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

**Table 405. UARTn RS-485 Address Match register (U0RS485ADRMATCH - address 0x4000 C050, U2RS485ADRMATCH - 0x4009 8050, U3RS485ADRMATCH - 0x4009 C050) bit description**

Bit	Symbol	Description	Reset value
7:0	ADRMATCH	Contains the address match value.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.15 UARTn RS-485 Delay value register (U0RS485DLY - 0x4000 C054, U2RS485DLY - 0x4009 8054, U3RS485DLY - 0x4009 C054)

The user may program the 8-bit RS485DLY register with a delay between the last stop bit leaving the TXFIFO and the de-assertion of Un\_OE. This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

**Table 406. UARTn RS-485 Delay value register (U0RS485DLY - address 0x4000 0054, U2RS485DLY - 0x4009 8054, U3RS485DLY - 0x4009 C054) bit description**

Bit	Symbol	Description	Reset value
7:0	DLY	Contains the direction control (UnOE) delay value. This register works in conjunction with an 8-bit counter.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 18.6.16 RS-485/EIA-485 modes of operation

The RS-485/EIA-485 feature allows the UART to be configured as an addressable slave. The addressable slave is one of multiple slaves controlled by a single master.

The UART master transmitter will identify an address character by setting the parity (9th) bit to '1'. For data characters, the parity bit is set to '0'.

Each UART slave receiver can be assigned a unique address. The slave can be programmed to either manually or automatically reject data following an address which is not theirs.

#### RS-485/EIA-485 Normal Multidrop Mode (NMM)

Setting the RS485CTRL bit 0 enables this mode. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received data bytes will be ignored and will not be stored in the RXFIFO. When an address byte is detected (parity bit = '1') it will be placed into the RXFIFO and an Rx Data Ready Interrupt will be generated. The processor can then read the address byte and decide whether or not to enable the receiver to accept the following data.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all received bytes will be accepted and stored in the RXFIFO regardless of whether they are data or address. When an address character is received a parity error interrupt will be generated and the processor can decide whether or not to disable the receiver.

#### RS-485/EIA-485 Auto Address Detection (AAD) mode

When both RS485CTRL register bits 0 (9-bit mode enable) and 2 (AAD mode enable) are set, the UART is in auto address detect mode.

In this mode, the receiver will compare any address byte received (parity = '1') to the 8-bit value programmed into the RS485ADRMATCH register.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received byte will be discarded if it is either a data byte OR an address byte which fails to match the RS485ADRMATCH value.

When a matching address character is detected it will be pushed onto the RXFIFO along with the parity bit, and the receiver will be automatically enabled (RS485CTRL bit 1 will be cleared by hardware). The receiver will also generate n Rx Data Ready Interrupt.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all bytes received will be accepted and stored in the RXFIFO until an address byte which does not match the RS485ADRMATCH value is received. When this occurs, the receiver will be automatically disabled in hardware (RS485CTRL bit 1 will be set), The received non-matching address character will not be stored in the RXFIFO.

### RS-485/EIA-485 Auto Direction Control

RS485/EIA-485 Mode includes the option of allowing the transmitter to automatically control the state of the UnOE pin as a direction control output signal.

Setting RS485CTRL bit 4 = '1' enables this feature.

When Auto Direction Control is enabled, the UnOE pin will be asserted (driven low) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven high) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

When Auto Direction Control is enabled, the selected pin will be asserted (driven low) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven high) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

### RS485/EIA-485 driver delay time

The driver delay time is the delay between the last stop bit leaving the TXFIFO and the de-assertion of UnOE. This delay time can be programmed in the 8-bit RS485DLY register. The delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

### RS485/EIA-485 output inversion

The polarity of the direction control signal on the UnOE pin can be reversed by programming bit 5 in the UnRS485CTRL register. When this bit is set, the direction control pin will be driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin will be driven to logic 0 after the last bit of data has been transmitted.



### 19.1 How to read this chapter

---

Most LPC178x/177x family devices include 5 UARTs. A few devices do not include UART4. Refer to [Section 1.4.1](#) and specific device data sheets for details. UART4 is essentially the same as UARTs 02/3, but with an added synchronous mode and Smart Card mode.

### 19.2 Basic configuration

---

UART4 is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCUART4.  
**Remark:** On reset, UART4 is disabled (PCUART4 = 0).
2. Peripheral clock: This UART operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Baud rate: In register U4LCR ([Table 417](#)), set bit DLAB = 1. This enables access to registers DLL ([Table 411](#)) and DLM ([Table 412](#)) for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register ([Table 423](#)).
4. UART FIFO: Use bit FIFO enable (bit 0) in register U4FCR ([Table 416](#)) to enable the FIFOs.
5. Pins: Select UART pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).  
**Remark:** UART receive pins should not have pull-down resistors enabled.
6. Interrupts: To enable UART interrupts set bit DLAB = 0 in register U4LCR ([Table 417](#)). This enables access to U4IER ([Table 413](#)). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
7. DMA: UART4 transmit and receive functions can operate with the GPDMA controller (see [Table 655](#)).

## 19.3 Features

- Data sizes of 5, 6, 7, and 8 bits.
- Parity generation and checking: odd, even mark, space or none.
- One or two stop bits.
- 16 byte Receive and Transmit FIFOs.
- Built-in baud rate generator, including a fractional rate divider for great versatility.
- Supports DMA for both transmit and receive.
- Auto-baud capability
- Break generation and detection.
- Multiprocessor addressing mode.
- IrDA mode to support infrared communication.
- Support for software flow control.
- RS-485/EIA-485 9-bit mode support with output enable.
- Optional synchronous send or receive mode.
- Optional ISO 7816-3 compliant smart card interface.

## 19.4 Architecture

The architecture of UART4 is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART.

The UART4 receiver block, U4RX, monitors the serial input line, RXDn, for valid input. The UART4 RX Shift Register (U4RSR) accepts valid characters via RXDn. After a valid character is assembled in U4RSR, it is passed to the UART4 RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART4 transmitter block, U4TX, accepts data written by the CPU or host and buffers the data in the UART4 TX Holding Register FIFO (U4THR). The UART4 TX Shift Register (U4TSR) reads the data stored in U4THR and assembles the data to transmit via the serial output pin, TXDn.

The UART4 Baud Rate Generator block, U4BRG, generates the timing enables used by the UART4 TX block. The U4BRG clock input source is the APB clock (PCLK). The main clock is divided down per the divisor specified in the U4DLL and U4DLM registers. This divided down clock is the 16x oversample clock.

The interrupt interface contains registers U4IER and U4IIR. The interrupt interface receives several one clock wide enables from the U4TX and U4RX blocks.

Status information from the U4TX and U4RX is stored in the U4LSR. Control information for the U4TX and U4RX is stored in U4LCR.

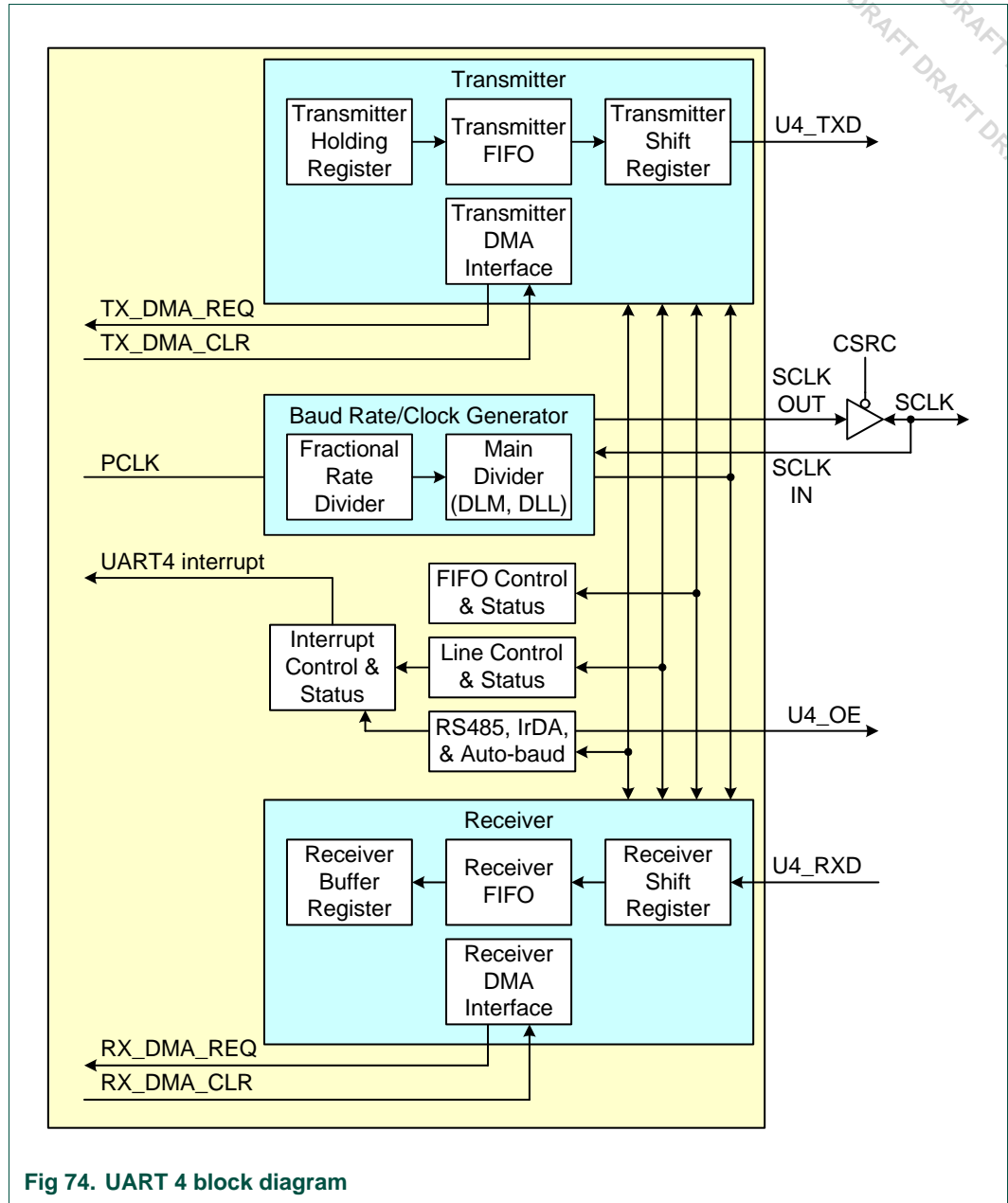


Fig 74. UART 4 block diagram

## 19.5 Pin description

Table 407: UART4 Pin description

Pin	Type	Description
U4_RXD	Input	<b>Serial Input.</b> Serial receive data.
U4_TXD	Output	<b>Serial Output.</b> Serial transmit data (input/output in smart card mode).
U4_OE	Output	<b>Output Enable.</b> RS-485/EIA-485 output enable.
U4_SCLK	I/O	<b>Serial Clock.</b> Clock input or output in synchronous mode.

## 19.6 Register description

UART4 contains registers as shown in [Table 408](#). The Divisor Latch Access Bit (DLAB) is contained in U4LCR7 and enables access to the Divisor Latches.

**Table 408. UART4 Register Map**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	Register Address	Table
U4RBR (DLAB =0)	Receiver Buffer Register. Contains the next received character to be read.	RO	NA	0x400A 4000	<a href="#">409</a>
U4THR (DLAB =0)	Transmit Holding Register. The next character to be transmitted is written here.	WO	NA	0x400A 4000	<a href="#">410</a>
U4DLL (DLAB =1)	Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x01	0x400A 4000	<a href="#">411</a>
U4DLM (DLAB =1)	Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0	0x400A 4004	<a href="#">412</a>
U4IER (DLAB =0)	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART interrupts.	R/W	0	0x400A 4004	<a href="#">413</a>
U4IIR	Interrupt ID Register. Identifies which interrupt(s) are pending.	RO	0x01	0x400A 4008	<a href="#">414</a>
U4FCR	FIFO Control Register. Controls UART FIFO usage and modes.	WO	0	0x400A 4008	<a href="#">416</a>
U4LCR	Line Control Register. Contains controls for frame formatting and break generation.	R/W	0	0x400A 400C	<a href="#">417</a>
U4LSR	Line Status Register. Contains flags for transmit and receive status, including line errors.	RO	0x60	0x400A 4014	<a href="#">418</a>
U4SCR	Scratch Pad Register. 8-bit temporary storage for software.	R/W	0	0x400A 401C	<a href="#">419</a>
U4ACR	Auto-baud Control Register. Contains controls for the auto-baud feature.	R/W	0	0x400A 4020	<a href="#">420</a>
U4ICR	IrDA Control Register. Enables and configures the IrDA mode.	R/W	0	0x400A 4024	<a href="#">421</a>
U4FDR	Fractional Divider Register. Generates a clock input for the baud rate divider.	R/W	0x10	0x400A 4028	<a href="#">423</a>
U4OSR	Oversampling register. Controls the degree of oversampling during each bit time.	R/W	0xF0	0x400A 402C	<a href="#">425</a>
U4SCICTRL	Smart Card Interface control register. Enables and configures the Smart Card Interface feature.	R/W	0	0x400A 4048	<a href="#">426</a>
U4RS485CTRL	RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes.	R/W	0	0x400A 404C	<a href="#">427</a>
U4ADRMATCH	RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode.	R/W	0	0x400A 4050	<a href="#">428</a>
U4RS485DLY	RS-485/EIA-485 direction control delay.	R/W	0	0x400A 4054	<a href="#">429</a>
U4SYNCCTRL	Synchronous mode control register.	R/W	0	0x400A 4058	<a href="#">430</a>
U4TER	Transmit Enable Register. Turns off UART transmitter for use with software flow control.	R/W	0x01	0x400A 405C	<a href="#">431</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 19.6.1 UART4 Receiver Buffer Register (U4RBR - 0x400A C000 when DLAB = 0)

The U4RBR is the top byte of the UART4 Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in LCR must be zero in order to access the U4RBR. The U4RBR is always read-only.

Since PE, FE and BI bits correspond to the byte sitting on the top of the RBR FIFO (i.e. the one that will be read in the next read from the RBR), the right approach for fetching the valid pair of received byte and its status bits is first to read the content of the U0LSR register, and then to read a byte from the U4RBR.

**Table 409: UART4 Receiver Buffer Register (U4RBR - 0x400A 4000 when DLAB = 0) bit description**

Bit	Symbol	Description	Reset Value
7:0	RBR	The UART4 Receiver Buffer Register contains the oldest received byte in the UART4 Rx FIFO.	Undefined
31:8	-	Reserved, the value read from a reserved bit is not defined.	NA

### 19.6.2 UART4 Transmit Holding Register (U4THR - 0x400A C000 when DLAB = 0)

The U4THR is the top byte of the UART4 TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

The Divisor Latch Access Bit (DLAB) in U4LCR must be zero in order to access the U4THR. The U4THR is always write-only.

**Table 410: UART4 Transmit Holding Register (U4THR - 0x400A 4000 when DLAB = 0) bit description**

Bit	Symbol	Description	Reset Value
7:0	THR	Writing to the UART4 Transmit Holding Register causes the data to be stored in the UART4 transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.3 UART4 Divisor Latch LSB register (U4DLL - 0x400A 4000 when DLAB = 1) and UART4 Divisor Latch MSB register (U4DLM - 0x400A C004 when DLAB = 1)

The UART4 Divisor Latch is part of the UART4 Baud Rate Generator and holds the value used, along with the Fractional Divider, to divide the APB clock (PCLK) in order to produce the baud rate clock, which must be 16× the desired baud rate. The U4DLL and U4DLM registers together form a 16-bit divisor where U4DLL contains the lower 8 bits of the divisor and U4DLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed. The Divisor Latch Access Bit (DLAB) in U4LCR must be one in order to access the UART4 Divisor Latches. Details on how to select the right value for U4DLL and U4DLM can be found later in this chapter, see [Section 19.6.12](#).

**Table 411: UART4 Divisor Latch LSB register (U4DLL - 0x400A 4000 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLLSB	The UART4 Divisor Latch LSB Register, along with the U4DLM register, determines the baud rate of the UART4.	0x01
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

**Table 412: UART4 Divisor Latch MSB register (U4DLL - 0x400A 4004 when DLAB = 1) bit description**

Bit	Symbol	Description	Reset Value
7:0	DLMSB	The UART4 Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UART4.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.4 UART4 Interrupt Enable Register (U4IER - 0x400A C004 when DLAB = 0)

The U4IER is used to enable the three UART4 interrupt sources.

**Table 413: UART4 Interrupt Enable Register (U4IER - 0x400A 4004 when DLAB = 0) bit description**

Bit	Symbol	Value	Description	Reset Value
0	RBR Interrupt Enable		Enables the Receive Data Available interrupt for UART4. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	
1	THRE Interrupt Enable		Enables the THRE interrupt for UART4. The status of this can be read from U4LSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Status Interrupt Enable		Enables the UART4 RX line status interrupts. The status of this interrupt can be read from U4LSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
7:3	-		Reserved. Read value is undefined, only zero should be written.	NA
8	ABEOIntEn		Enables the end of auto-baud interrupt.	0
		0	Disable end of auto-baud Interrupt.	
		1	Enable end of auto-baud Interrupt.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable auto-baud time-out Interrupt.	
		1	Enable auto-baud time-out Interrupt.	
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.5 UART4 Interrupt Identification Register (U4IIR - 0x400A C008)

The U4IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an U4IIR access. If an interrupt occurs during an U4IIR access, the interrupt is recorded for the next U4IIR access.

**Table 414: UART4 Interrupt Identification Register (U4IIR - 0x400A 4008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	IntStatus		Interrupt status. Note that U4IIR[0] is active low. The pending interrupt can be determined by evaluating U4IIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	IntId		Interrupt identification. U4IER[3:1] identifies an interrupt corresponding to the UART4 Rx or TX FIFO. All other combinations of U4IER[3:1] not listed below are reserved (000,100,101,111).	0
		011	1 - Receive Line Status (RLS).	
		010	2a - Receive Data Available (RDA).	
		110	2b - Character Time-out Indicator (CTI).	
		001	3 - THRE Interrupt	
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	FIFO Enable		Copies of U4FCR[0].	0
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

Bit U4IIR[9:8] are set by the auto-baud function and signal a time-out or end of auto-baud condition. The auto-baud interrupt conditions are cleared by setting the corresponding Clear bits in the Auto-baud Control Register.

If the IntStatus bit is 1 no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in [Table 415](#). Given the status of U4IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The U4IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UART4 RLS interrupt (U4IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART4 Rx input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART4 Rx error condition that set the interrupt can be observed via U0LSR[4:1]. The interrupt is cleared upon an U4LSR read.

The UART4 RDA interrupt (U4IIR[3:1] = 010) shares the second level priority with the CTI interrupt (U4IIR[3:1] = 110). The RDA is activated when the UART4 Rx FIFO reaches the trigger level defined in U4FCR[7:6] and is reset when the UART4 Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (U4IIR[3:1] = 110) is a second level interrupt and is set when the UART4 Rx FIFO contains at least one character and no UART4 Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART4 Rx FIFO activity (read or write of UART4 RSR) will clear the interrupt. This interrupt is intended to flush the UART4 RBR after a message has been received that is not a multiple of the trigger level size. For example, if a peripheral wished to send a 105 character message and the trigger level was 10 characters, the

CPU would receive 10 RDA interrupts resulting in the transfer of 100 characters and 1 to 5 CTI interrupts (depending on the service routine) resulting in the transfer of the remaining 5 characters.

**Table 415: UART4 Interrupt Handling**

U0IIR[3:0] value <sup>[1]</sup>	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE <sup>[2]</sup> or PE <sup>[2]</sup> or FE <sup>[2]</sup> or BI <sup>[2]</sup>	U4LSR Read <sup>[2]</sup>
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (U4FCR0=1)	U4RBR Read <sup>[3]</sup> or UART4 FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).  The exact time will be: [(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs	U4RBR Read <sup>[3]</sup>
0010	Third	THRE	THRE <sup>[2]</sup>	U4IIR Read (if source of interrupt) or THR write <sup>[4]</sup>

- [1] Values "0000", "0011", "0101", "0111", "1000", "1001", "1010", "1011", "1101", "1110", "1111" are reserved.
- [2] For details see [Section 19.6.8 "UART4 Line Status Register \(U4LSR - 0x400A C014\)"](#)
- [3] For details see [Section 19.6.1 "UART4 Receiver Buffer Register \(U4RBR - 0x400A C000 when DLAB = 0\)"](#)
- [4] For details see [Section 19.6.5 "UART4 Interrupt Identification Register \(U4IIR - 0x400A C008\)"](#) and [Section 19.6.2 "UART4 Transmit Holding Register \(U4THR - 0x400A C000 when DLAB = 0\)"](#)

The UART4 THRE interrupt (U4IIR[3:1] = 001) is a third level interrupt and is activated when the UART4 THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART4 THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE = 1 and there have not been at least two characters in the U4THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to U4THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UART4 THR FIFO has held two or more characters at one time and currently, the U4THR is empty. The THRE interrupt is reset when a U4THR write occurs or a read of the U4IIR occurs and the THRE is the highest interrupt (U4IIR[3:1] = 001).

### 19.6.6 UART4 FIFO Control Register (U4FCR - 0x400A C008)

The write-only U4FCR controls the operation of the UART4 Rx and TX FIFOs.

**Table 416: UART4 FIFO Control Register (U4FCR - 0x400A 4008) bit description**

Bit	Symbol	Value	Description	Reset Value
0	FIFO Enable	0	UART4 FIFOs are disabled. Must not be used in the application.	0
		1	Active high enable for both UART4 Rx and TX FIFOs and U4FCR[7:1] access. This bit must be set for proper UART operation. Any transition on this bit will automatically clear the related UART FIFOs.	



**Table 416: UART4 FIFO Control Register (U4FCR - 0x400A 4008) bit description**

Bit	Symbol	Value	Description	Reset Value
1	RX FIFO Reset	0	No impact on either of UART4 FIFOs.	0
		1	Writing a logic 1 to U4FCR[1] will clear all bytes in UART4 Rx FIFO, reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UART4 FIFOs.	0
		1	Writing a logic 1 to U4FCR[2] will clear all bytes in UART4 TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	DMA Mode Select		When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode. See <a href="#">Section 19.6.6.1</a> .	0
5:4	-		Reserved. Read value is undefined, only zero should be written.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UART4 FIFO characters must be written before an interrupt or DMA request is activated.	0
		00	Trigger level 0 (1 character or 0x01)	
		01	Trigger level 1 (4 characters or 0x04)	
		10	Trigger level 2 (8 characters or 0x08)	
		11	Trigger level 3 (14 characters or 0x0E)	
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.6.1 DMA Operation

The user can optionally operate the UART transmit and/or receive using DMA. The DMA mode is determined by the DMA Mode Select bit in the FCR register. This bit only has an affect when the FIFOs are enabled via the FIFO Enable bit in the FCR register.

#### UART receiver DMA

In DMA mode, the receiver DMA request is asserted on the event of the receiver FIFO level becoming equal to or greater than trigger level, or if a character timeout occurs. See the description of the RX Trigger Level above. The receiver DMA request is cleared by the DMA controller.

#### UART transmitter DMA

In DMA mode, the transmitter DMA request is asserted on the event of the transmitter FIFO transitioning to not full. The transmitter DMA request is cleared by the DMA controller.

### 19.6.7 UART4 Line Control Register (U4LCR - 0x400A C00C)

The U4LCR determines the format of the data character that is to be transmitted or received.

**Table 417: UART4 Line Control Register (U4LCR - 0x400A 400C) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Word Length Select	00	5-bit character length	0
		01	6-bit character length	
		10	7-bit character length	
		11	8-bit character length	

Table 417: UART4 Line Control Register (U4LCR - 0x400A 400C) bit description

Bit	Symbol	Value	Description	Reset Value
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if U4LCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
5:4	Parity Select	00	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	0
		01	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		10	Forced "1" stick parity.	
		11	Forced "0" stick parity.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UART4 TXD is forced to logic 0 when U4LCR[6] is active high.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	
31:8	-	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.8 UART4 Line Status Register (U4LSR - 0x400A C014)

The U4LSR is a read-only register that provides status information on the UART4 TX and RX blocks.

Table 418: UART4 Line Status Register (U4LSR - 0x400A 4014) bit description

Bit	Symbol	Value	Description	Reset Value
0	Receiver Data Ready (RDR)	-	U4LSR[0] is set when the U4RBR holds an unread character and is cleared when the UART4 RBR FIFO is empty.	0
		0	The UART4 receiver FIFO is empty.	
		1	The UART4 receiver FIFO is not empty.	
1	Overrun Error (OE)	-	The overrun error condition is set as soon as it occurs. An U4LSR read clears U4LSR[1]. U4LSR[1] is set when UART4 RSR has a new character assembled and the UART4 RBR FIFO is full. In this case, the UART4 RBR FIFO will not be overwritten and the character in the UART4 RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	Parity Error (PE)	-	When the parity bit of a received character is in the wrong state, a parity error occurs. An U4LSR read clears U4LSR[2]. Time of parity error detection is dependent on U4FCR[0]. <b>Note:</b> A parity error is associated with the character at the top of the UART4 RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	

Table 418: UART4 Line Status Register (U4LSR - 0x400A 4014) bit description

Bit	Symbol	Value	Description	Reset Value
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An U4LSR read clears U4LSR[3]. The time of the framing error detection is dependent on U4FCR[0]. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. <b>Note:</b> A framing error is associated with the character at the top of the UART4 RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXDn is held in the spacing state (all zeroes) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all ones). An U4LSR read clears this status bit. The time of break detection is dependent on U4FCR[0]. <b>Note:</b> The break interrupt is associated with the character at the top of the UART4 RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE))		THRE is set immediately upon detection of an empty UART4 THR and is cleared on a U4THR write.	1
		0	U4THR contains valid data.	
6	Transmitter Empty (TEMT)		TEMT is set when both U4THR and U4TSR are empty; TEMT is cleared when either the U4TSR or the U4THR contain valid data.	1
		0	U4THR and/or the U4TSR contains valid data.	
7	Error in RX FIFO (RXFE)		U4LSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the U4RBR. This bit is cleared when the U4LSR register is read and there are no subsequent errors in the UART4 FIFO.	0
		0	U4RBR contains no UART4 RX errors or U4FCR[0]=0.	
31:8	-		Reserved. The value read from a reserved bit is not defined.	NA
		1	UART4 RBR contains at least one UART4 RX error.	

### 19.6.9 UART4 Scratch Pad Register (U4SCR - 0x400A C01C)

The U4SCR has no effect on the UART4 operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the U4SCR has occurred.

Table 419: UART4 Scratch Pad Register (U4SCR - 0x400A 401C) bit description

Bit	Symbol	Description	Reset Value
7:0	Pad	A readable, writable byte.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.10 UART4 Auto-baud Control Register (U4ACR - 0x400A C020)

The UART4 Auto-baud Control Register (U4ACR) controls the process of measuring the incoming clock/data rate for the baud rate generation and can be read and written at user's discretion.

**Table 420: UART4 Auto-baud Control Register (U4ACR - 0x400A 4020) bit description**

Bit	Symbol	Value	Description	Reset value
0	Start		This bit is automatically cleared after auto-baud completion.	0
		0	Auto-baud stop (auto-baud is not running).	
		1	Auto-baud start (auto-baud is running). Auto-baud run bit. This bit is automatically cleared after auto-baud completion.	
1	Mode		Auto-baud mode select bit.	0
		0	Mode 0.	
		1	Mode 1.	
2	AutoRestart	0	No restart.	0
		1	Restart in case of time-out (counter restarts at next UART4 Rx falling edge)	0
7:3	-		Reserved. Read value is undefined, only zero should be written.	NA
8	ABEOIntClr		End of auto-baud interrupt clear bit (write-only accessible). Writing a 1 will clear the corresponding interrupt in the U4IIR. Writing a 0 has no impact.	0
9	ABTOIntClr		Auto-baud time-out interrupt clear bit (write-only accessible). Writing a 1 will clear the corresponding interrupt in the U4IIR. Writing a 0 has no impact.	0
31:10	-		Reserved. Read value is undefined, only zero should be written.	NA

#### 19.6.10.1 Auto-baud

The UART4 auto-baud function can be used to measure the incoming baud-rate based on the "AT" protocol (Hayes command). If enabled the auto-baud feature will measure the bit time of the receive data stream and set the divisor latch registers U4DLM and U4DLL accordingly.

**Remark:** the fractional rate divider is not connected during auto-baud operations, and therefore should not be used when the auto-baud feature is needed.

Auto-baud is started by setting the U4ACR Start bit. Auto-baud can be stopped by clearing the U4ACR Start bit. The Start bit will clear once auto-baud has finished and reading the bit will return the status of auto-baud (pending/finished).

Two auto-baud measuring modes are available which can be selected by the U4ACR Mode bit. In mode 0 the baud-rate is measured on two subsequent falling edges of the UART4 Rx pin (the falling edge of the start bit and the falling edge of the least significant bit). In mode 1 the baud-rate is measured between the falling edge and the subsequent rising edge of the UART4 Rx pin (the length of the start bit).

The U4ACR AutoRestart bit can be used to automatically restart baud-rate measurement if a time-out occurs (the rate measurement counter overflows). If this bit is set the rate measurement will restart at the next falling edge of the UART4 Rx pin.

The auto-baud function can generate two interrupts.

- The U4IIR ABTOInt interrupt will get set if the interrupt is enabled (U4IER ABTOIntEn is set and the auto-baud rate measurement counter overflows).

- The U4IIR ABEOInt interrupt will get set if the interrupt is enabled (U4IER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding U4ACR ABTOIntClr and ABEOIntEn bits.

Typically the fractional baud-rate generator is disabled (DIVADDVAL = 0) during auto-baud. However, if the fractional baud-rate generator is enabled (DIVADDVAL > 0), it is going to impact the measuring of UART4 Rx pin baud-rate, but the value of the U4FDR register is not going to be modified after rate measurement. Also, when auto-baud is used, any write to U4DLM and U4DLL registers should be done before U4ACR register write. The minimum and the maximum baud rates supported by UART4 are function of pclk, number of data bits, stop bits and parity bits.

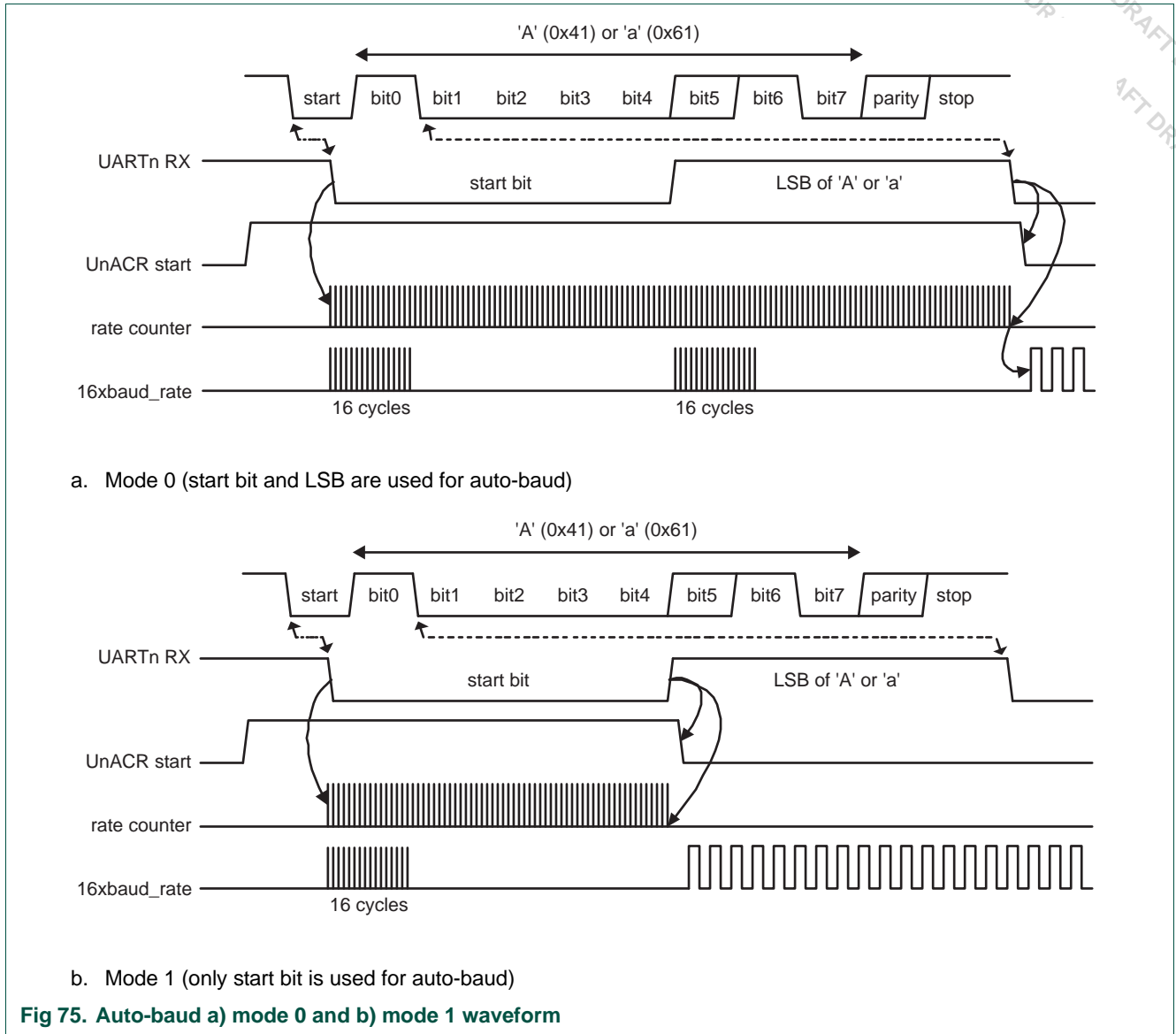
(5)

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART4_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

#### 19.6.10.2 Auto-baud modes

When the software is expecting an “AT” command, it configures the UART4 with the expected character format and sets the U4ACR Start bit. The initial values in the divisor latches U4DLM and U4DLL don't care. Because of the “A” or “a” ASCII coding (“A” = 0x41, “a” = 0x61), the UART4 Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the U4ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On U4ACR Start bit setting, the baud rate measurement counter is reset and the UART4 U4RSR is reset. The U4RSR baud rate is switch to the highest rate.
2. A falling edge on UART4 Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting pclk cycles optionally pre-scaled by the fractional baud-rate generator.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the (fractional baud-rate pre-scaled) UART4 input clock, guaranteeing the start bit is stored in the U4RSR.
4. During the receipt of the start bit (and the character LSB for mode = 0) the rate counter will continue incrementing with the pre-scaled UART4 input clock (pclk).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UART4 Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UART4 Rx pin.
6. The rate counter is loaded into U4DLM/U4DLL and the baud-rate will be switched to normal operation. After setting the U4DLM/U4DLL the end of auto-baud interrupt U4IIR ABEOInt will be set, if enabled. The U4RSR will now continue receiving the remaining bits of the “A/a” character.



### 19.6.11 UART4 IrDA Control Register (U4ICR - 0x400A C024)

The IrDA Control Register enables and configures the IrDA mode on each UART. The value of U4ICR should not be changed while transmitting or receiving data, or data loss or corruption may occur.

**Table 421: UART4 IrDA Control Register (U4ICR - 0x400A 4024) bit description**

Bit	Symbol	Value	Description	Reset value
0	IrDAEn	0	IrDA mode on UART4 is disabled, UART4 acts as a standard UART.	0
		1	IrDA mode on UART4 is enabled.	
1	IrDAInv		When 1, the serial input is inverted. This has no effect on the serial output. When 0, the serial input is not inverted.	0

**Table 421: UART4 IrDA Control Register (U4ICR - 0x400A 4024) bit description**

Bit	Symbol	Value	Description	Reset value
2	FixPulseEn		When 1, enabled IrDA fixed pulse width mode.	0
5:3	PulseDiv		Configures the pulse when FixPulseEn = 1. See text below for details.	0
31:6	-		Reserved. Read value is undefined, only zero should be written.	0

The PulseDiv bits in U4ICR are used to select the pulse width when the fixed pulse width mode is used in IrDA mode (IrDAEn = 1 and FixPulseEn = 1). The value of these bits should be set so that the resulting pulse width is at least 1.63 μs. [Table 422](#) shows the possible pulse widths.

**Table 422: IrDA Pulse Width**

FixPulseEn	PulseDiv	IrDA Transmitter Pulse width (μs)
0	x	3 / (16 × baud rate)
1	0	2 × T <sub>PCLK</sub>
1	1	4 × T <sub>PCLK</sub>
1	2	8 × T <sub>PCLK</sub>
1	3	16 × T <sub>PCLK</sub>
1	4	32 × T <sub>PCLK</sub>
1	5	64 × T <sub>PCLK</sub>
1	6	128 × T <sub>PCLK</sub>
1	7	256 × T <sub>PCLK</sub>

### 19.6.12 UART4 Fractional Divider Register (U4FDR - 0x400A C028)

The UART4 Fractional Divider Register (U4FDR) controls the clock pre-scaler for the baud rate generation and can be read and written at the user’s discretion. This pre-scaler takes the APB clock and generates an output clock according to the specified fractional requirements.

**Important:** If the fractional divider is active (DIVADDVAL > 0) and DLM = 0, the value of the DLL register must be greater than 2.

**Table 423: UART4 Fractional Divider Register (U4FDR - 0x400A 4028) bit description**

Bit	Function	Value	Description	Reset value
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UART4 baud rate.	0
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UART4 to operate properly, regardless of whether the fractional baud-rate generator is used or not.	1
31:8	-		Reserved. Read value is undefined, only zero should be written.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of the UART disabled, making sure that the UART is fully software and hardware compatible with UARTs not equipped with this feature.

The UART baud rate can be calculated as:

(6)

$$UART4_{baudrate} = \frac{PCLK}{16 \times (256 \times U4DLM + U4DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Where PCLK is the peripheral clock, U4DLM and U4DLL are the standard UART baud rate divider registers, and DIVADDVAL and MULVAL are UART fractional baud rate generator specific parameters.

The value of MULVAL and DIVADDVAL should comply to the following conditions:

1.  $1 \leq MULVAL \leq 15$
2.  $0 \leq DIVADDVAL \leq 14$
3.  $DIVADDVAL < MULVAL$

The value of the U4FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the U4FDR register value does not comply to these two requests, then the fractional divider output is undefined. If DIVADDVAL is zero then the fractional divider is disabled, and the clock will not be divided.

### 19.6.12.1 Baud rate calculation

UART4 can operate with or without using the Fractional Divider. In real-life applications it is likely that the desired baud rate can be achieved using several different Fractional Divider settings. The following algorithm illustrates one way of finding a set of DLM, DLL, MULVAL, and DIVADDVAL values. Such set of parameters yields a baud rate with a relative error of less than 1.1% from the desired one.



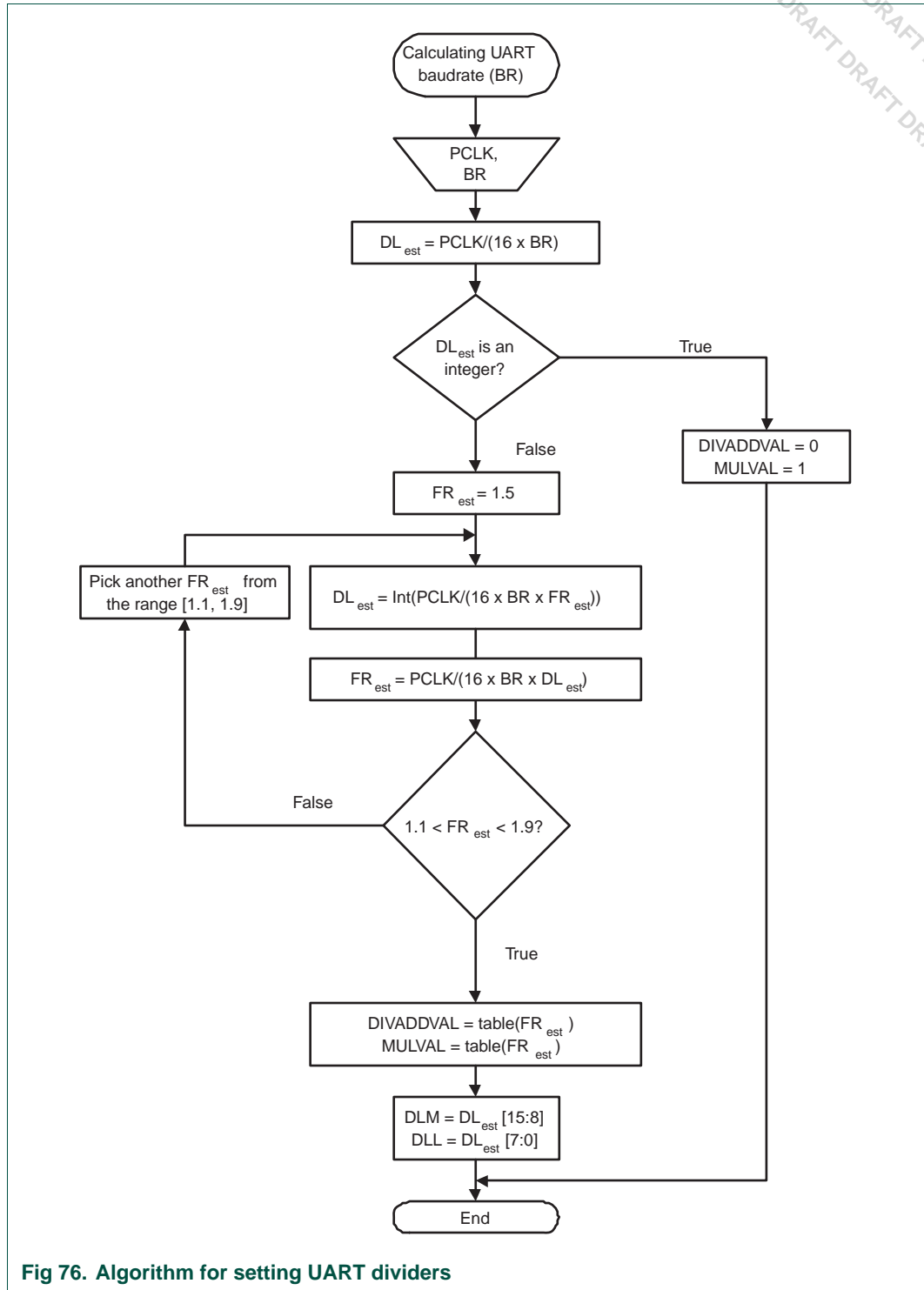


Fig 76. Algorithm for setting UART dividers

**Table 424. Fractional Divider setting look-up table**

FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
1.000	0/1	1.250	1/4	1.500	1/2	1.750	3/4
1.067	1/15	1.267	4/15	1.533	8/15	1.769	10/13
1.071	1/14	1.273	3/11	1.538	7/13	1.778	7/9
1.077	1/13	1.286	2/7	1.545	6/11	1.786	11/14
1.083	1/12	1.300	3/10	1.556	5/9	1.800	4/5
1.091	1/11	1.308	4/13	1.571	4/7	1.818	9/11
1.100	1/10	1.333	1/3	1.583	7/12	1.833	5/6
1.111	1/9	1.357	5/14	1.600	3/5	1.846	11/13
1.125	1/8	1.364	4/11	1.615	8/13	1.857	6/7
1.133	2/15	1.375	3/8	1.625	5/8	1.867	13/15
1.143	1/7	1.385	5/13	1.636	7/11	1.875	7/8
1.154	2/13	1.400	2/5	1.643	9/14	1.889	8/9
1.167	1/6	1.417	5/12	1.667	2/3	1.900	9/10
1.182	2/11	1.429	3/7	1.692	9/13	1.909	10/11
1.200	1/5	1.444	4/9	1.700	7/10	1.917	11/12
1.214	3/14	1.455	5/11	1.714	5/7	1.923	12/13
1.222	2/9	1.462	6/13	1.727	8/11	1.929	13/14
1.231	3/13	1.467	7/15	1.733	11/15	1.933	14/15

**19.6.12.1.1 Example 1: PCLK = 14.7456 MHz, BR = 9600**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 14.7456 \text{ MHz} / (16 \times 9600) = 96$ . Since this  $DL_{est}$  is an integer number,  $DIVADDVAL = 0$ ,  $MULVAL = 1$ ,  $DLM = 0$ , and  $DLL = 96$ .

**19.6.12.1.2 Example 2: PCLK = 12 MHz, BR = 115200**

According to the provided algorithm  $DL_{est} = PCLK / (16 \times BR) = 12 \text{ MHz} / (16 \times 115200) = 6.51$ . This  $DL_{est}$  is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of  $FR_{est} = 1.5$  a new  $DL_{est} = 4$  is calculated and  $FR_{est}$  is recalculated as  $FR_{est} = 1.628$ . Since  $FR_{est} = 1.628$  is within the specified range of 1.1 and 1.9,  $DIVADDVAL$  and  $MULVAL$  values can be obtained from the attached look-up table.

The closest value for  $FR_{est} = 1.628$  in the look-up [Table 424](#) is  $FR = 1.625$ . It is equivalent to  $DIVADDVAL = 5$  and  $MULVAL = 8$ .

Based on these findings, the suggested UART setup would be:  $DLM = 0$ ,  $DLL = 4$ ,  $DIVADDVAL = 5$ , and  $MULVAL = 8$ . According to [Equation 6](#) the UART rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

**19.6.13 UART4 Oversampling Register (U4OSR - 0x400A 402C)**

In most applications, the UART samples received data 16 times in each nominal bit time, and sends bits that are 16 input clocks wide. This register allows software to control the ratio between the input clock and bit clock. This is required for smart card mode, and provides an alternative to fractional division for other modes.

**Table 425. UART4 Oversampling Register (U4OSR - 0x400A 402C) bit description**

Bit	Symbol	Description	Reset value
0	-	Reserved. Read value is undefined, only zero should be written.	NA
3:1	OSFrac	Fractional part of the oversampling ratio, in units of 1/8th of an input clock period. (001 = 0.125, ..., 111 = 0.875)	0
7:4	OSInt	Integer part of the oversampling ratio, minus 1. The reset values equate to the normal operating mode of 16 input clocks per bit time.	0xF
14:8	FDInt	In Smart Card mode, these bits act as a more-significant extension of the OSInt field, allowing an oversampling ratio up to 2048 as required by ISO7816-3. In Smart Card mode, bits 14:4 should initially be set to 371, yielding an oversampling ratio of 372.	0
31:15	-	Reserved. Read value is undefined, only zero should be written.	NA

**Example:** For a baud rate of 3.25Mbps with a 24 MHz UART clock frequency, the ideal oversampling ratio is 24/3.25 or 7.3846. Setting OSInt to 0110 for 7 clocks/bit and OSFrac to 011 for 0.375 clocks/bit, results in an oversampling ratio of 7.375.

In smart card mode, OSInt is extended by FDInt. This extends the possible oversampling to 2048, as required to support ISO 7816-3. Note that this value can be exceeded when  $D < 0$ , but this is not supported by the UART. When smart card mode is enabled, the initial value of OSInt and FDInt should be programmed as “00101110011” (372 minus one).

### 19.6.14 UART4 Smart Card Interface Control register (U4SCICTRL - 0x400A 4048)

This register allows the UART to be used in ISO7816-3 compliant asynchronous smart card applications.

**Table 426. UART4 Smart Card Interface Control register (U4SCICTRL - 0x400A 4048) bit description**

Bit	Symbol	Value	Description	Reset value
0	SCIEN		Smart Card Interface Enable.	0
		0	Smart card interface disabled.	
		1	Asynchronous half duplex smart card interface is enabled.	
1	NACKDIS		NACK response disable. Only applicable in T=0.	0
		0	A NACK response is enabled.	
		1	A NACK response is inhibited.	
2	PROTSEL		Protocol selection as defined in the ISO7816-3 standard.	0
		0	T = 0	
		1	T = 1	
7:5	TXRETRY		When the protocol selection T bit (above) is 0, the field controls the maximum number of retransmissions that the UART will attempt if the remote device signals NACK. When NACK has occurred this number of times plus one, the Tx Error bit in the LSR is set, an interrupt is requested if enabled, and the UART is locked until the FIFO is cleared.	-
15:8	XTRAGUARD		When the protocol selection T bit (above) is 0, this field indicates the number of bit times by which the guard time after a character transmitted by the UART should exceed the nominal 2 bit times. 0xFF in this field may indicate that there is just a single bit after a character and 11 bit times/character	
31:16	-		Reserved. Read value is undefined, only zero should be written.	NA

**19.6.14.1 Smart Card Connection**

When the SCIEN bit in the U4SCICTRL register is set as described above, the UART provides bidirectional serial data on the TXD pin. No RXD pin is used when SCIEN is 1. If the UART SCLK function is enabled in the I/O Configuration block, a serial clock is output on the pin: use of such a clock is optional for smart cards. Software must use timers to implement character and block waiting times (no hardware support via trigger signals is provided in the UART). GPIO pins can be used to control the smart card reset and power pins.

**19.6.14.2 Smart Card Setup**

The following must be set up in smart card applications:

- If necessary, reset the UART as described in [Section 3.5](#).
- Program one IOCON register to enable a UART TXD function.
- If the smart card to be communicated with requires (or may require) a clock, program one IOCON register for the UART SCLK function. The UART will use it as an output.
- Enable the UART clock and set up UART clocking for an initial UART frequency of 3.58 MHz.
- Program the OSR ([Section 19.6.13](#)) for 372x oversampling.
- Program the LCR ([Section 19.6.7](#)) for 8-bit characters, parity enabled, even parity.
- Program the GPIO signals associated with the smart card so that (in this order):
  - Reset is low.
  - VCC is provided to the card (GPIO pins do not have the required 200 mA drive).
  - VPP (if provided to the card) is at “idle” state.
- Program SCICTRL ([Section 19.6.14](#)) to enable the smart card feature with the desired options.
- Set up one or more timer(s) to provide timing as needed for ISO 7816 startup.

Thereafter, software should monitor the UART and timer status so as to interact with the smart card as described in ISO 7816 3.2.b and subsequently.

**19.6.15 UART4 RS485 Control register (U4RS485CTRL - 0x400A 404C)**

The U4RS485CTRL register controls configuration of the RS-485/EIA-485 mode.

**Table 427: UART4 RS485 Control register (U4RS485CTRL - 0x400A 404C) bit description**

Bit	Symbol	Value	Description	Reset value
0	NMMEN	0	RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled.	0
		1	RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.	
1	RXDIS	0	The receiver is enabled.	0
		1	The receiver is disabled.	
2	AADEN	0	Auto Address Detect (AAD) is disabled.	0
		1	Auto Address Detect (AAD) is enabled.	
3	-		Reserved. Read value is undefined, only zero should be written.	NA

**Table 427: UART4 RS485 Control register (U4RS485CTRL - 0x400A 404C) bit description**

Bit	Symbol	Value	Description	Reset value
4	DCTRL	0	Disable Auto Direction Control.	0
		1	Enable Auto Direction Control.	
5	OINV		This bit reverses the polarity of the direction control signal on the U4_OE pin.	0
		0	The direction control pin will be driven to logic '0' when the transmitter has data to be sent. It will be driven to logic '1' after the last bit of data has been transmitted.	
		1	The direction control pin will be driven to logic '1' when the transmitter has data to be sent. It will be driven to logic '0' after the last bit of data has been transmitted.	
31:6	-		Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.16 UART4 RS-485 Address Match register (U4RS485ADRMATCH - 0x400A 4050)

The U4RS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

**Table 428. UART4 RS-485 Address Match register (U4RS485ADRMATCH - 0x400A 4050) bit description**

Bit	Symbol	Description	Reset value
7:0	ADRMATCH	Contains the address match value.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.17 UART4 RS-485 Delay value register (U4RS485DLY - 0x400A 4054)

The user may program the 8-bit RS485DLY register with a delay between the last stop bit leaving the TXFIFO and the de-assertion of U4\_OE. This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

**Table 429. UART4 RS-485 Delay value register (U4RS485DLY - 0x400A 4054) bit description**

Bit	Symbol	Description	Reset value
7:0	DLY	Contains the direction control (U4OE) delay value. This register works in conjunction with an 8-bit counter.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 19.6.18 RS-485/EIA-485 modes of operation

The RS-485/EIA-485 feature allows the UART to be configured as an addressable slave. The addressable slave is one of multiple slaves controlled by a single master.

The UART master transmitter will identify an address character by setting the parity (9th) bit to '1'. For data characters, the parity bit is set to '0'.

Each UART slave receiver can be assigned a unique address. The slave can be programmed to either manually or automatically reject data following an address which is not theirs.

#### RS-485/EIA-485 Normal Multidrop Mode (NMM)

Setting the RS485CTRL bit 0 enables this mode. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received data bytes will be ignored and will not be stored in the RXFIFO. When an address byte is detected (parity bit = '1') it will be placed into the RXFIFO and an Rx Data Ready Interrupt will be generated. The processor can then read the address byte and decide whether or not to enable the receiver to accept the following data.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all received bytes will be accepted and stored in the RXFIFO regardless of whether they are data or address. When an address character is received a parity error interrupt will be generated and the processor can decide whether or not to disable the receiver.

### RS-485/EIA-485 Auto Address Detection (AAD) mode

When both RS485CTRL register bits 0 (9-bit mode enable) and 2 (AAD mode enable) are set, the UART is in auto address detect mode.

In this mode, the receiver will compare any address byte received (parity = '1') to the 8-bit value programmed into the RS485ADRMATCH register.

If the receiver is DISABLED (RS485CTRL bit 1 = '1') any received byte will be discarded if it is either a data byte OR an address byte which fails to match the RS485ADRMATCH value.

When a matching address character is detected it will be pushed onto the RXFIFO along with the parity bit, and the receiver will be automatically enabled (RS485CTRL bit 1 will be cleared by hardware). The receiver will also generate n Rx Data Ready Interrupt.

While the receiver is ENABLED (RS485CTRL bit 1 = '0') all bytes received will be accepted and stored in the RXFIFO until an address byte which does not match the RS485ADRMATCH value is received. When this occurs, the receiver will be automatically disabled in hardware (RS485CTRL bit 1 will be set), The received non-matching address character will not be stored in the RXFIFO.

### RS-485/EIA-485 Auto Direction Control

RS485/EIA-485 Mode includes the option of allowing the transmitter to automatically control the state of the U4OE pin as a direction control output signal.

Setting RS485CTRL bit 4 = '1' enables this feature.

When Auto Direction Control is enabled, the U4OE pin will be asserted (driven low) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven high) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

When Auto Direction Control is enabled, the selected pin will be asserted (driven low) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven high) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

### RS485/EIA-485 driver delay time

The driver delay time is the delay between the last stop bit leaving the TXFIFO and the de-assertion of U4OE. This delay time can be programmed in the 8-bit RS485DLY register. The delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

### RS485/EIA-485 output inversion

The polarity of the direction control signal on the U4OE pin can be reversed by programming bit 5 in the U4RS485CTRL register. When this bit is set, the direction control pin will be driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin will be driven to logic 0 after the last bit of data has been transmitted.

### 19.6.19 UART4 Synchronous mode control register (U4SYNCCTRL - 0x400A 4058)

SYNCCTRL register controls the synchronous mode. When this mode is in effect, the UART generates or receives a bit clock on the SCLK pin and applies it to the transmit and receive shift registers.

Table 430. UART4 Synchronous mode control register (U4SYNCCTRL - 0x400A 4058) bit description

Bit	Symbol	Value	Description	Reset value
0	SYNC		Enables synchronous mode.	0
		0	Disabled	
		1	Enabled	
1	CSRC		Clock source select.	0
		0	Synchronous slave mode (SCLK in)	
		1	Synchronous master mode (SCLK out)	
2	FES		Falling edge sampling.	0
		0	RxD is sampled on the rising edge of SCLK	
		1	RxD is sampled on the falling edge of SCLK	
3	TSBYPASS		Transmit synchronization bypass in synchronous slave mode.	0
		0	The input clock is synchronized prior to being used in clock edge detection logic.	
		1	The input clock is not synchronized prior to being used in clock edge detection logic. This allows for a higher input clock rate at the expense of potential metastability.	
4	CSCEN		Continuous master clock enable (used only when CSRC is 1)	0
		0	SCLK cycles only when characters are being sent on TxD	
		1	SCLK runs continuously (characters can be received on RxD independently from transmission on TxD)	
5	SSSDIS		Start/stop bits	0
		0	Send start and stop bits as in other modes.	
		1	Do not send start/stop bits.	
6	CCCLR		Continuous clock clear	0
		0	CSCEN is under software control.	
		1	Hardware clears CSCEN after each character is received.	
31:6	-		Reserved. The value read from a reserved bit is not defined.	NA

After reset, synchronous mode is disabled. Synchronous mode is enabled by setting the SYNC bit. When SYNC is 1, the UART operates as follows:

1. The CSRC bit controls whether the UART sends (master mode) or receives (slave mode) a serial bit clock on the SCLK pin.

2. When CSRC is 1 selecting master mode, the CSCEN bit selects whether the UART produces clocks on SCLK continuously (CSCEN=1) or only when transmit data is being sent on TxD (CSCEN=0).
3. The SSDIS bit controls whether start and stop bits are used. When SSDIS is 0, the UART sends and samples for start and stop bits as in other modes. When SSDIS is 1, the UART neither sends nor samples for start or stop bits, and each falling edge on SCLK samples a data bit on RxD into the receive shift register, as well as shifting the transmit shift register.

The rest of this section provides further details of operation when SYNC is 1.

Data changes on TxD from falling edges on SCLK. When SSDIS is 0, the FES bit controls whether the UART samples serial data on RxD on rising edges or falling edges on SCLK. When SSDIS is 1, the UART ignores FES and always samples RxD on falling edges on SCLK.

The combination SYNC=1, CSRC=1, CSCEN=1, and SSDIS=1 is a difficult operating mode, because SCLK applies to both directions of data flow and there is no defined mechanism to signal the receivers when valid data is present on TxD or RxD.

Lacking such a mechanism, SSDIS=1 can be used with CSCEN=0 or CSRC=0 in a mode similar to the SPI protocol, in which characters are (at least conceptually) “exchanged” between the UART and remote device for each set of 8 clock cycles on SCLK. Such operation can be called full-duplex, but the same hardware mode can be used in a half-duplex way under control of a higher-layer protocol, in which the source of SCLK toggles it in groups of N cycles whenever data is to be sent in either direction. (N being the number of bits/character.)

When the UART4 is the clock source (CSRC=1), such half-duplex operation can lead to the requirement of writing a dummy character to the Transmitter Holding Register in order to generate 8 clocks so that a character can be received. The CCCLR bit provides a more natural way of programming half-duplex reception. When the higher-layer protocol dictates that the UART should receive a character, software should write the SYNCCTRL register with CSCEN=1 and CCCLR=1. After the UART has sent N clock cycles and thus received a character, it clears the CSCEN bit. If more characters need to be received thereafter, software can repeat setting CSCEN and CCCLR.

Aside from such half-duplex operation, the primary use of CSCEN=1 is with SSDIS=0, so that start bits indicate the transmission of each character in each direction.

### 19.6.20 UART4 Transmit Enable Register (U4TER - 0x400A C05C)

The U4TER register enables implementation of software flow control. When TXEn=1, UART4 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART4 transmission will stop.

[Table 431](#) describes how to use the TXEn bit in order to achieve software flow control.



**Table 431: UART4 Transmit Enable Register (U4TER - 0x400A 4030) bit description**

Bit	Symbol	Description	Reset Value
0	TXEN	When this bit is 1, as it is after a Reset, data written to the THR is output on the TXD pin as soon as any preceding data has been sent. If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again. In other words, a 0 in this bit blocks the transfer of characters from the THR or TX FIFO into the transmit shift register. Software implementing software-handshaking can clear this bit when it receives an XOFF character (DC3). Software can set this bit again when it receives an XON (DC1) character.	1
7:6	-	Reserved. Read value is undefined, only zero should be written.	NA

## 20.1 Basic configuration

The CAN1/2 peripherals are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bits PCAN1/2.  
**Remark:** On reset, the CAN1/2 blocks are disabled (PCAN1/2 = 0).
2. Peripheral clock: The CAN interfaces operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).  
**Remark:** If CAN baud rates above 100 kbit/s (see [Table 443](#)) are needed, do not select the IRC as the clock source (see [Section 4.6](#)).
3. Wake-up: CAN controllers are able to wake up the microcontroller from Power-down mode, see [Section 4.7.8](#).
4. Pins: Select CAN1/2 pins through and their pin modes through the relevant IOCON registers ([Section 8.4.1](#)).
5. Interrupts: CAN interrupts are enabled using the CAN1/2IER registers ([Table 442](#)). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
6. CAN controller initialization: see CANMOD register ([Section 20.7.1](#)).

## 20.2 CAN controllers

Controller Area Network (CAN) is the definition of a high performance communication protocol for serial data communication. The CAN Controller is designed to provide a full implementation of the CAN-Protocol according to the CAN Specification Version 2.0B. Microcontrollers with this on-chip CAN controller are used to build powerful local networks by supporting distributed real-time control with a very high level of security. The applications are automotive, industrial environments, and high speed networks as well as low cost multiplex wiring. The result is a strongly reduced wiring harness and enhanced diagnostic and supervisory capabilities.

The CAN block is intended to support multiple CAN buses simultaneously, allowing the device to be used as a gateway, switch, or router among a number of CAN buses in various applications.

The CAN module consists of two elements: the controller and the Acceptance Filter. All registers and the RAM are accessed as 32-bit words.

## 20.3 Features

### 20.3.1 General CAN features

- Compatible with *CAN specification 2.0B, ISO 11898-1*.
- Multi-master architecture with non destructive bit-wise arbitration.
- Bus access priority determined by the message identifier (11-bit or 29-bit).

- Guaranteed latency time for high priority messages.
- Programmable transfer rate (up to 1 Mbit/s).
- Multicast and broadcast message facility.
- Data length from 0 up to 8 bytes.
- Powerful error handling capability.
- Non-return-to-zero (NRZ) encoding/decoding with bit stuffing.

### 20.3.2 CAN controller features

- 2 CAN controllers and buses.
- Supports 11-bit identifier as well as 29-bit identifier.
- Double Receive Buffer and Triple Transmit Buffer.
- Programmable Error Warning Limit and Error Counters with read/write access.
- Arbitration Lost Capture and Error Code Capture with detailed bit position.
- Single Shot Transmission (no re-transmission).
- Listen Only Mode (no acknowledge, no active error flags).
- Reception of "own" messages (Self Reception Request).

### 20.3.3 Acceptance filter features

- Fast hardware implemented search algorithm supporting a large number of CAN identifiers.
- Global Acceptance Filter recognizes 11-bit and 29-bit Rx Identifiers for all CAN buses.
- Allows definition of explicit and groups for 11-bit and 29-bit CAN identifiers.
- Acceptance Filter can provide FullCAN-style automatic reception for selected Standard Identifiers.

## 20.4 Pin description

Table 432. CAN Pin descriptions

Pin Name	Type	Description
CAN1_RD, CAN2_RD	Input	<b>Serial Inputs.</b> From CAN transceivers.
CAN1_TD, CAN2_TD	Output	<b>Serial Outputs.</b> To CAN transceivers.

## 20.5 CAN controller architecture

The CAN Controller is a complete serial interface with both Transmit and Receive Buffers but without Acceptance Filter. CAN Identifier filtering is done for all CAN channels in a separate block (Acceptance Filter). Except for message buffering and acceptance filtering the functionality is similar to the PeliCAN concept.

The CAN Controller Block includes interfaces to the following blocks:

- APB Interface
- Acceptance Filter
- Nested Vectored Interrupt Controller (NVIC)
- CAN Transceiver
- Common Status Registers

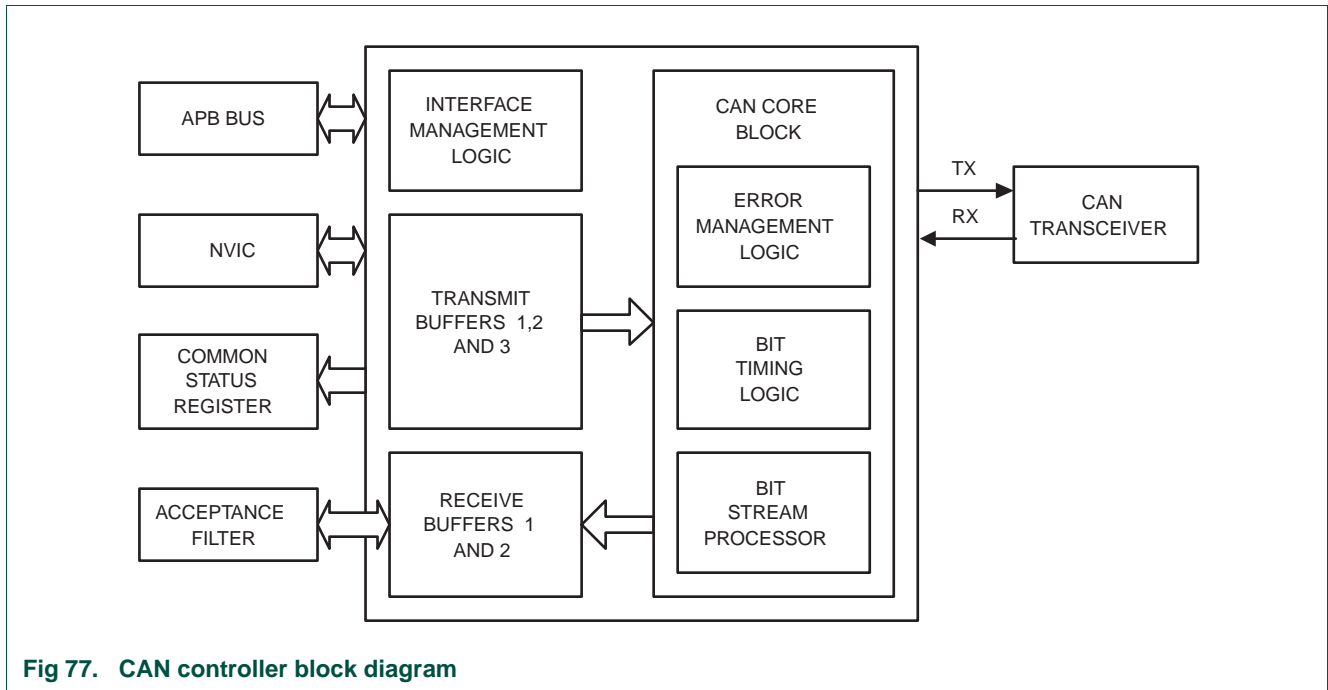


Fig 77. CAN controller block diagram

### 20.5.1 APB Interface Block (AIB)

The APB Interface Block provides access to all CAN Controller registers.

### 20.5.2 Interface Management Logic (IML)

The Interface Management Logic interprets commands from the CPU, controls internal addressing of the CAN Registers and provides interrupts and status information to the CPU.

### 20.5.3 Transmit Buffers (TXB)

The TXB represents a Triple Transmit Buffer, which is the interface between the Interface Management Logic (IML) and the Bit Stream Processor (BSP). Each Transmit Buffer is able to store a complete message which can be transmitted over the CAN network. This buffer is written by the CPU and read out by the BSP.

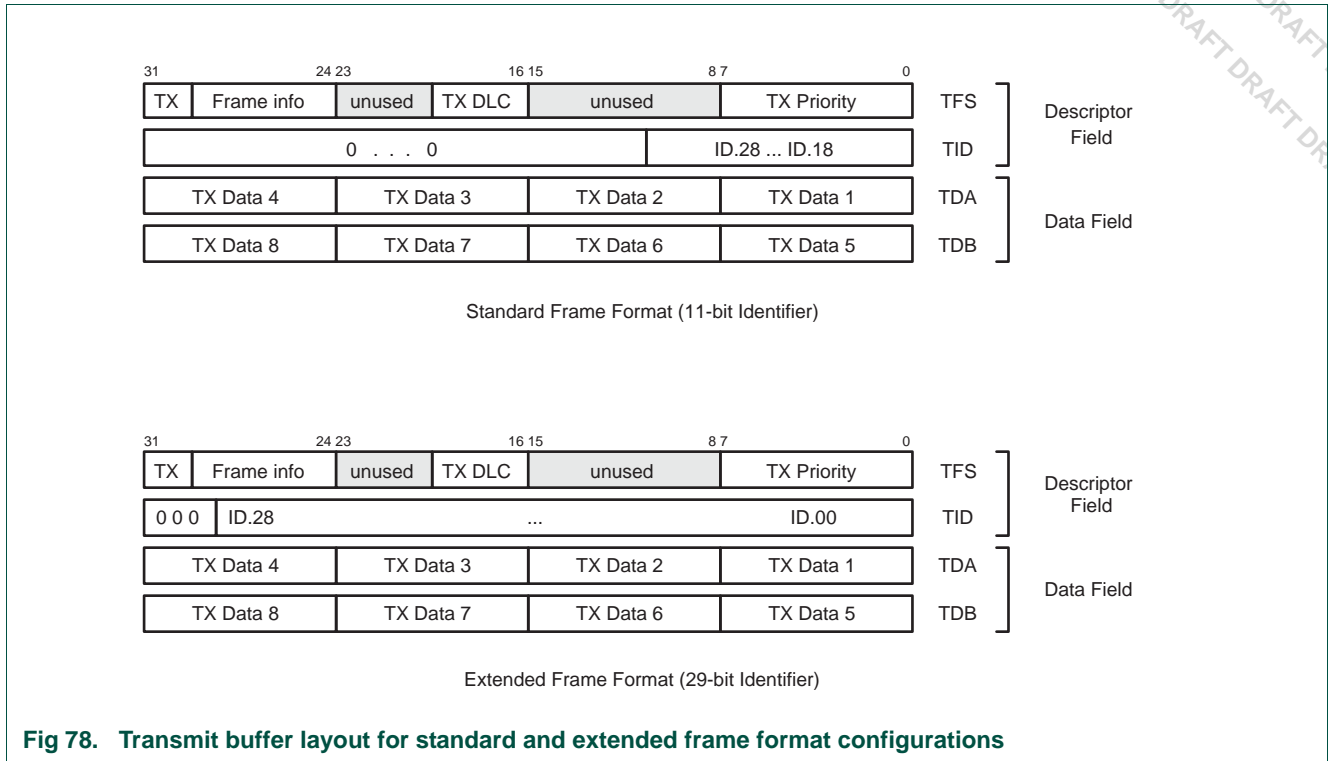


Fig 78. Transmit buffer layout for standard and extended frame format configurations

### 20.5.4 Receive Buffer (RXB)

The Receive Buffer (RXB) represents a CPU accessible Double Receive Buffer. It is located between the CAN Controller Core Block and APB Interface Block and stores all received messages from the CAN Bus line. With the help of this Double Receive Buffer concept the CPU is able to process one message while another message is being received.

The global layout of the Receive Buffer is very similar to the Transmit Buffer described earlier. Identifier, Frame Format, Remote Transmission Request bit and Data Length Code have the same meaning as described for the Transmit Buffer. In addition, the Receive Buffer includes an ID Index field (see [Section 20.7.9.1 "ID index field"](#)).

The received Data Length Code represents the real transmitted Data Length Code, which may be greater than 8 depending on transmitting CAN node. Nevertheless, the maximum number of received data bytes is 8. This should be taken into account by reading a message from the Receive Buffer. If there is not enough space for a new message within the Receive Buffer, the CAN Controller generates a Data Overrun condition when this message becomes valid and the acceptance test was positive. A message that is partly written into the Receive Buffer (when the Data Overrun situation occurs) is deleted. This situation is signalled to the CPU via the Status Register and the Data Overrun Interrupt, if enabled.

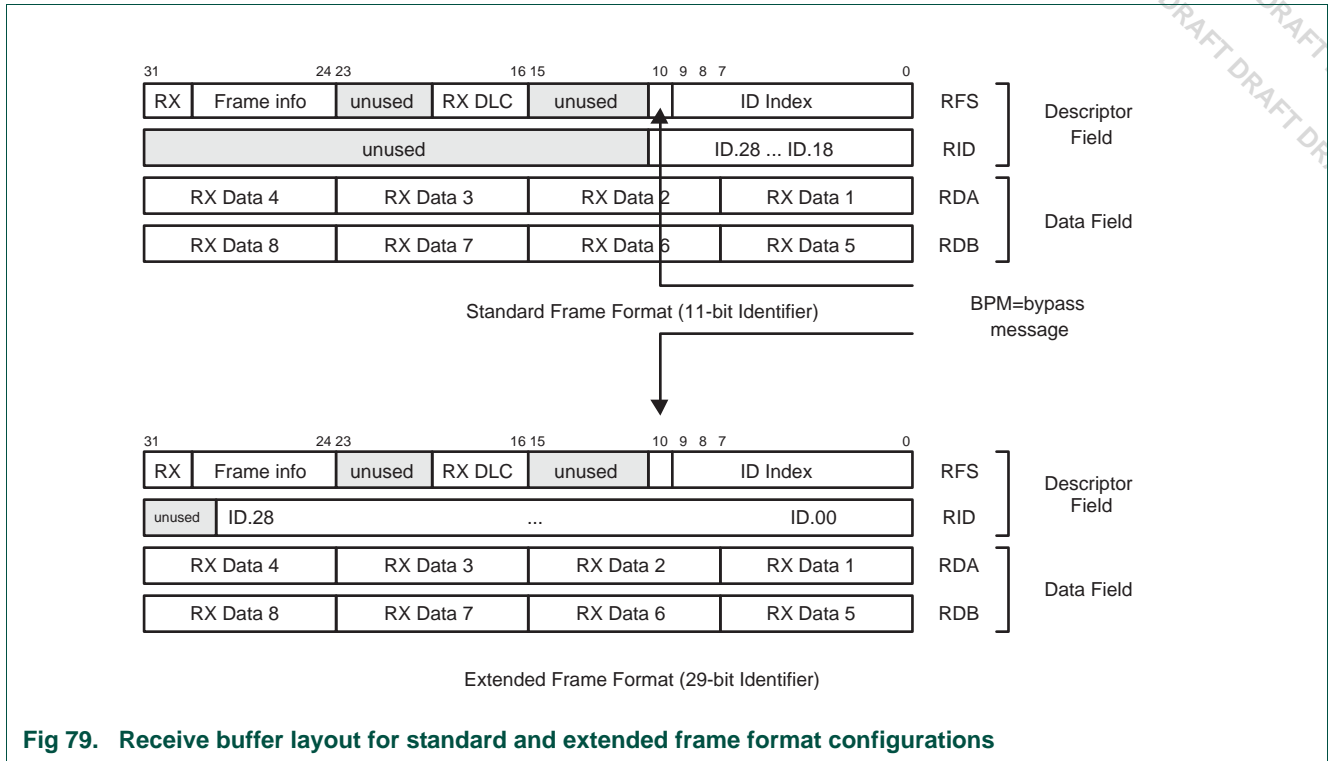


Fig 79. Receive buffer layout for standard and extended frame format configurations

### 20.5.5 Error Management Logic (EML)

The EML is responsible for the error confinement. It gets error announcements from the BSP and then informs the BSP and IML about error statistics.

### 20.5.6 Bit Timing Logic (BTL)

The Bit Timing Logic monitors the serial CAN Bus line and handles the Bus line related bit timing. It synchronizes to the bit stream on the CAN Bus on a "recessive" to "dominant" Bus line transition at the beginning of a message (hard synchronization) and re-synchronizes on further transitions during the reception of a message (soft synchronization). The BTL also provides programmable time segments to compensate for the propagation delay times and phase shifts (e.g. due to oscillator drifts) and to define the sample point and the number of samples to be taken within a bit time.

### 20.5.7 Bit Stream Processor (BSP)

The Bit Stream Processor is a sequencer, controlling the data stream between the Transmit Buffer, Receive Buffers and the CAN Bus. It also performs the error detection, arbitration, stuffing and error handling on the CAN Bus.

### 20.5.8 CAN controller self-tests

The CAN controller supports two different options for self-tests:

- Global Self-Test (setting the self reception request bit in normal Operating Mode)
- Local Self-Test (setting the self reception request bit in Self Test Mode)

Both self-tests are using the 'Self Reception' feature of the CAN Controller. With the Self Reception Request, the transmitted message is also received and stored in the receive buffer. Therefore the acceptance filter has to be configured accordingly. As soon as the CAN message is transmitted, a transmit and a receive interrupt are generated, if enabled.

**Global self test**

A Global Self-Test can for example be used to verify the chosen configuration of the CAN Controller in a given CAN system. As shown in Figure 80, at least one other CAN node, which is acknowledging each CAN message has to be connected to the CAN bus.

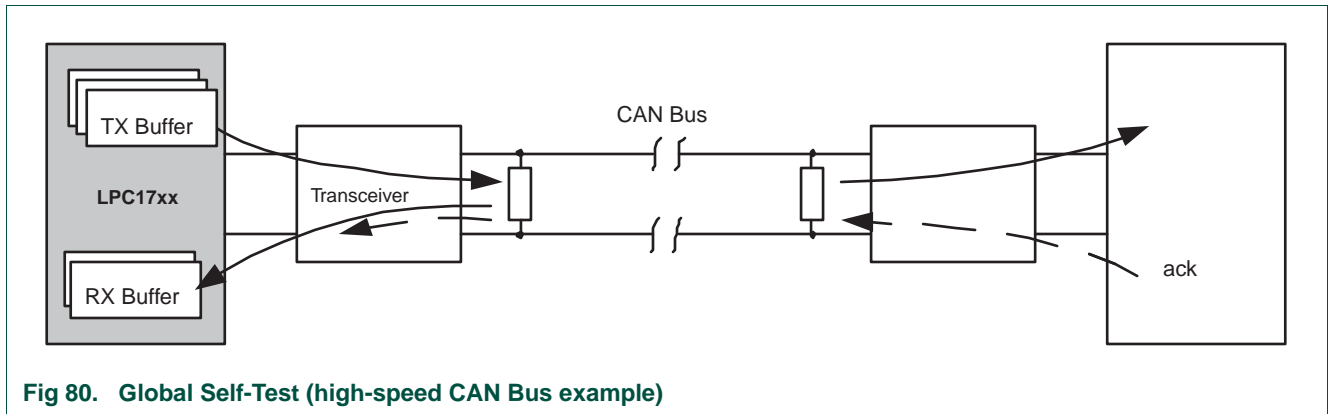


Fig 80. Global Self-Test (high-speed CAN Bus example)

Initiating a Global Self-Test is similar to a normal CAN transmission. In this case the transmission of a CAN message(s) is initiated by setting Self Reception Request bit (SRR) in conjunction with the selected Message Buffer bits (STB3, STB2, STB1) in the CAN Controller Command register (CANCMR).

**Local self test**

The Local Self-Test perfectly fits for single node tests. In this case an acknowledge from other nodes is not needed. As shown in the Figure below, a CAN transceiver with an appropriate CAN bus termination has to be connected to the LPC178x/177x. The CAN Controller has to be put into the 'Self Test Mode' by setting the STM bit in the CAN Controller Mode register (CANMOD). Hint: Setting the Self Test Mode bit (STM) is possible only when the CAN Controller is in Reset Mode.

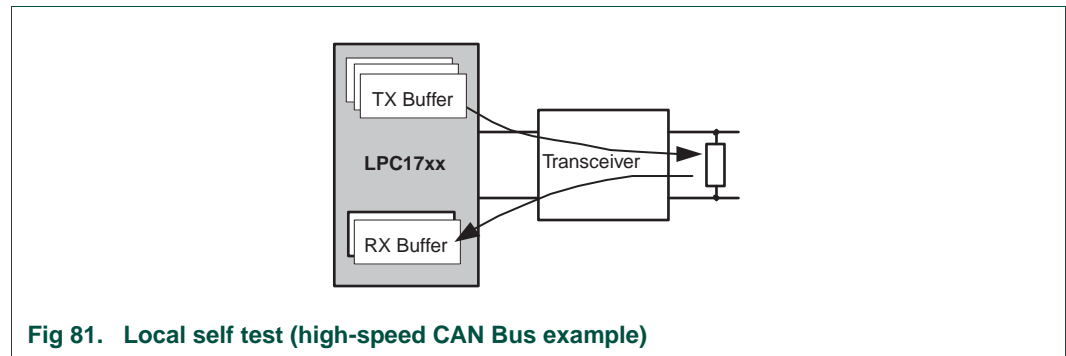


Fig 81. Local self test (high-speed CAN Bus example)

A message transmission is initiated by setting Self Reception Request bit (SRR) in conjunction with the selected Message Buffer(s) (STB3, STB2, STB1).

## 20.6 Memory map of the CAN block

The CAN Controllers and Acceptance Filter occupy a number of APB slots, as follows:

**Table 433. Memory map of the CAN block**

Address Range	Used for
0x4003 8000 - 0x4003 87FF	Acceptance Filter RAM.
0x4003 C000 - 0x4003 C017	Acceptance Filter Registers.
0x4004 0000 - 0x4004 000B	Central CAN Registers.
0x4004 4000 - 0x4004 405F	CAN Controller 1 Registers.
0x4004 8000 - 0x4004 805F	CAN Controller 2 Registers.
0x400F C110 - 0x400F C114	CAN Wake and Sleep Registers.

## 20.7 CAN controller registers

CAN block implements the registers shown in [Table 434](#) and [Table 435](#). More detailed descriptions follow.

**Table 434. CAN acceptance filter and central CAN registers**

Name	Description	Access	Reset Value	Address	Table
AFMR	Acceptance Filter Register	R/W	1	0x4003 C000	<a href="#">463</a>
SFF_sa	Standard Frame Individual Start Address Register	R/W	0	0x4003 C004	<a href="#">464</a>
SFF_GRP_sa	Standard Frame Group Start Address Register	R/W	0	0x4003 C008	<a href="#">465</a>
EFF_sa	Extended Frame Start Address Register	R/W	0	0x4003 C00C	<a href="#">466</a>
EFF_GRP_sa	Extended Frame Group Start Address Register	R/W	0	0x4003 C010	<a href="#">467</a>
ENDofTable	End of AF Tables register	R/W	0	0x4003 C014	<a href="#">468</a>
LUTerrAd	LUT Error Address register	RO	0	0x4003 C018	<a href="#">469</a>
LUTerr	LUT Error Register	RO	0	0x4003 C01C	<a href="#">470</a>
CANTxSR	CAN Central Transmit Status Register	RO	0x0003 0300	0x4004 0000	<a href="#">458</a>
CANRxSR	CAN Central Receive Status Register	RO	0	0x4004 0004	<a href="#">459</a>
CANMSR	CAN Central Miscellaneous Register	RO	0	0x4004 0008	<a href="#">460</a>

**Table 435. CAN1 and CAN2 controller register map**

Generic Name	Description	Access	Reset value	CAN1 & 2 Register Name & Address	Table
MOD	Controls the operating mode of the CAN Controller.	R/W	1	CAN1MOD - 0x4004 4000 CAN2MOD - 0x4004 8000	<a href="#">438</a>
CMR	Command bits that affect the state of the CAN Controller	WO	0	CAN1CMR - 0x4004 4004 CAN2CMR - 0x4004 8004	<a href="#">439</a>
GSR	Global Controller Status and Error Counters	RO <sup>[1]</sup>	0x3C	CAN1GSR - 0x4004 4008 CAN2GSR - 0x4004 8008	<a href="#">440</a>
ICR	Interrupt status, Arbitration Lost Capture, Error Code Capture	RO	0	CAN1ICR - 0x4004 400C CAN2ICR - 0x4004 800C	<a href="#">441</a>
IER	Interrupt Enable	R/W	0	CAN1IER - 0x4004 4010 CAN2IER - 0x4004 8010	<a href="#">442</a>



Table 435. CAN1 and CAN2 controller register map

Generic Name	Description	Access	Reset value	CAN1 & 2 Register Name & Address	Table
BTR	Bus Timing	R/W <sup>[2]</sup>	0x1C0000	CAN1BTR - 0x4004 4014 CAN2BTR - 0x4004 8014	<a href="#">443</a>
EWL	Error Warning Limit	R/W <sup>[2]</sup>	0x60	CAN1EWL - 0x4004 4018 CAN2EWL - 0x4004 8018	<a href="#">444</a>
SR	Status Register	RO	0x3C3C3C	CAN1SR - 0x4004 401C CAN2SR - 0x4004 801C	<a href="#">445</a>
RFS	Receive frame status	R/W <sup>[2]</sup>	0	CAN1RFS - 0x4004 4020 CAN2RFS - 0x4004 8020	<a href="#">446</a>
RID	Received Identifier	R/W <sup>[2]</sup>	0	CAN1RID - 0x4004 4024 CAN2RID - 0x4004 8024	<a href="#">447</a>
RDA	Received data bytes 1-4	R/W <sup>[2]</sup>	0	CAN1RDA - 0x4004 4028 CAN2RDA - 0x4004 8028	<a href="#">449</a>
RDB	Received data bytes 5-8	R/W <sup>[2]</sup>	0	CAN1RDB - 0x4004 402C CAN2RDB - 0x4004 802C	<a href="#">450</a>
TFI1	Transmit frame info (Tx Buffer 1)	R/W	0	CAN1TFI1 - 0x4004 4030 CAN2TFI1 - 0x4004 8030	<a href="#">451</a>
TID1	Transmit Identifier (Tx Buffer 1)	R/W	0	CAN1TID1 - 0x4004 4034 CAN2TID1 - 0x4004 8034	<a href="#">452</a>
TDA1	Transmit data bytes 1-4 (Tx Buffer 1)	R/W	0	CAN1TDA1 - 0x4004 4038 CAN2TDA1 - 0x4004 8038	<a href="#">454</a>
TDB1	Transmit data bytes 5-8 (Tx Buffer 1)	R/W	0	CAN1TDB1 - 0x4004 403C CAN2TDB1 - 0x4004 803C	<a href="#">455</a>
TFI2	Transmit frame info (Tx Buffer 2)	R/W	0	CAN1TFI2 - 0x4004 4040 CAN2TFI2 - 0x4004 8040	<a href="#">451</a>
TID2	Transmit Identifier (Tx Buffer 2)	R/W	0	CAN1TID2 - 0x4004 4044 CAN2TID2 - 0x4004 8044	<a href="#">452</a>
TDA2	Transmit data bytes 1-4 (Tx Buffer 2)	R/W	0	CAN1TDA2 - 0x4004 4048 CAN2TDA2 - 0x4004 8048	<a href="#">454</a>
TDB2	Transmit data bytes 5-8 (Tx Buffer 2)	R/W	0	CAN1TDB2 - 0x4004 404C CAN2TDB2 - 0x4004 804C	<a href="#">455</a>
TFI3	Transmit frame info (Tx Buffer 3)	R/W	0	CAN1TFI3 - 0x4004 4050 CAN2TFI3 - 0x4004 8050	<a href="#">451</a>
TID3	Transmit Identifier (Tx Buffer 3)	R/W	0	CAN1TID3 - 0x4004 4054 CAN2TID3 - 0x4004 8054	<a href="#">452</a>
TDA3	Transmit data bytes 1-4 (Tx Buffer 3)	R/W	0	CAN1TDA3 - 0x4004 4058 CAN2TDA3 - 0x4004 8058	<a href="#">454</a>
TDB3	Transmit data bytes 5-8 (Tx Buffer 3)	R/W	0	CAN1TDB3 - 0x4004 405C CAN2TDB3 - 0x4004 805C	<a href="#">455</a>

[1] The error counters can only be written when RM in CANMOD is 1.

[2] These registers can only be written when RM in CANMOD is 1.

The internal registers of each CAN Controller appear to the CPU as on-chip memory mapped peripheral registers. Because the CAN Controller can operate in different modes (Operating/Reset, see also [Section 20.7.1 “CAN Mode register \(CAN1MOD - 0x4004 4000, CAN2MOD - 0x4004 8000\)”](#)), one has to distinguish between different internal address definitions. Note that write access to some registers is only allowed in Reset Mode.

**Table 436. CAN1 and CAN2 controller register summary**

Generic Name	Operating Mode		Reset Mode	
	Read	Write	Read	Write
MOD	Mode	Mode	Mode	Mode
CMR	0x00	Command	0x00	Command
GSR	Global Status and Error Counters	-	Global Status and Error Counters	Error Counters only
ICR	Interrupt and Capture	-	Interrupt and Capture	-
IER	Interrupt Enable	Interrupt Enable	Interrupt Enable	Interrupt Enable
BTR	Bus Timing	-	Bus Timing	Bus Timing
EWL	Error Warning Limit	-	Error Warning Limit	Error Warning Limit
SR	Status	-	Status	-
RFS	Rx Info and Index	-	Rx Info and Index	Rx Info and Index
RID	Rx Identifier	-	Rx Identifier	Rx Identifier
RDA	Rx Data	-	Rx Data	Rx Data
RDB	Rx Info and Index	-	Rx Info and Index	Rx Info and Index
TFI1	Tx Info1	Tx Info	Tx Info	Tx Info
TID1	Tx Identifier	Tx Identifier	Tx Identifier	Tx Identifier
TDA1	Tx Data	Tx Data	Tx Data	Tx Data
TDB1	Tx Data	Tx Data	Tx Data	Tx Data

**Table 437. CAN Wake and Sleep registers**

Name	Description	Access	Reset Value	Address	Table
CANSLEEPCLR	Allows clearing the current CAN channel sleep state as well as reading that state.	R/W	0	0x400F C110	<a href="#">456</a>
CANWAKEFLAGS	Allows reading the wake-up state of the CAN channels.	R/W	0	0x400F C114	<a href="#">457</a>

In the following register tables, the column “Reset Value” shows how a hardware reset affects each bit or field, while the column “RM Set” indicates how each bit or field is affected if software sets the RM bit, or RM is set because of a Bus-Off condition. Note that while hardware reset sets RM, in this case the setting noted in the “Reset Value” column prevails over that shown in the “RM Set” column, in the few bits where they differ. In both columns, X indicates the bit or field is unchanged.

### 20.7.1 CAN Mode register (CAN1MOD - 0x4004 4000, CAN2MOD - 0x4004 8000)

The contents of the Mode Register are used to change the behavior of the CAN Controller. Bits may be set or reset by the CPU that uses the Mode Register as a read/write memory.

Table 438. CAN Mode register (CAN1MOD - address 0x4004 4000, CAN2MOD - address 0x4004 8000) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RM <sup>[1][6]</sup>		Reset Mode.	1	1
		0 (normal)	The CAN Controller is in the Operating Mode, and certain registers can not be written.		
		1 (reset)	CAN operation is disabled, writable registers can be written and the current transmission/reception of a message is aborted.		
1	LOM <sup>[3][2][6]</sup>		Listen Only Mode.	0	x
		0 (normal)	The CAN controller acknowledges a successfully received message on the CAN bus. The error counters are stopped at the current value.		
		1 (listen only)	The controller gives no acknowledgment, even if a message is successfully received. Messages cannot be sent, and the controller operates in "error passive" mode. This mode is intended for software bit rate detection and "hot plugging".		
2	STM <sup>[3][6]</sup>		Self Test Mode.	0	x
		0 (normal)	A transmitted message must be acknowledged to be considered successful.		
		1 (self test)	The controller will consider a Tx message successful even if there is no acknowledgment received.  In this mode a full node test is possible without any other active node on the bus using the SRR bit in CANxCMR.		
3	TPM <sup>[4]</sup>		Transmit Priority Mode.	0	x
		0 (CAN ID)	The transmit priority for 3 Transmit Buffers depends on the CAN Identifier.		
		1 (local priority)	The transmit priority for 3 Transmit Buffers depends on the contents of the Tx Priority register within the Transmit Buffer.		
4	SM <sup>[5]</sup>		Sleep Mode.	0	0
		0 (wake-up)	Normal operation.		
		1 (sleep)	The CAN controller enters Sleep Mode if no CAN interrupt is pending and there is no bus activity. See the Sleep Mode description <a href="#">Section 20.8.2 on page 525</a> .		
5	RPM		Receive Polarity Mode.	0	x
		0 (low active)	RD input is active Low (dominant bit = 0).		
		1 (high active)	RD input is active High (dominant bit = 1) -- reverse polarity.		
6	-		Reserved. Read value is undefined, only zero should be written.	0	0
7	TM		Test Mode.	0	x
		0 (disabled)	Normal operation.		
		1 (enabled)	The TD pin will reflect the bit, detected on RD pin, with the next positive edge of the system clock.		
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA	

- [1] During a Hardware reset or when the Bus Status bit is set '1' (Bus-Off), the Reset Mode bit is set '1' (present). After the Reset Mode bit is set '0' the CAN Controller will wait for:
  - one occurrence of Bus-Free signal (11 recessive bits), if the preceding reset has been caused by a Hardware reset or a CPU-initiated reset.
  - 128 occurrences of Bus-Free, if the preceding reset has been caused by a CAN Controller initiated Bus-Off, before re-entering the Bus-On mode.
- [2] This mode of operation forces the CAN Controller to be error passive. Message Transmission is not possible. The Listen Only Mode can be used e.g. for software driven bit rate detection and "hot plugging".
- [3] A write access to the bits MOD.1 and MOD.2 is possible only if the Reset Mode is entered previously.

- [4] Transmit Priority Mode is explained in more detail in [Section 20.5.3 “Transmit Buffers \(TXB\)”](#).
- [5] The CAN Controller will enter Sleep Mode, if the Sleep Mode bit is set '1' (sleep), there is no bus activity, and none of the CAN interrupts is pending. Setting of SM with at least one of the previously mentioned exceptions valid will result in a wake-up interrupt. The CAN Controller will wake up if SM is set LOW (wake-up) or there is bus activity. On wake-up, a Wake-up Interrupt is generated. A sleeping CAN Controller which wakes up due to bus activity will not be able to receive this message until it detects 11 consecutive recessive bits (Bus-Free sequence). Note that setting of SM is not possible in Reset Mode. After clearing of Reset Mode, setting of SM is possible only when Bus-Free is detected again.
- [6] The LOM and STM bits can only be written if the RM bit is 1 prior to the write operation.

### 20.7.2 CAN Command Register (CAN1CMR - 0x4004 x004, CAN2CMR - 0x4004 8004)

Writing to this write-only register initiates an action within the transfer layer of the CAN Controller. Reading this register yields zeroes.

At least one internal clock cycle is needed for processing between two commands.

**Table 439. CAN Command Register (CAN1CMR - address 0x4004 4004, CAN2CMR - address 0x4004 8004) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
0 <sup>[1][2]</sup>	TR		Transmission Request.	0	0
		0 (absent)	No transmission request.		
		1 (present)	The message, previously written to the CANxTFI, CANxTID, and optionally the CANxTDA and CANxTDB registers, is queued for transmission from the selected Transmit Buffer. If at two or all three of STB1, STB2 and STB3 bits are selected when TR=1 is written, Transmit Buffer will be selected based on the chosen priority scheme (for details see <a href="#">Section 20.5.3 “Transmit Buffers (TXB)”</a> )		
1 <sup>[1][3]</sup>	AT		Abort Transmission.	0	0
		0 (no action)	Do not abort the transmission.		
		1 (present)	if not already in progress, a pending Transmission Request for the selected Transmit Buffer is cancelled.		
2 <sup>[4]</sup>	RRB		Release Receive Buffer.	0	0
		0 (no action)	Do not release the receive buffer.		
		1 (released)	The information in the Receive Buffer (consisting of CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers) is released, and becomes eligible for replacement by the next received frame. If the next received frame is not available, writing this command clears the RBS bit in the Status Register(s).		
3 <sup>[5]</sup>	CDO		Clear Data Overrun.	0	0
		0 (no action)	Do not clear the data overrun bit.		
		1 (clear)	The Data Overrun bit in Status Register(s) is cleared.		
4 <sup>[1][6]</sup>	SRR		Self Reception Request.	0	0
		0 (absent)	No self reception request.		
		1 (present)	The message, previously written to the CANxTFS, CANxTID, and optionally the CANxTDA and CANxTDB registers, is queued for transmission from the selected Transmit Buffer and received simultaneously. This differs from the TR bit above in that the receiver is not disabled during the transmission, so that it receives the message if its Identifier is recognized by the Acceptance Filter.		

**Table 439. CAN Command Register (CAN1CMR - address 0x4004 4004, CAN2CMR - address 0x4004 8004) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
5	STB1		Select Tx Buffer 1.	0	0
		0 (not selected)	Tx Buffer 1 is not selected for transmission.		
		1 (selected)	Tx Buffer 1 is selected for transmission.		
6	STB2		Select Tx Buffer 2.	0	0
		0 (not selected)	Tx Buffer 2 is not selected for transmission.		
		1 (selected)	Tx Buffer 2 is selected for transmission.		
7	STB3		Select Tx Buffer 3.	0	0
		0 (not selected)	Tx Buffer 3 is not selected for transmission.		
		1 (selected)	Tx Buffer 3 is selected for transmission.		
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA	

- [1] - Setting the command bits TR and AT simultaneously results in transmitting a message once. No re-transmission will be performed in case of an error or arbitration lost (single shot transmission).  
 - Setting the command bits SRR and TR simultaneously results in sending the transmit message once using the self-reception feature. No re-transmission will be performed in case of an error or arbitration lost.  
 - Setting the command bits TR, AT and SRR simultaneously results in transmitting a message once as described for TR and AT. The moment the Transmit Status bit is set within the Status Register, the internal Transmission Request Bit is cleared automatically.  
 - Setting TR and SRR simultaneously will ignore the set SRR bit.
- [2] If the Transmission Request or the Self-Reception Request bit was set '1' in a previous command, it cannot be cancelled by resetting the bits. The requested transmission may only be cancelled by setting the Abort Transmission bit.
- [3] The Abort Transmission bit is used when the CPU requires the suspension of the previously requested transmission, e.g. to transmit a more urgent message before. A transmission already in progress is not stopped. In order to see if the original message has been either transmitted successfully or aborted, the Transmission Complete Status bit should be checked. This should be done after the Transmit Buffer Status bit has been set to '1' or a Transmit Interrupt has been generated.
- [4] After reading the contents of the Receive Buffer, the CPU can release this memory space by setting the Release Receive Buffer bit '1'. This may result in another message becoming immediately available. If there is no other message available, the Receive Interrupt bit is reset. If the RRB command is given, it will take at least 2 internal clock cycles before a new interrupt is generated.
- [5] This command bit is used to clear the Data Overrun condition signalled by the Data Overrun Status bit. As long as the Data Overrun Status bit is set no further Data Overrun Interrupt is generated.
- [6] Upon Self Reception Request, a message is transmitted and simultaneously received if the Acceptance Filter is set to the corresponding identifier. A receive and a transmit interrupt will indicate correct self reception (see also Self Test Mode in [Section 20.7.1 "CAN Mode register \(CAN1MOD - 0x4004 4000, CAN2MOD - 0x4004 8000\)"](#)).

### 20.7.3 CAN Global Status Register (CAN1GSR - 0x4004 x008, CAN2GSR - 0x4004 8008)

The content of the Global Status Register reflects the status of the CAN Controller. This register is read-only, except that the Error Counters can be written when the RM bit in the CANMOD register is 1. Bits not listed read as 0 and should be written as 0.

**Table 440. CAN Global Status Register (CAN1GSR - address 0x4004 4008, CAN2GSR - address 0x4004 8008) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RBS <sup>[1]</sup>		Receive Buffer Status.	0	0
		0 (empty)	No message is available.		
		1 (full)	At least one complete message is received by the Double Receive Buffer and available in the CANxRFS, CANxRID, and if applicable the CANxRDA and CANxRDB registers. This bit is cleared by the Release Receive Buffer command in CANxCMR, if no subsequent received message is available.		
1	DOS <sup>[2]</sup>		Data Overrun Status.	0	0
		0 (absent)	No data overrun has occurred since the last Clear Data Overrun command was given/written to CANxCMR (or since Reset).		
		1 (overrun)	A message was lost because the preceding message to this CAN controller was not read and released quickly enough (there was not enough space for a new message in the Double Receive Buffer).		
2	TBS		Transmit Buffer Status.	1	1
		0 (locked)	At least one of the Transmit Buffers is not available for the CPU, i.e. at least one previously queued message for this CAN controller has not yet been sent, and therefore software should not write to the CANxTFI, CANxTID, CANxTDA, nor CANxTDB registers of that (those) Tx buffer(s).		
		1 (released)	All three Transmit Buffers are available for the CPU. No transmit message is pending for this CAN controller (in any of the 3 Tx buffers), and software may write to any of the CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		
3	TCS <sup>[3]</sup>		Transmit Complete Status.	1	x
		0 (incomplete)	At least one requested transmission has not been successfully completed yet.		
		1 (complete)	All requested transmission(s) has (have) been successfully completed.		
4	RS <sup>[4]</sup>		Receive Status.	1	0
		0 (idle)	The CAN controller is idle.		
		1 (receive)	The CAN controller is receiving a message.		
5	TS <sup>[4]</sup>		Transmit Status.	1	0
		0 (idle)	The CAN controller is idle.		
		1 (transmit)	The CAN controller is sending a message.		
6	ES <sup>[5]</sup>		Error Status.	0	0
		0 (ok)	Both error counters are below the Error Warning Limit.		
		1 (error)	One or both of the Transmit and Receive Error Counters has reached the limit set in the Error Warning Limit register.		
7	BS <sup>[6]</sup>		Bus Status.	0	0
		0 (Bus-On)	The CAN Controller is involved in bus activities		
		1 (Bus-Off)	The CAN controller is currently not involved/prohibited from bus activity because the Transmit Error Counter reached its limiting value of 255.		
15:8	-		Reserved. Read value is undefined, only zero should be written.	NA	
23:16	RXERR	-	The current value of the Rx Error Counter (an 8-bit value).	0	X
31:24	TXERR	-	The current value of the Tx Error Counter (an 8-bit value).	0	X

[1] After reading all messages and releasing their memory space with the command 'Release Receive Buffer,' this bit is cleared.

- [2] If there is not enough space to store the message within the Receive Buffer, that message is dropped and the Data Overrun condition is signalled to the CPU in the moment this message becomes valid. If this message is not completed successfully (e.g. because of an error), no overrun condition is signalled.
- [3] The Transmission Complete Status bit is set '0' (incomplete) whenever the Transmission Request bit or the Self Reception Request bit is set '1' at least for one of the three Transmit Buffers. The Transmission Complete Status bit will remain '0' until all messages are transmitted successfully.
- [4] If both the Receive Status and the Transmit Status bits are '0' (idle), the CAN-Bus is idle. If both bits are set, the controller is waiting to become idle again. After hardware reset 11 consecutive recessive bits have to be detected until idle status is reached. After Bus-off this will take 128 times of 11 consecutive recessive bits.
- [5] Errors detected during reception or transmission will effect the error counters according to the CAN specification. The Error Status bit is set when at least one of the error counters has reached or exceeded the Error Warning Limit. An Error Warning Interrupt is generated, if enabled. The default value of the Error Warning Limit after hardware reset is 96 decimal, see also [Section 20.7.7 "CAN Error Warning Limit register \(CAN1EWL - 0x4004 4018, CAN2EWL - 0x4004 8018\)"](#).
- [6] Mode bit '1' (present) and an Error Warning Interrupt is generated, if enabled. Afterwards the Transmit Error Counter is set to '127', and the Receive Error Counter is cleared. It will stay in this mode until the CPU clears the Reset Mode bit. Once this is completed the CAN Controller will wait the minimum protocol-defined time (128 occurrences of the Bus-Free signal) counting down the Transmit Error Counter. After that, the Bus Status bit is cleared (Bus-On), the Error Status bit is set '0' (ok), the Error Counters are reset, and an Error Warning Interrupt is generated, if enabled. Reading the TX Error Counter during this time gives information about the status of the Bus-Off recovery.

### RX error counter

The RX Error Counter Register, which is part of the Status Register, reflects the current value of the Receive Error Counter. After hardware reset this register is initialized to 0. In Operating Mode this register appears to the CPU as a read-only memory. A write access to this register is possible only in Reset Mode. If a Bus Off event occurs, the RX Error Counter is initialized to 0. As long as Bus Off is valid, writing to this register has no effect. The Rx Error Counter is determined as follows:

$$\text{RX Error Counter} = (\text{CANxGSR AND } 0x00FF0000) / 0x00010000$$

Note that a CPU-forced content change of the RX Error Counter is possible only if the Reset Mode was entered previously. An Error Status change (Status Register), an Error Warning or an Error Passive Interrupt forced by the new register content will not occur until the Reset Mode is cancelled again.

### TX error counter

The TX Error Counter Register, which is part of the Status Register, reflects the current value of the Transmit Error Counter. In Operating Mode this register appears to the CPU as a read-only memory. After hardware reset this register is initialized to 0. A write access to this register is possible only in Reset Mode. If a bus-off event occurs, the TX Error Counter is initialized to 127 to count the minimum protocol-defined time (128 occurrences of the Bus-Free signal). Reading the TX Error Counter during this time gives information about the status of the Bus-Off recovery. If Bus Off is active, a write access to TXERR in the range of 0 to 254 clears the Bus Off Flag and the controller will wait for one occurrence of 11 consecutive recessive bits (bus free) after clearing of Reset Mode. The Tx error counter is determined as follows:

$$\text{TX Error Counter} = (\text{CANxGSR AND } 0xFF000000) / 0x01000000$$

Writing 255 to TXERR allows initiation of a CPU-driven Bus Off event. Note that a CPU-forced content change of the TX Error Counter is possible only if the Reset Mode was entered previously. An Error or Bus Status change (Status Register), an Error Warning, or an Error Passive Interrupt forced by the new register content will not occur

until the Reset Mode is cancelled again. After leaving the Reset Mode, the new TX Counter content is interpreted and the Bus Off event is performed in the same way as if it was forced by a bus error event. That means, that the Reset Mode is entered again, the TX Error Counter is initialized to 127, the RX Counter is cleared, and all concerned Status and Interrupt Register bits are set. Clearing of Reset Mode now will perform the protocol defined Bus Off recovery sequence (waiting for 128 occurrences of the Bus-Free signal). If the Reset Mode is entered again before the end of Bus Off recovery (TXERR>0), Bus Off keeps active and TXERR is frozen.

### 20.7.4 CAN Interrupt and Capture Register (CAN1ICR - 0x4004 400C, CAN2ICR - 0x4004 800C)

Bits in this register indicate information about events on the CAN bus. This register is read-only.

The Interrupt flags of the Interrupt and Capture Register allow the identification of an interrupt source. When one or more bits are set, a CAN interrupt will be indicated to the CPU. After this register is read from the CPU all interrupt bits are reset **except** of the Receive Interrupt bit. The Interrupt Register appears to the CPU as a read-only memory.

Bits 1 through 10 clear when they are read.

Bits 16-23 are captured when a bus error occurs. At the same time, if the BEIE bit in CANIER is 1, the BEI bit in this register is set, and a CAN interrupt can occur.

Bits 24-31 are captured when CAN arbitration is lost. At the same time, if the ALIE bit in CANIER is 1, the ALI bit in this register is set, and a CAN interrupt can occur. Once either of these bytes is captured, its value will remain the same until it is read, at which time it is released to capture a new value.

The clearing of bits 1 to 10 and the releasing of bits 16-23 and 24-31 all occur on any read from CANxICR, regardless of whether part or all of the register is read. This means that software should always read CANxICR as a word, and process and deal with all bits of the register as appropriate for the application.

**Table 441. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RI <sup>[1]</sup>	0 (reset) 1 (set)	Receive Interrupt. This bit is set whenever the RBS bit in CANxSR and the RIE bit in CANxIER are both 1, indicating that a new message was received and stored in the Receive Buffer.	0	0
1	TI1	0 (reset) 1 (set)	Transmit Interrupt 1. This bit is set when the TBS1 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB1 was successfully transmitted or aborted), indicating that Transmit buffer 1 is available, and the TIE1 bit in CANxIER is 1.	0	0
2	EI	0 (reset) 1 (set)	Error Warning Interrupt. This bit is set on every change (set or clear) of either the Error Status or Bus Status bit in CANxSR and the EIE bit bit is set within the Interrupt Enable Register at the time of the change.	0	X
3	DOI	0 (reset) 1 (set)	Data Overrun Interrupt. This bit is set when the DOS bit in CANxSR goes from 0 to 1 and the DOIE bit in CANxIER is 1.	0	0
4	WUI <sup>[2]</sup>	0 (reset) 1 (set)	Wake-Up Interrupt. This bit is set if the CAN controller is sleeping and bus activity is detected and the WUIE bit in CANxIER is 1.	0	0



**Table 441. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
5	EPI	0 (reset) 1 (set)	Error Passive Interrupt. This bit is set if the EPIE bit in CANxIER is 1, and the CAN controller switches between Error Passive and Error Active mode in either direction.  This is the case when the CAN Controller has reached the Error Passive Status (at least one error counter exceeds the CAN protocol defined level of 127) or if the CAN Controller is in Error Passive Status and enters the Error Active Status again.	0	0
6	ALI	0 (reset) 1 (set)	Arbitration Lost Interrupt. This bit is set if the ALIE bit in CANxIER is 1, and the CAN controller loses arbitration while attempting to transmit. In this case the CAN node becomes a receiver.	0	0
7	BEI	0 (reset) 1 (set)	Bus Error Interrupt -- this bit is set if the BEIE bit in CANxIER is 1, and the CAN controller detects an error on the bus.	0	X
8	IDI	0 (reset) 1 (set)	ID Ready Interrupt -- this bit is set if the IDIE bit in CANxIER is 1, and a CAN Identifier has been received (a message was successfully transmitted or aborted). This bit is set whenever a message was successfully transmitted or aborted and the IDIE bit is set in the IER register.	0	0
9	TI2	0 (reset) 1 (set)	Transmit Interrupt 2. This bit is set when the TBS2 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB2 was successfully transmitted or aborted), indicating that Transmit buffer 2 is available, and the TIE2 bit in CANxIER is 1.	0	0
10	TI3	0 (reset) 1 (set)	Transmit Interrupt 3. This bit is set when the TBS3 bit in CANxSR goes from 0 to 1 (whenever a message out of TXB3 was successfully transmitted or aborted), indicating that Transmit buffer 3 is available, and the TIE3 bit in CANxIER is 1.	0	0
15:11	-		Reserved. The value read from a reserved bit is not defined.	0	0

**Table 441. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
20:16	ERRBIT		Error Code Capture: when the CAN controller detects a bus error, the location of the error within the frame is captured in this field. The value reflects an internal state variable, and as a result is not very linear:	0	X
	4:0				
		00011	Start of Frame		
		00010	ID28 ... ID21		
		00110	ID20 ... ID18		
		00100	SRTR Bit		
		00101	IDE bit		
		00111	ID17 ... 13		
		01111	ID12 ... ID5		
		01110	ID4 ... ID0		
		01100	RTR Bit		
		01101	Reserved Bit 1		
		01001	Reserved Bit 0		
		01011	Data Length Code		
		01010	Data Field		
		01000	CRC Sequence		
		11000	CRC Delimiter		
		11001	Acknowledge Slot		
		11011	Acknowledge Delimiter		
		11010	End of Frame		
		10010	Intermission		
		10001	Active Error Flag		
		10110	Passive Error Flag		
		10011	Tolerate Dominant Bits		
		10111	Error Delimiter		
		11100	Overload flag		
21	ERRDIR		When the CAN controller detects a bus error, the direction of the current bit is captured in this bit.	0	X
		0	Error occurred during transmitting.		
		1	Error occurred during receiving.		
23:22	ERRC1:0		When the CAN controller detects a bus error, the type of error is captured in this field:	0	X
		00	Bit error		
		01	Form error		
		10	Stuff error		
		11	Other error		

**Table 441. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
31:24	ALCBIT <sup>[4]</sup>	-	Each time arbitration is lost while trying to send on the CAN, the bit number within the frame is captured into this field. After the content of ALCBIT is read, the ALI bit is cleared and a new Arbitration Lost interrupt can occur.	0	X
		00	arbitration lost in the first bit (MS) of identifier		
		...			
		11	arbitration lost in SRTS bit (RTR bit for standard frame messages)		
		12	arbitration lost in IDE bit		
		13	arbitration lost in 12th bit of identifier (extended frame only)		
		...			
		30	arbitration lost in last bit of identifier (extended frame only)		
		31	arbitration lost in RTR bit (extended frame only)		

- [1] The Receive Interrupt Bit is not cleared upon a read access to the Interrupt Register. Giving the Command "Release Receive Buffer" will clear RI temporarily. If there is another message available within the Receive Buffer after the release command, RI is set again. Otherwise RI remains cleared.
- [2] A Wake-Up Interrupt is also generated if the CPU tries to set the Sleep bit while the CAN controller is involved in bus activities or a CAN Interrupt is pending. The WUI flag can also get asserted when the according enable bit WUIE is not set. In this case a Wake-Up Interrupt does not get asserted.
- [3] Whenever a bus error occurs, the corresponding bus error interrupt is forced, if enabled. At the same time, the current position of the Bit Stream Processor is captured into the Error Code Capture Register. The content within this register is fixed until the user software has read out its content once. From now on, the capture mechanism is activated again, i.e. reading the CANxICR enables another Bus Error Interrupt.
- [4] On arbitration lost, the corresponding arbitration lost interrupt is forced, if enabled. At that time, the current bit position of the Bit Stream Processor is captured into the Arbitration Lost Capture Register. The content within this register is fixed until the user application has read out its contents once. From now on, the capture mechanism is activated again.

### 20.7.5 CAN Interrupt Enable Register (CAN1IER - 0x4004 4010, CAN2IER - 0x4004 8010)

This read/write register controls whether various events on the CAN controller will result in an interrupt or not. Bits 10:0 in this register correspond 1-to-1 with bits 10:0 in the CANxICR register. If a bit in the CANxIER register is 0 the corresponding interrupt is disabled; if a bit in the CANxIER register is 1 the corresponding source is enabled to trigger an interrupt.

**Table 442. CAN Interrupt Enable Register (CAN1IER - address 0x4004 4010, CAN2IER - address 0x4004 8010) bit description**

Bit	Symbol	Function	Reset Value	RM Set
0	RIE	Receiver Interrupt Enable. When the Receive Buffer Status is 'full', the CAN Controller requests the respective interrupt.	0	X
1	TIE1	Transmit Interrupt Enable for Buffer1. When a message has been successfully transmitted out of TXB1 or Transmit Buffer 1 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
2	EIE	Error Warning Interrupt Enable. If the Error or Bus Status change (see Status Register), the CAN Controller requests the respective interrupt.	0	X

**Table 442. CAN Interrupt Enable Register (CAN1IER - address 0x4004 4010, CAN2IER - address 0x4004 8010) bit description**

Bit	Symbol	Function	Reset Value	RM Set
3	DOIE	Data Overrun Interrupt Enable. If the Data Overrun Status bit is set (see Status Register), the CAN Controller requests the respective interrupt.	0	X
4	WUIE	Wake-Up Interrupt Enable. If the sleeping CAN controller wakes up, the respective interrupt is requested.	0	X
5	EPIE	Error Passive Interrupt Enable. If the error status of the CAN Controller changes from error active to error passive or vice versa, the respective interrupt is requested.	0	X
6	ALIE	Arbitration Lost Interrupt Enable. If the CAN Controller has lost arbitration, the respective interrupt is requested.	0	X
7	BEIE	Bus Error Interrupt Enable. If a bus error has been detected, the CAN Controller requests the respective interrupt.	0	X
8	IDIE	ID Ready Interrupt Enable. When a CAN identifier has been received, the CAN Controller requests the respective interrupt.	0	X
9	TIE2	Transmit Interrupt Enable for Buffer2. When a message has been successfully transmitted out of TXB2 or Transmit Buffer 2 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
10	TIE3	Transmit Interrupt Enable for Buffer3. When a message has been successfully transmitted out of TXB3 or Transmit Buffer 3 is accessible again (e.g. after an Abort Transmission command), the CAN Controller requests the respective interrupt.	0	X
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA	

### 20.7.6 CAN Bus Timing Register (CAN1BTR - 0x4004 4014, CAN2BTR - 0x4004 8014)

This register controls how various CAN timings are derived from the APB clock. It defines the values of the Baud Rate Prescaler (BRP) and the Synchronization Jump Width (SJW). Furthermore, it defines the length of the bit period, the location of the sample point and the number of samples to be taken at each sample point. It can be read at any time but can only be written if the RM bit in CANmod is 1.

**Table 443. CAN Bus Timing Register (CAN1BTR - address 0x4004 4014, CAN2BTR - address 0x4004 8014) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
9:0	BRP		Baud Rate Prescaler. The APB clock is divided by (this value plus one) to produce the CAN clock.	0	X
13:10	-		Reserved. Read value is undefined, only zero should be written.	NA	
15:14	SJW		The Synchronization Jump Width is (this value plus one) CAN clocks.	0	X
19:16	TESG1		The delay from the nominal Sync point to the sample point is (this value plus one) CAN clocks.	1100	X
22:20	TESG2		The delay from the sample point to the next nominal sync point is (this value plus one) CAN clocks. The nominal CAN bit time is (this value plus the value in TSEG1 plus 3) CAN clocks.	001	X

**Table 443. CAN Bus Timing Register (CAN1BTR - address 0x4004 4014, CAN2BTR - address 0x4004 8014) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
23	SAM		Sampling		
		0	The bus is sampled once (recommended for high speed buses)	0	X
		1	The bus is sampled 3 times (recommended for low to medium speed buses to filter spikes on the bus-line)		
31:24	-		Reserved. Read value is undefined, only zero should be written.	NA	

**Baud rate prescaler**

The period of the CAN system clock  $t_{SCL}$  is programmable and determines the individual bit timing. The CAN system clock  $t_{SCL}$  is calculated using the following equation:

$$t_{SCL} = t_{CANsuppliedCLK} \times (BRP + 1) \tag{7}$$

**Synchronization jump width**

To compensate for phase shifts between clock oscillators of different bus controllers, any bus controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width  $t_{SJW}$  defines the maximum number of clock cycles a certain bit period may be shortened or lengthened by one re-synchronization:

$$t_{SJW} = t_{SCL} \times (SJW + 1) \tag{8}$$

**Time segment 1 and time segment 2**

Time segments TSEG1 and TSEG2 determine the number of clock cycles per bit period and the location of the sample point:

$$t_{SYNCSEG} = t_{SCL} \tag{9}$$

$$t_{TSEG1} = t_{SCL} \times (TSEG1 + 1) \tag{10}$$

$$t_{TSEG2} = t_{SCL} \times (TSEG2 + 1) \tag{11}$$

**20.7.7 CAN Error Warning Limit register (CAN1EWL - 0x4004 4018, CAN2EWL - 0x4004 8018)**

This register sets a limit on Tx or Rx errors at which an interrupt can occur. It can be read at any time but can only be written if the RM bit in CANmod is 1.

**Table 444. CAN Error Warning Limit register (CAN1EWL - address 0x4004 4018, CAN2EWL - address 0x4004 8018) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	EWL	During CAN operation, this value is compared to both the Tx and Rx Error Counters. If either of these counter matches this value, the Error Status (ES) bit in CANSR is set.	96 <sub>10</sub> = 0x60	X
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA	

Note that a content change of the Error Warning Limit Register is possible only if the Reset Mode was entered previously. An Error Status change (Status Register) and an Error Warning Interrupt forced by the new register content will not occur until the Reset Mode is cancelled again.

### 20.7.8 CAN Status Register (CAN1SR - 0x4004 401C, CAN2SR - 0x4004 801C)

This read-only register contains three status bytes in which the bits not related to transmission are identical to the corresponding bits in the Global Status Register, while those relating to transmission reflect the status of each of the 3 Tx Buffers.

**Table 445. CAN Status Register (CAN1SR - address 0x4004 401C, CAN2SR - address 0x4004 801C) bit description**

Bit	Symbol	Value	Function	Reset Value	RM Set
0	RBS		Receive Buffer Status. This bit is identical to the RBS bit in the CANxGSR.	0	0
1	DOS		Data Overrun Status. This bit is identical to the DOS bit in the CANxGSR.	0	0
2	TBS1 <sup>[1]</sup>		Transmit Buffer Status 1.	1	1
		0(locke	Software cannot access the Tx Buffer 1 nor write to the corresponding CANxTFI, CANxTID, CANxTDA, and CANxTDB registers because a message is either waiting for transmission or is in transmitting process.		
		1(release	Software may write a message into the Transmit Buffer 1 and its CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		
3	TCS1 <sup>[2]</sup>		Transmission Complete Status.	1	x
		0(incomplete)	The previously requested transmission for Tx Buffer 1 is not complete.		
		1(complete)	The previously requested transmission for Tx Buffer 1 has been successfully completed.		
4	RS		Receive Status. This bit is identical to the RS bit in the GSR.	1	0
5	TS1		Transmit Status 1.	1	0
		0(idle)	There is no transmission from Tx Buffer 1.		
		1(transmit)	The CAN Controller is transmitting a message from Tx Buffer 1.		
6	ES		Error Status. This bit is identical to the ES bit in the CANxGSR.	0	0
7	BS		Bus Status. This bit is identical to the BS bit in the CANxGSR.	0	0
8	RBS		Receive Buffer Status. This bit is identical to the RBS bit in the CANxGSR.	0	0
9	DOS		Data Overrun Status. This bit is identical to the DOS bit in the CANxGSR.	0	0

Table 445. CAN Status Register (CAN1SR - address 0x4004 401C, CAN2SR - address 0x4004 801C) bit description

Bit	Symbol	Value	Function	Reset Value	RM Set
10	TBS2 <sup>[1]</sup>		Transmit Buffer Status 2.	1	1
		0(lockeD)	Software cannot access the Tx Buffer 2 nor write to the corresponding CANxTFI, CANxTID, CANxTDA, and CANxTDB registers because a message is either waiting for transmission or is in transmitting process.		
		1(releaseD)	Software may write a message into the Transmit Buffer 2 and its CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		
11	TCS2 <sup>[2]</sup>		Transmission Complete Status.	1	x
		0(incomplete)	The previously requested transmission for Tx Buffer 2 is not complete.		
		1(complete)	The previously requested transmission for Tx Buffer 2 has been successfully completed.		
12	RS		Receive Status. This bit is identical to the RS bit in the GSR.	1	0
13	TS2		Transmit Status 2.	1	0
		0(idle)	There is no transmission from Tx Buffer 2.		
		1(transmit)	The CAN Controller is transmitting a message from Tx Buffer 2.		
14	ES		Error Status. This bit is identical to the ES bit in the CANxGSR.	0	0
15	BS		Bus Status. This bit is identical to the BS bit in the CANxGSR.	0	0
16	RBS		Receive Buffer Status. This bit is identical to the RBS bit in the CANxGSR.	0	0
17	DOS		Data Overrun Status. This bit is identical to the DOS bit in the CANxGSR.	0	0
18	TBS3 <sup>[1]</sup>		Transmit Buffer Status 3.	1	1
		0(lockeD)	Software cannot access the Tx Buffer 3 nor write to the corresponding CANxTFI, CANxTID, CANxTDA, and CANxTDB registers because a message is either waiting for transmission or is in transmitting process.		
		1(releaseD)	Software may write a message into the Transmit Buffer 3 and its CANxTFI, CANxTID, CANxTDA, and CANxTDB registers.		
19	TCS3 <sup>[2]</sup>		Transmission Complete Status.	1	x
		0(incomplete)	The previously requested transmission for Tx Buffer 3 is not complete.		
		1(complete)	The previously requested transmission for Tx Buffer 3 has been successfully completed.		
20	RS		Receive Status. This bit is identical to the RS bit in the GSR.	1	0
21	TS3		Transmit Status 3.	1	0
		0(idle)	There is no transmission from Tx Buffer 3.		
		1(transmit)	The CAN Controller is transmitting a message from Tx Buffer 3.		
22	ES		Error Status. This bit is identical to the ES bit in the CANxGSR.	0	0
23	BS		Bus Status. This bit is identical to the BS bit in the CANxGSR.	0	0
31:24	-		Reserved, the value read from a reserved bit is not defined.	NA	

- [1] If the CPU tries to write to this Transmit Buffer when the Transmit Buffer Status bit is '0' (locked), the written byte is not accepted and is lost without this being signalled.
- [2] The Transmission Complete Status bit is set '0' (incomplete) whenever the Transmission Request bit or the Self Reception Request bit is set '1' for this TX buffer. The Transmission Complete Status bit remains '0' until a message is transmitted successfully.

### 20.7.9 CAN Receive Frame Status register (CAN1RFS - 0x4004 4020, CAN2RFS - 0x4004 8020)

This register defines the characteristics of the current received message. It is read-only in normal operation but can be written for testing purposes if the RM bit in CANxMOD is 1.

**Table 446. CAN Receive Frame Status register (CAN1RFS - address 0x4004 4020, CAN2RFS - address 0x4004 8020) bit description**

Bit	Symbol	Function	Reset Value	RM Set
9:0	ID Index	If the BP bit (below) is 0, this value is the zero-based number of the Lookup Table RAM entry at which the Acceptance Filter matched the received Identifier. Disabled entries in the Standard tables are included in this numbering, but will not be matched. See <a href="#">Section 20.17 "Examples of acceptance filter tables and ID index values" on page 548</a> for examples of ID Index values.	0	X
10	BP	If this bit is 1, the current message was received in AF Bypass mode, and the ID Index field (above) is meaningless.	0	X
15:11	-	Reserved. The value read from a reserved bit is not defined.	NA	
19:16	DLC	The field contains the Data Length Code (DLC) field of the current received message. When RTR = 0, this is related to the number of data bytes available in the CANRDA and CANRDB registers as follows: 0000-0111 = 0 to 7 bytes 1000-1111 = 8 bytes With RTR = 1, this value indicates the number of data bytes requested to be sent back, with the same encoding.	0	X
29:20	-	Reserved. Read value is undefined, only zero should be written.	NA	
30	RTR	This bit contains the Remote Transmission Request bit of the current received message. 0 indicates a Data Frame, in which (if DLC is non-zero) data can be read from the CANRDA and possibly the CANRDB registers. 1 indicates a Remote frame, in which case the DLC value identifies the number of data bytes requested to be sent using the same Identifier.	0	X
31	FF	A 0 in this bit indicates that the current received message included an 11-bit Identifier, while a 1 indicates a 29-bit Identifier. This affects the contents of the CANid register described below.	0	X

#### 20.7.9.1 ID index field

The ID Index is a 10-bit field in the Info Register that contains the table position of the ID Look-up Table if the currently received message was accepted. The software can use this index to simplify message transfers from the Receive Buffer into the Shared Message Memory. Whenever bit 10 (BP) of the ID Index in the CANRFS register is 1, the current CAN message was received in acceptance filter bypass mode.

#### 20.7.10 CAN Receive Identifier register (CAN1RID - 0x4004 4024, CAN2RID - 0x4004 8024)

This register contains the Identifier field of the current received message. It is read-only in normal operation but can be written for testing purposes if the RM bit in CANmod is 1. It has two different formats depending on the FF bit in CANRFS. See [Table 434](#) for details on specific CAN channel register address.



**Table 447. CAN Receive Identifier register (CAN1RID - address 0x4004 4024, CAN2RID - address 0x4004 8024) bit description**

Bit	Symbol	Function	Reset Value	RM Set
10:0	ID	The 11-bit Identifier field of the current received message. In CAN 2.0A, these bits are called ID10-0, while in CAN 2.0B they're called ID29-18.	0	X
31:11	-	Reserved. The value read from a reserved bit is not defined.	NA	

**Table 448. RX Identifier register when FF = 1**

Bit	Symbol	Function	Reset Value	RM Set
28:0	ID	The 29-bit Identifier field of the current received message. In CAN 2.0B these bits are called ID29-0.	0	X
31:29	-	Reserved. The value read from a reserved bit is not defined.	NA	

### 20.7.11 CAN Receive Data register A (CAN1RDA - 0x4004 4028, CAN2RDA - 0x4004 8028)

This register contains the first 1-4 Data bytes of the current received message. It is read-only in normal operation, but can be written for testing purposes if the RM bit in CANMOD is 1. See [Table 434](#) for details on specific CAN channel register address.

**Table 449. CAN Receive Data register A (CAN1RDA - address 0x4004 4028, CAN2RDA - address 0x4004 8028) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	Data 1	If the DLC field in CANRFS $\geq$ 0001, this contains the first Data byte of the current received message.	0	X
15:8	Data 2	If the DLC field in CANRFS $\geq$ 0010, this contains the first Data byte of the current received message.	0	X
23:16	Data 3	If the DLC field in CANRFS $\geq$ 0011, this contains the first Data byte of the current received message.	0	X
31:24	Data 4	If the DLC field in CANRFS $\geq$ 0100, this contains the first Data byte of the current received message.	0	X

### 20.7.12 CAN Receive Data register B (CAN1RDB - 0x4004 402C, CAN2RDB - 0x4004 802C)

This register contains the 5th through 8th Data bytes of the current received message. It is read-only in normal operation, but can be written for testing purposes if the RM bit in CANMOD is 1. See [Table 434](#) for details on specific CAN channel register address.

**Table 450. CAN Receive Data register B (CAN1RDB - address 0x4004 402C, CAN2RDB - address 0x4004 802C) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	Data 5	If the DLC field in CANRFS $\geq$ 0101, this contains the first Data byte of the current received message.	0	X
15:8	Data 6	If the DLC field in CANRFS $\geq$ 0110, this contains the first Data byte of the current received message.	0	X
23:16	Data 7	If the DLC field in CANRFS $\geq$ 0111, this contains the first Data byte of the current received message.	0	X
31:24	Data 8	If the DLC field in CANRFS $\geq$ 1000, this contains the first Data byte of the current received message.	0	X

**20.7.13 CAN Transmit Frame Information register (CAN1TFI[1/2/3] - 0x4004 40[30/ 40/50], CAN2TFI[1/2/3] - 0x4004 80[30/40/50])**

When the corresponding TBS bit in CANSR is 1, software can write to one of these registers to define the format of the next transmit message for that Tx buffer. Bits not listed read as 0 and should be written as 0.

The values for the reserved bits of the CANxTFI register in the Transmit Buffer should be set to the values expected in the Receive Buffer for an easy comparison, when using the Self Reception facility (self test), otherwise they are not defined.

The CAN Controller consist of three Transmit Buffers. Each of them has a length of 4 words and is able to store one complete CAN message as shown in [Figure 78](#).

The buffer layout is subdivided into Descriptor and Data Field where the first word of the Descriptor Field includes the TX Frame Info that describes the Frame Format, the Data Length and whether it is a Remote or Data Frame. In addition, a TX Priority register allows the definition of a certain priority for each transmit message. Depending on the chosen Frame Format, an 11-bit identifier for Standard Frame Format (SFF) or an 29-bit identifier for Extended Frame Format (EFF) follows. Note that unused bits in the TID field have to be defined as 0. The Data Field in TDA and TDB contains up to eight data bytes.

**Table 451. CAN Transmit Frame Information register (CAN1TFI[1/2/3] - address 0x4004 40[30/40/50], CAN2TFI[1/2/3] - 0x4004 80[30/40/50]) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	PRIO	If the TPM (Transmit Priority Mode) bit in the CANxMOD register is set to 1, enabled Tx Buffers contend for the right to send their messages based on this field. The buffer with the lowest TX Priority value wins the prioritization and is sent first.		x
15:8	-	Reserved. Read value is undefined, only zero should be written.	0	
19:16	DLC	Data Length Code. This value is sent in the DLC field of the next transmit message. In addition, if RTR = 0, this value controls the number of Data bytes sent in the next transmit message, from the CANxTDA and CANxTDB registers: 0000-0111 = 0-7 bytes 1xxx = 8 bytes	0	X
29:20	-	Reserved. Read value is undefined, only zero should be written.	0	
30	RTR	This value is sent in the RTR bit of the next transmit message. If this bit is 0, the number of data bytes called out by the DLC field are sent from the CANxTDA and CANxTDB registers. If this bit is 1, a Remote Frame is sent, containing a request for that number of bytes.	0	X
31	FF	If this bit is 0, the next transmit message will be sent with an 11-bit Identifier (standard frame format), while if it's 1, the message will be sent with a 29-bit Identifier (extended frame format).	0	X

**Automatic transmit priority detection**

To allow uninterrupted streams of transmit messages, the CAN Controller provides Automatic Transmit Priority Detection for all Transmit Buffers. Depending on the selected Transmit Priority Mode, internal prioritization is based on the CAN Identifier or a user defined "local priority". If more than one message is enabled for transmission (TR=1) the internal transmit message queue is organized such as that the transmit buffer with the lowest CAN Identifier (TID) or the lowest "local priority" (TX Priority) wins the prioritization

and is sent first. The result of the internal scheduling process is taken into account short before a new CAN message is sent on the bus. This is also true after the occurrence of a transmission error and right before a re-transmission.

**Tx DLC**

The number of bytes in the Data Field of a message is coded with the Data Length Code (DLC). At the start of a Remote Frame transmission the DLC is not considered due to the RTR bit being '1' (remote). This forces the number of transmitted/received data bytes to be 0. Nevertheless, the DLC must be specified correctly to avoid bus errors, if two CAN Controllers start a Remote Frame transmission with the same identifier simultaneously. For reasons of compatibility no DLC > 8 should be used. If a value greater than 8 is selected, 8 bytes are transmitted in the data frame with the Data Length Code specified in DLC. The range of the Data Byte Count is 0 to 8 bytes and is coded as follows:

(12)

$$DataByteCount = DLC$$

**20.7.14 CAN Transmit Identifier register (CAN1TID[1/2/3] - 0x4004 40[34/44/54], CAN2TID[1/2/3] - 0x4004 80[34/44/54])**

When the corresponding TBS bit in CANxSR is 1, software can write to one of these registers to define the Identifier field of the next transmit message. Bits not listed read as 0 and should be written as 0. The register assumes two different formats depending on the FF bit in CANTFI.

In Standard Frame Format messages, the CAN Identifier consists of 11 bits (ID.28 to ID.18), and in Extended Frame Format messages, the CAN identifier consists of 29 bits (ID.28 to ID.0). ID.28 is the most significant bit, and it is transmitted first on the bus during the arbitration process. The Identifier acts as the message's name, used in a receiver for acceptance filtering, and also determines the bus access priority during the arbitration process.

**Table 452. CAN Transfer Identifier register (CAN1TID[1/2/3] - address 0x4004 40[34/44/54], CAN2TID[1/2/3] - address 0x4004 80[34/44/54]) bit description**

Bit	Symbol	Function	Reset Value	RM Set
10:0	ID	The 11-bit Identifier to be sent in the next transmit message.	0	X
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA	

**Table 453. Transfer Identifier register when FF = 1**

Bit	Symbol	Function	Reset Value	RM Set
28:0	ID	The 29-bit Identifier to be sent in the next transmit message.	0	X
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA	

**20.7.15 CAN Transmit Data register A (CAN1TDA[1/2/3] - 0x4004 40[38/48/58], CAN2TDA[1/2/3] - 0x4004 80[38/48/58])**

When the corresponding TBS bit in CANSR is 1, software can write to one of these registers to define the first 1 - 4 data bytes of the next transmit message. The Data Length Code defines the number of transferred data bytes. The first bit transmitted is the most significant bit of TX Data Byte 1.

**Table 454. CAN Transmit Data register A (CAN1TDA[1/2/3] - address 0x4004 40[38/48/58], CAN2TDA[1/2/3] - address 0x4004 80[38/48/58]) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	Data 1	If RTR = 0 and DLC ≥ 0001 in the corresponding CANxTFI, this byte is sent as the first Data byte of the next transmit message.	0	X
15:8	Data 2	If RTR = 0 and DLC ≥ 0010 in the corresponding CANxTFI, this byte is sent as the 2nd Data byte of the next transmit message.	0	X
23:16	Data 3	If RTR = 0 and DLC ≥ 0011 in the corresponding CANxTFI, this byte is sent as the 3rd Data byte of the next transmit message.	0	X
31:24	Data 4	If RTR = 0 and DLC ≥ 0100 in the corresponding CANxTFI, this byte is sent as the 4th Data byte of the next transmit message.	0	X

### 20.7.16 CAN Transmit Data register B (CAN1TDB[1/2/3] - 0x4004 40[3C/4C/5C], CAN2TDB[1/2/3] - 0x4004 80[3C/4C/5C])

When the corresponding TBS bit in CANSR is 1, software can write to one of these registers to define the 5th through 8th data bytes of the next transmit message. The Data Length Code defines the number of transferred data bytes. The first bit transmitted is the most significant bit of TX Data Byte 1.

**Table 455. CAN Transmit Data register B (CAN1TDB[1/2/3] - address 0x4004 40[3C/4C/5C], CAN2TDB[1/2/3] - address 0x4004 80[3C/4C/5C]) bit description**

Bit	Symbol	Function	Reset Value	RM Set
7:0	Data 5	If RTR = 0 and DLC ≥ 0101 in the corresponding CANTFI, this byte is sent as the 5th Data byte of the next transmit message.	0	X
15:8	Data 6	If RTR = 0 and DLC ≥ 0110 in the corresponding CANTFI, this byte is sent as the 6th Data byte of the next transmit message.	0	X
23:16	Data 7	If RTR = 0 and DLC ≥ 0111 in the corresponding CANTFI, this byte is sent as the 7th Data byte of the next transmit message.	0	X
31:24	Data 8	If RTR = 0 and DLC ≥ 1000 in the corresponding CANTFI, this byte is sent as the 8th Data byte of the next transmit message.	0	X

### 20.7.17 CAN Sleep Clear register (CANSLEEPCLR - 0x400F C110)

This register provides the current sleep state of the two CAN channels and provides a means to restore the clocks to that channel following wake-up. Refer to [Section 20.8.2 “Sleep mode”](#) for more information on the CAN sleep feature.

**Table 456. CAN Sleep Clear register (CANSLEEPCLR - address 0x400F C110) bit description**

Bit	Symbol	Function	Reset Value
0	-	Reserved. Read value is undefined, only zero should be written.	NA
1	CAN1SLEEP	Sleep status and control for CAN channel 1. Read: when 1, indicates that CAN channel 1 is in the sleep mode. Write: writing a 1 causes clocks to be restored to CAN channel 1.	0
2	CAN2SLEEP	Sleep status and control for CAN channel 2. Read: when 1, indicates that CAN channel 2 is in the sleep mode. Write: writing a 1 causes clocks to be restored to CAN channel 2.	0
31:3	-	Reserved. Read value is undefined, only zero should be written.	NA

### 20.7.18 CAN Wake-up Flags register (CANWAKEFLAGS - 0x400F C114)

This register provides the wake-up status for the two CAN channels and allows clearing wake-up events. Refer to [Section 20.8.2 “Sleep mode”](#) for more information on the CAN sleep feature.

**Table 457. CAN Wake-up Flags register (CANWAKEFLAGS - address 0x400F C114) bit description**

Bit	Symbol	Function	Reset Value
0	-	Reserved. Read value is undefined, only zero should be written.	NA
1	CAN1WAKE	Wake-up status for CAN channel 1. Read: when 1, indicates that a falling edge has occurred on the receive data line of CAN channel 1. Write: writing a 1 clears this bit.	0
2	CAN2WAKE	Wake-up status for CAN channel 2. Read: when 1, indicates that a falling edge has occurred on the receive data line of CAN channel 2. Write: writing a 1 clears this bit.	0
31:3	-	Reserved. Read value is undefined, only zero should be written.	NA

## 20.8 CAN controller operation

### 20.8.1 Error handling

The CAN Controllers count and handle transmit and receive errors as specified in CAN Spec 2.0B. The Transmit and Receive Error Counters are incremented for each detected error and are decremented when operation is error-free. If the Transmit Error counter contains 255 and another error occurs, the CAN Controller is forced into a state called Bus-Off. In this state, the following register bits are set: BS in CANxSR, BEI and EI in CANxIR if these are enabled, and RM in CANxMOD. RM resets and disables much of the CAN Controller. Also at this time the Transmit Error Counter is set to 127 and the Receive Error Counter is cleared. Software must next clear the RM bit. Thereafter the Transmit Error Counter will count down 128 occurrences of the Bus Free condition (11 consecutive recessive bits). Software can monitor this countdown by reading the Tx Error Counter. When this countdown is complete, the CAN Controller clears BS and ES in CANxSR, and sets EI in CANxSR if EIE in IER is 1.

The Tx and Rx error counters can be written if RM in CANxMOD is 1. Writing 255 to the Tx Error Counter forces the CAN Controller to Bus-Off state. If Bus-Off (BS in CANxSR) is 1, writing any value 0 through 254 to the Tx Error Counter clears Bus-Off. When software clears RM in CANxMOD thereafter, only one Bus Free condition (11 consecutive recessive bits) is needed before operation resumes.

### 20.8.2 Sleep mode

The CAN Controller will enter sleep mode if the SM bit in the CAN Mode register is 1, no CAN interrupt is pending, and there is no activity on the CAN bus. Software can only set SM when RM in the CAN Mode register is 0; it can also set the WUIE bit in the CAN Interrupt Enable register to enable an interrupt on any wake-up condition.

The CAN Controller wakes up (and sets WUI in the CAN Interrupt register if WUIE in the CAN Interrupt Enable register is 1) in response to a) a dominant bit on the CAN bus, or b) software clearing SM in the CAN Mode register. A sleeping CAN Controller that wakes up in response to bus activity is not able to receive an initial message until after it detects Bus\_Free (11 consecutive recessive bits). If an interrupt is pending or the CAN bus is active when software sets SM, the wake-up is immediate.

Upon wake-up, software needs to do the following things:

1. Write a 1 to the relevant bit(s) in the CANSLEEPCLR register.
2. Write a 0 to the SM bit in the CAN1MOD and/or CAN2MOD register.
3. Write a 1 to the relevant bit(s) in the CANWAKEFLAGS register. Failure to perform this step will prevent subsequent entry into Power-down mode.

If the LPC178x/177x is in Deep Sleep or Power-down mode, CAN activity will wake up the device if the CAN activity interrupt is enabled. See [Section 4.7 “Power control”](#).

### 20.8.3 Interrupts

Each CAN Controller produces 3 interrupt requests, Receive, Transmit, and “other status”. The Transmit interrupt is the OR of the Transmit interrupts from the three Tx Buffers. All of the interrupts share one NVIC channel. A separate interrupt is provided for the CAN activity interrupt.

### 20.8.4 Transmit priority

If the TPM bit in the CANxMOD register is 0, multiple enabled Tx Buffers contend for the right to send their messages based on the value of their CAN Identifier (TID). If TPM is 1, they contend based on the PRIO fields in bits 7:0 of their CANxTFS registers. In both cases the smallest binary value has priority. If two (or three) transmit-enabled buffers have the same smallest value, the lowest-numbered buffer sends first.

The CAN controller selects among multiple enabled Tx Buffers dynamically, just before it sends each message.

## 20.9 Centralized CAN registers

For easy and fast access, all CAN Controller Status bits from each CAN Controller Status register are bundled together. Each defined byte of the following registers contains one particular status bit from each of the CAN controllers, in its LS bits.

All Status registers are read-only and allow byte, half word and word access.

### 20.9.1 Central Transmit Status Register (CANTxSR - 0x4004 0000)

Table 458. Central Transit Status Register (CANTxSR - address 0x4004 0000) bit description

Bit	Symbol	Description	Reset Value
0	TS1	When 1, the CAN controller 1 is sending a message (same as TS in the CAN1GSR).	0
1	TS2	When 1, the CAN controller 2 is sending a message (same as TS in the CAN2GSR)	0
7:2	-	Reserved, the value read from a reserved bit is not defined.	NA

**Table 458. Central Transit Status Register (CANTxSR - address 0x4004 0000) bit description**

Bit	Symbol	Description	Reset Value
8	TBS1	When 1, all 3 Tx Buffers of the CAN1 controller are available to the CPU (same as TBS in CAN1GSR).	1
9	TBS2	When 1, all 3 Tx Buffers of the CAN2 controller are available to the CPU (same as TBS in CAN2GSR).	1
15:10	-	Reserved, the value read from a reserved bit is not defined.	NA
16	TCS1	When 1, all requested transmissions have been completed successfully by the CAN1 controller (same as TCS in CAN1GSR).	1
17:16	TCS2	When 1, all requested transmissions have been completed successfully by the CAN2 controller (same as TCS in CAN2GSR).	1
31:18	-	Reserved, the value read from a reserved bit is not defined.	NA

### 20.9.2 Central Receive Status Register (CANRxSR - 0x4004 0004)

**Table 459. Central Receive Status Register (CANRxSR - address 0x4004 0004) bit description**

Bit	Symbol	Description	Reset Value
0	RS1	When 1, CAN1 is receiving a message (same as RS in CAN1GSR).	0
1	RS2	When 1, CAN2 is receiving a message (same as RS in CAN2GSR).	0
7:2	-	Reserved, the value read from a reserved bit is not defined.	NA
8	RB1	When 1, a received message is available in the CAN1 controller (same as RBS in CAN1GSR).	0
9	RB2	When 1, a received message is available in the CAN2 controller (same as RBS in CAN2GSR).	0
15:10	-	Reserved, the value read from a reserved bit is not defined.	NA
16	DOS1	When 1, a message was lost because the preceding message to CAN1 controller was not read out quickly enough (same as DOS in CAN1GSR).	0
17:16	DOS2	When 1, a message was lost because the preceding message to CAN2 controller was not read out quickly enough (same as DOS in CAN2GSR).	0
31:18	-	Reserved, the value read from a reserved bit is not defined.	NA

### 20.9.3 Central Miscellaneous Status Register (CANMSR - 0x4004 0008)

**Table 460. Central Miscellaneous Status Register (CANMSR - address 0x4004 0008) bit description**

Bit	Symbol	Description	Reset Value
0	E1	When 1, one or both of the CAN1 Tx and Rx Error Counters has reached the limit set in the CAN1EWL register (same as ES in CAN1GSR)	0
1	E2	When 1, one or both of the CAN2 Tx and Rx Error Counters has reached the limit set in the CAN2EWL register (same as ES in CAN2GSR)	0
7:2	-	Reserved, the value read from a reserved bit is not defined.	NA
8	BS1	When 1, the CAN1 controller is currently involved in bus activities (same as BS in CAN1GSR).	0
9	BS2	When 1, the CAN2 controller is currently involved in bus activities (same as BS in CAN2GSR).	0
31:10	-	Reserved, the value read from a reserved bit is not defined.	NA

## 20.10 Global acceptance filter

This block provides lookup for received Identifiers (called Acceptance Filtering in CAN terminology) for all the CAN Controllers. It includes a 512 × 32 (2 kB) RAM in which software maintains one to five tables of Identifiers. This RAM can contain up to 1024 Standard Identifiers or 512 Extended Identifiers, or a mixture of both types.

## 20.11 Acceptance filter modes

The Acceptance Filter can be put into different modes by setting the according AccOff, AccBP, and eFCAN bits in the Acceptance Filter Mode Register ([Section 20.14.1 “Acceptance Filter Mode Register \(AFMR - 0x4003 C000\)”](#)). During each mode the access to the Configuration Register and the ID Look-up table is handled differently.

**Table 461. Acceptance filter modes and access control**

Acceptance filter mode	Bit AccOff	Bit AccBP	Acceptance filter state	ID Look-up table RAM <sup>[1]</sup>	Acceptance filter config. registers	CAN controller message receive interrupt
Off Mode	1	0	reset & halted	r/w access from CPU	r/w access from CPU	no messages accepted
Bypass Mode	X	1	reset & halted	r/w access from CPU	r/w access from CPU	all messages accepted
Operating Mode and FullCAN Mode	0	0	running	read-only from CPU <sup>[2]</sup>	access from Acceptance filter only	hardware acceptance filtering

[1] The whole ID Look-up Table RAM is only word accessible.

[2] During the Operating Mode of the Acceptance Filter the Look-up Table can be accessed only to disable or enable Messages.

A write access to all section configuration registers is only possible during the Acceptance Filter Off and Bypass Mode. Read access is allowed in all Acceptance Filter Modes.

### 20.11.1 Acceptance filter Off mode

The Acceptance Filter Off Mode is typically used during initialization. During this mode an unconditional access to all registers and to the Look-up Table RAM is possible. With the Acceptance Filter Off Mode, CAN messages are not accepted and therefore not stored in the Receive Buffers of active CAN Controllers.

### 20.11.2 Acceptance filter Bypass mode

The Acceptance Filter Bypass Mode can be used for example to change the acceptance filter configuration during a running system, e.g. change of identifiers in the ID-Look-up Table memory. During this re-configuration, software acceptance filtering has to be used.

It is recommended to use the ID ready Interrupt (ID Index) and the Receive Interrupt (RI). In this mode all CAN message are accepted and stored in the Receive Buffers of active CAN Controllers.



### 20.11.3 Acceptance filter Operating mode

The Acceptance Filter is in Operating Mode when neither the AccOff nor the AccBP in the Configuration Register is set and the eFCAN = 0.

### 20.11.4 FullCAN mode

The Acceptance Filter is in Operating Mode when neither the AccOff nor the AccBP in the Configuration Register is set and the eFCAN = 1. More details on FullCAN mode are available in [Section 20.16 “FullCAN mode”](#).

## 20.12 Sections of the ID look-up table RAM

Four 12-bit section configuration registers (SFF\_sa, SFF\_GRP\_sa, EFF\_sa, EFF\_GRP\_sa) are used to define the boundaries of the different identifier sections in the ID-Look-up Table Memory. The fifth 12-bit section configuration register, the End of Table address register (ENDofTable) is used to define the end of all identifier sections. The End of Table address is also used to assign the start address of the section where FullCAN Message Objects, if enabled are stored.

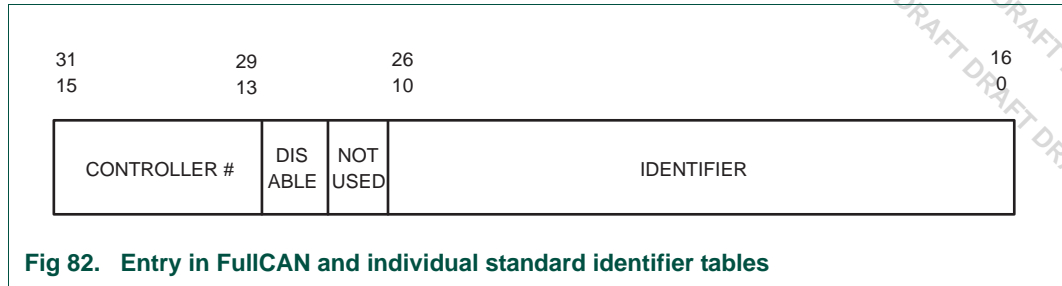
**Table 462. Section configuration register settings**

ID-Look up Table Section	Register	Value	Section status
FullCAN (Standard Frame Format) Identifier Section	SFF_sa	= 0x000	disabled
		> 0x000	enabled
Explicit Standard Frame Format Identifier Section	SFF_GRP_sa	= SFF_sa	disabled
		> SFF_sa	enabled
Group of Standard Frame Format Identifier Section	EFF_sa	= SFF_GRP_sa	disabled
		> SFF_GRP_sa	enabled
Explicit Extended Frame Format Identifier Section	EFF_GRP_sa	= EFF_sa	disabled
		> EFF_sa	enabled
Group of Extended Frame Format Identifier Section	ENDofTable	= EFF_GRP_sa	disabled
		> EFF_GRP_sa	enabled

## 20.13 ID look-up table RAM

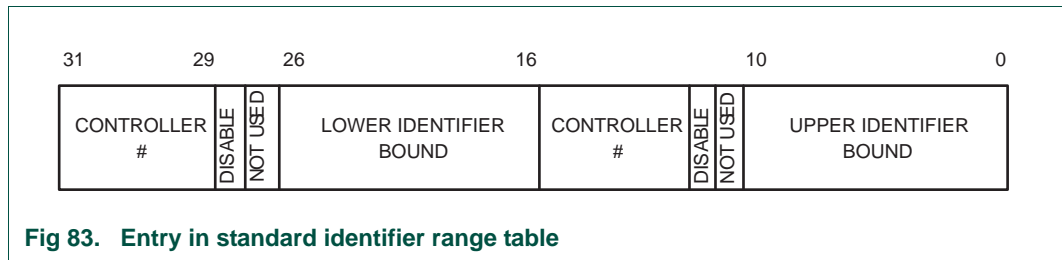
The Whole ID Look-up Table RAM is only word accessible. A write access is only possible during the Acceptance Filter Off or Bypass Mode. Read access is allowed in all Acceptance Filter Modes.

If Standard (11-bit) Identifiers are used in the application, at least one of 3 tables in Acceptance Filter RAM must not be empty. If the optional “FullCAN mode” is enabled, the first table contains Standard identifiers for which reception is to be handled in this mode. The next table contains individual Standard Identifiers and the third contains ranges of Standard Identifiers, for which messages are to be received via the CAN Controllers. The tables of FullCAN and individual Standard Identifiers must be arranged in ascending numerical order, one per halfword, two per word. Since each CAN bus has its own address map, each entry also contains the number of the CAN Controller (001-010) to which it applies.



**Fig 82. Entry in FullCAN and individual standard identifier tables**

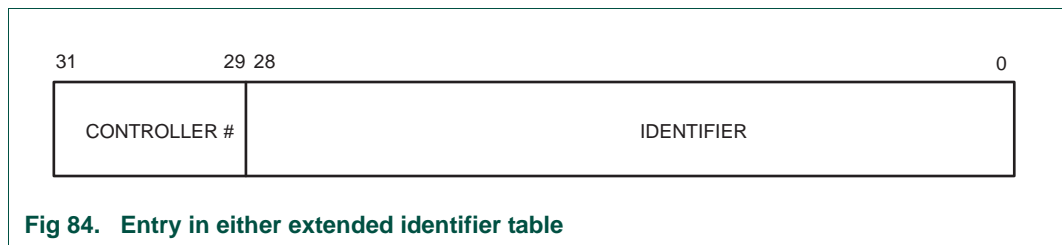
The table of Standard Identifier Ranges contains paired upper and lower (inclusive) bounds, one pair per word. These must also be arranged in ascending numerical order.



**Fig 83. Entry in standard identifier range table**

The disable bits in Standard entries provide a means to turn response, to particular CAN Identifiers or ranges of Identifiers, on and off dynamically. When the Acceptance Filter function is enabled, only the disable bits in Acceptance Filter RAM can be changed by software. Response to a range of Standard addresses can be enabled by writing 32 zero bits to its word in RAM, and turned off by writing 32 one bits (0xFFFF FFFF) to its word in RAM. Only the disable bits are actually changed. Disabled entries must maintain the ascending sequence of Identifiers.

If Extended (29-bit) Identifiers are used in the application, at least one of the other two tables in Acceptance Filter RAM must not be empty, one for individual Extended Identifiers and one for ranges of Extended Identifiers. The table of individual Extended Identifiers must be arranged in ascending numerical order.



**Fig 84. Entry in either extended identifier table**

The table of ranges of Extended Identifiers must contain an even number of entries, of the same form as in the individual Extended Identifier table. Like the Individual Extended table, the Extended Range must be arranged in ascending numerical order. The first and second (3rd and 4th ...) entries in the table are implicitly paired as an inclusive range of Extended addresses, such that any received address that falls in the inclusive range is received (accepted). Software must maintain the table to consist of such word pairs.

There is no facility to receive messages to Extended identifiers using the FullCAN method.

Five address registers point to the boundaries between the tables in Acceptance Filter RAM: FullCAN Standard addresses, Standard Individual addresses, Standard address ranges, Extended Individual addresses, and Extended address ranges. These tables must be consecutive in memory. The start of each of the latter four tables is implicitly the end of the preceding table. The end of the Extended range table is given in an End of Tables register. If the start address of a table equals the start of the next table or the End Of Tables register, that table is empty.

When the Receive side of a CAN controller has received a complete Identifier, it signals the Acceptance Filter of this fact. The Acceptance Filter responds to this signal, and reads the Controller number, the size of the Identifier, and the Identifier itself from the Controller. It then proceeds to search its RAM to determine whether the message should be received or ignored.

If FullCAN mode is enabled and the CAN controller signals that the current message contains a Standard identifier, the Acceptance Filter first searches the table of identifiers for which reception is to be done in FullCAN mode. Otherwise, or if the AF doesn't find a match in the FullCAN table, it searches its individual Identifier table for the size of Identifier signalled by the CAN controller. If it finds an equal match, the AF signals the CAN controller to retain the message, and provides it with an ID Index value to store in its Receive Frame Status register.

If the Acceptance Filter does not find a match in the appropriate individual Identifier table, it then searches the Identifier Range table for the size of Identifier signalled by the CAN controller. If the AF finds a match to a range in the table, it similarly signals the CAN controller to retain the message, and provides it with an ID Index value to store in its Receive Frame Status register. If the Acceptance Filter does not find a match in either the individual or Range table for the size of Identifier received, it signals the CAN controller to discard/ignore the received message.

## 20.14 Acceptance filter registers

### 20.14.1 Acceptance Filter Mode Register (AFMR - 0x4003 C000)

The AccBP and AccOff bits of the acceptance filter mode register are used for putting the acceptance filter into the Bypass and Off mode. The eFCAN bit of the mode register can be used to activate a FullCAN mode enhancement for received 11-bit CAN ID messages.

**Table 463. Acceptance Filter Mode Register (AFMR - address 0x4003 C000) bit description**

Bit	Symbol	Value	Description	Reset Value
0	AccOff <sup>[2]</sup>	1	if AccBP is 0, the Acceptance Filter is not operational. All Rx messages on all CAN buses are ignored.	1
1	AccBP <sup>[1]</sup>	1	All Rx messages are accepted on enabled CAN controllers. Software must set this bit before modifying the contents of any of the registers described below, and before modifying the contents of Lookup Table RAM in any way other than setting or clearing Disable bits in Standard Identifier entries. When both this bit and AccOff are 0, the Acceptance filter operates to screen received CAN Identifiers.	0

**Table 463. Acceptance Filter Mode Register (AFMR - address 0x4003 C000) bit description**

Bit	Symbol	Value	Description	Reset Value
2	eFCAN <sup>[3]</sup>	0	Software must read all messages for all enabled IDs on all enabled CAN buses, from the receiving CAN controllers.	0
		1	The Acceptance Filter itself will take care of receiving and storing messages for selected Standard ID values on selected CAN buses. See <a href="#">Section 20.16 "FullCAN mode" on page 537</a> .	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

- [1] Acceptance Filter Bypass Mode (AccBP): By setting the AccBP bit in the Acceptance Filter Mode Register, the Acceptance filter is put into the Acceptance Filter Bypass mode. During bypass mode, the internal state machine of the Acceptance Filter is reset and halted. All received CAN messages are accepted, and acceptance filtering can be done by software.
- [2] Acceptance Filter Off mode (AccOff): After power-up or hardware reset, the Acceptance filter will be in Off mode, the AccOff bit in the Acceptance filter Mode register 0 will be set to 1. The internal state machine of the acceptance filter is reset and halted. If not in Off mode, setting the AccOff bit, either by hardware or by software, will force the acceptance filter into Off mode.
- [3] FullCAN Mode Enhancements: A FullCAN mode for received CAN messages can be enabled by setting the eFCAN bit in the acceptance filter mode register.

### 20.14.2 Section configuration registers

The 10-bit section configuration registers are used for the ID look-up table RAM to indicate the boundaries of the different sections for explicit and group of CAN identifiers for 11-bit CAN and 29-bit CAN identifiers, respectively. The 10-bit wide section configuration registers allow the use of a 512x32 (2 kB) look-up table RAM. The whole ID Look-up Table RAM is only word accessible. All five section configuration registers contain APB addresses for the acceptance filter RAM and do not include the APB base address. A write access to all section configuration registers is only possible during the Acceptance filter off and Bypass modes. Read access is allowed in all acceptance filter modes.

### 20.14.3 Standard Frame Individual Start Address register (SFF\_sa - 0x4003 C004)

**Table 464. Standard Frame Individual Start Address register (SFF\_sa - address 0x4003 C004) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
10:2	SFF_sa <sup>[1]</sup>	The start address of the table of individual Standard Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the SFF_GRP_sa register described below. For compatibility with possible future devices, write zeroes in bits 31:11 and 1:0 of this register. If the eFCAN bit in the AFMR is 1, this value also indicates the size of the table of Standard IDs which the Acceptance Filter will search and (if found) automatically store received messages in Acceptance Filter RAM.	0
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA

- [1] Write access to the look-up table section configuration registers are possible only during the Acceptance filter bypass mode or the Acceptance filter off mode.

### 20.14.4 Standard Frame Group Start Address register (SFF\_GRP\_sa - 0x4003 C008)

**Table 465. Standard Frame Group Start Address register (SFF\_GRP\_sa - address 0x4003 C008) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
11:2	SFF_GRP_sa <sup>[1]</sup>	The start address of the table of grouped Standard Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the EFF_sa register described below. The largest value that should be written to this register is 0x800, when only the Standard Individual table is used, and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register.	0
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Write access to the look-up table section configuration registers are possible only during the Acceptance filter bypass mode or the Acceptance filter off mode.

### 20.14.5 Extended Frame Start Address register (EFF\_sa - 0x4003 C00C)

**Table 466. Extended Frame Start Address register (EFF\_sa - address 0x4003 C00C) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
10:2	EFF_sa <sup>[1]</sup>	The start address of the table of individual Extended Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the EFF_GRP_sa register described below. The largest value that should be written to this register is 0x800, when both Extended Tables are empty and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:11 and 1:0 of this register.	0
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Write access to the look-up table section configuration registers are possible only during the Acceptance filter bypass mode or the Acceptance filter off mode.

### 20.14.6 Extended Frame Group Start Address register (EFF\_GRP\_sa - 0x4003 C010)

**Table 467. Extended Frame Group Start Address register (EFF\_GRP\_sa - address 0x4003 C010) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
11:2	Eff_GRP_sa <sup>[1]</sup>	The start address of the table of grouped Extended Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the ENDOfTable register described below. The largest value that should be written to this register is 0x800, when this table is empty and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register.	0
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Write access to the look-up table section configuration registers are possible only during the Acceptance filter bypass mode or the Acceptance filter off mode.

### 20.14.7 End of AF Tables register (ENDofTable - 0x4003 C014)

Table 468. End of AF Tables register (ENDofTable - address 0x4003 C014) bit description

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
11:2	EndofTable <sup>[1]</sup>	<p>The address above the last active address in the last active AF table. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register.</p> <p>If the eFCAN bit in the AFMR is 0, the largest value that should be written to this register is 0x800, which allows the last word (address 0x7FC) in AF Lookup Table RAM to be used.</p> <p>If the eFCAN bit in the AFMR is 1, this value marks the start of the area of Acceptance Filter RAM, into which the Acceptance Filter will automatically receive messages for selected IDs on selected CAN buses. In this case, the maximum value that should be written to this register is 0x800 minus 6 times the value in SFF_sa. This allows 12 bytes of message storage between this address and the end of Acceptance Filter RAM, for each Standard ID that is specified between the start of Acceptance Filter RAM, and the next active AF table.</p>	0
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] Write access to the look-up table section configuration registers are possible only during the Acceptance filter bypass mode or the Acceptance filter off mode.

### 20.14.8 Status registers

The look-up table error status registers, the error addresses, and the flag register provide information if a programming error in the look-up table RAM during the ID screening was encountered. The look-up table error address and flag register have only read access. If an error is detected, the LUTerror flag is set, and the LUTerrorAddr register provides the information under which address during an ID screening an error in the look-up table was encountered. Any read of the LUTerrorAddr Filter block can be used for a look-up table interrupt.

### 20.14.9 LUT Error Address register (LUTerrAd - 0x4003 C018)

Table 469. LUT Error Address register (LUTerrAd - address 0x4003 C018) bit description

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
10:2	LUTerrAd	If the LUT Error bit (below) is 1, this read-only field contains the address in AF Lookup Table RAM, at which the Acceptance Filter encountered an error in the content of the tables.	0
31:11	-	Reserved. Read value is undefined, only zero should be written.	NA

### 20.14.10 LUT Error register (LUTerr - 0x4003 C01C)

Table 470. LUT Error register (LUTerr - address 0x4003 C01C) bit description

Bit	Symbol	Description	Reset Value
0	LUTerr	This read-only bit is set to 1 if the Acceptance Filter encounters an error in the content of the tables in AF RAM. It is cleared when software reads the LUTerrAd register. This condition is ORed with the “other CAN” interrupts from the CAN controllers, to produce the request that is connected to the NVIC.	0
31:1	-	Reserved, the value read from a reserved bit is not defined.	NA

### 20.14.11 Global FullCANInterrupt Enable register (FCANIE - 0x4003 C020)

A write access to the Global FullCAN Interrupt Enable register is only possible when the Acceptance Filter is in the off mode.

**Table 471. Global FullCAN Enable register (FCANIE - address 0x4003 C020) bit description**

Bit	Symbol	Description	Reset Value
0	FCANIE	Global FullCAN Interrupt Enable. When 1, this interrupt is enabled.	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	NA

### 20.14.12 FullCAN Interrupt and Capture registers (FCANIC0 - 0x4003 C024 and FCANIC1 - 0x4003 C028)

For detailed description on these two registers, see [Section 20.16.2 "FullCAN interrupts"](#).

**Table 472. FullCAN Interrupt and Capture register 0 (FCANIC0 - address 0x4003 C024) bit description**

Bit	Symbol	Description	Reset Value
0	IntPnd0	FullCan Interrupt Pending bit 0.	0
...	IntPndx (0<x<31)	FullCan Interrupt Pending bit x.	0
31	IntPnd31	FullCan Interrupt Pending bit 31.	0

**Table 473. FullCAN Interrupt and Capture register 1 (FCANIC1 - address 0x4003 C028) bit description**

Bit	Symbol	Description	Reset Value
0	IntPnd32	FullCan Interrupt Pending bit 32.	0
...	IntPndx (32<x<63)	FullCan Interrupt Pending bit x.	0
31	IntPnd63	FullCan Interrupt Pending bit 63.	0

## 20.15 Configuration and search algorithm

The CAN Identifier Look-up Table Memory can contain explicit identifiers and groups of CAN identifiers for Standard and Extended CAN Frame Formats. They are organized as a sorted list or table with an increasing order of the Source CAN Channel (SCC) together with CAN Identifier in each section.

SCC value equals CAN\_controller - 1, i.e., SCC = 0 matches CAN1 and SCC = 1 matches CAN2.

Every CAN identifier is linked to an ID Index number. In case of a CAN Identifier match, the matching ID Index is stored in the Identifier Index of the Frame Status Register (CANRFS) of the according CAN Controller.

### 20.15.1 Acceptance filter search algorithm

The identifier screening process of the acceptance filter starts in the following order:

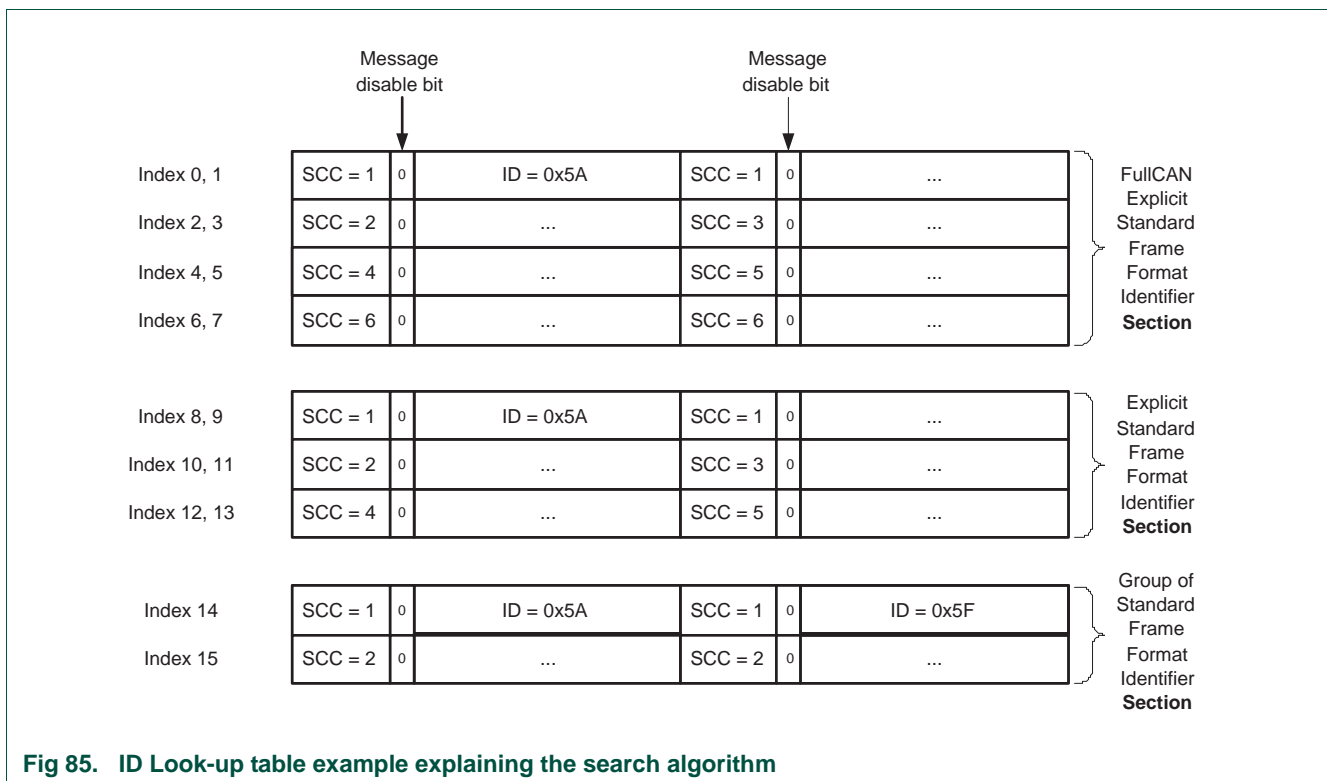
1. FullCAN (Standard Frame Format) Identifier Section
2. Explicit Standard Frame Format Identifier Section
3. Group of Standard Frame Format Identifier Section
4. Explicit Extended Frame Format Identifier Section
5. Group of Extended Frame Format Identifier Section

Note: Only activated sections will take part in the screening process.

In cases where equal message identifiers of same frame format are defined in more than one section, the first match will end the screening process for this identifier.

For example, if the same Source CAN Channel in conjunction with the identifier is defined in the FullCAN, the Explicit Standard Frame Format and the Group of Standard Frame Format Identifier Sections, the screening will already be finished with the match in the FullCAN section.

In the example of [Figure 85](#), Identifiers with their Source CAN Channel have been defined in the FullCAN, Explicit and Group of Standard Frame Format Identifier Sections. This example corresponds part with 6 CAN controllers.



The identifier 0x5A of the CAN Controller 1 with the Source CAN Channel SCC = 1, is defined in all three sections. With this configuration incoming CAN messages on CAN Controller 1 with a 0x5A identifier will find a match in the FullCAN section.

It is possible to disable the '0x5A identifier' in the FullCAN section. With that, the screening process would be finished with the match in the Explicit Identifier Section.

The first group in the Group Identifier Section has been defined in that way, that incoming CAN messages with identifiers of 0x5A up to 0x5F are accepted on CAN Controller 1 with the Source CAN Channel SCC = 1. As stated above, the identifier 0x5A would find a match already in the FullCAN or in the Explicit Identifier section if enabled. The rest of the defined identifiers of this group (0x5B to 0x5F) will find a match in this Group Identifier Section.



This way the user can switch dynamically between different filter modes for same identifiers.

## 20.16 FullCAN mode

The FullCAN mode is based on capabilities provided by the CAN Gateway module used in the LPC2xxx family of products. This block uses the Acceptance Filter to provide filtering for both CAN channels.

The concept of the CAN Gateway block is mainly based on a BasicCAN functionality. This concept fits perfectly in systems where a gateway is used to transfer messages or message data between different CAN channels. A BasicCAN device is generating a receive interrupt whenever a CAN message is accepted and received. Software has to move the received message out of the receive buffer from the according CAN controller into the user RAM.

To cover dashboard like applications where the controller typically receives data from several CAN channels for further processing, the CAN Gateway block was extended by a so-called FullCAN receive function. This additional feature uses an internal message handler to move received FullCAN messages from the receive buffer of the according CAN controller into the FullCAN message object data space of Look-up Table RAM.

When FullCAN mode is enabled, the Acceptance Filter itself takes care of receiving and storing messages for selected Standard ID values on selected CAN buses, in the style of "FullCAN" controllers.

In order to set this bit and use this mode, two other conditions must be met with respect to the contents of Acceptance Filter RAM and the pointers into it:

- The Standard Frame Individual Start Address Register (SFF\_sa) must be greater than or equal to the number of IDs for which automatic receive storage is to be done, times two. SFF\_sa must be rounded up to a multiple of 4 if necessary.
- The EndOfTable register must be less than or equal to 0x800 minus 6 times the SFF\_sa value, to allow 12 bytes of message storage for each ID for which automatic receive storage will be done.

When these conditions are met and eFCAN is set:

- The area between the start of Acceptance Filter RAM and the SFF\_sa address, is used for a table of individual Standard IDs and CAN Controller/bus identification, sorted in ascending order and in the same format as in the Individual Standard ID table (see [Figure 82 "Entry in FullCAN and individual standard identifier tables" on page 530](#)). Entries can be marked as "disabled" as in the other Standard tables. If there are an odd number of "FullCAN" ID's, at least one entry in this table must be so marked.
- The first (SFF\_sa)/2 IDindex values are assigned to these automatically-stored ID's. That is, IDindex values stored in the Rx Frame Status Register, for IDs not handled in this way, are increased by (SFF\_sa)/2 compared to the values they would have when eFCAN is 0.
- When a Standard ID is received, the Acceptance Filter searches this table before the Standard Individual and Group tables.

- When a message is received for a controller and ID in this table, the Acceptance filter reads the received message out of the CAN controller and stores it in Acceptance Filter RAM, starting at (EndOfTable) + its IDindex\*12.
- The format of such messages is shown in [Table 474](#).

### 20.16.1 FullCAN message layout

**Table 474. Format of automatically stored Rx messages**

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	F	R	0000				SEM	0000				DLC					00000					ID.28 ...			ID.18							
	F	T					[1:0]																									
+4	Rx Data 4				Rx Data 3				Rx Data 2				Rx Data 1																			
+8	Rx Data 8				Rx Data 7				Rx Data 6				Rx Data 5																			

The FF, RTR, and DLC fields are as described in [Table 446](#).

Since the FullCAN message object section of the Look-up table RAM can be accessed both by the Acceptance Filter and the CPU, there is a method for insuring that no CPU reads from FullCAN message object occurs while the Acceptance Filter hardware is writing to that object.

For this purpose the Acceptance Filter uses a 3-state semaphore, encoded with the two semaphore bits SEM1 and SEM0 (see [Table 474 "Format of automatically stored Rx messages"](#)) for each message object. This mechanism provides the CPU with information about the current state of the Acceptance Filter activity in the FullCAN message object section.

The semaphore operates in the following manner:

**Table 475. FullCAN semaphore operation**

SEM1	SEM0	activity
0	1	Acceptance Filter is updating the content
1	1	Acceptance Filter has finished updating the content
0	0	CPU is in process of reading from the Acceptance Filter

Prior to writing the first data byte into a message object, the Acceptance Filter will write the FrameInfo byte into the according buffer location with SEM[1:0] = 01.

After having written the last data byte into the message object, the Acceptance Filter will update the semaphore bits by setting SEM[1:0] = 11.

Before reading a message object, the CPU should read SEM[1:0] to determine the current state of the Acceptance Filter activity therein. If SEM[1:0] = 01, then the Acceptance Filter is currently active in this message object. If SEM[1:0] = 11, then the message object is available to be read.

Before the CPU begins reading from the message object, it should clear SEM[1:0] = 00.

When the CPU is finished reading, it can check SEM[1:0] again. At the time of this final check, if SEM[1:0] = 01 or 11, then the Acceptance Filter has updated the message object during the time when the CPU reads were taking place, and the CPU should discard the data. If, on the other hand, SEM[1:0] = 00 as expected, then valid data has been successfully read by the CPU.

[Figure 86](#) shows how software should use the SEM field to ensure that all three words read from the message are all from the same received message.

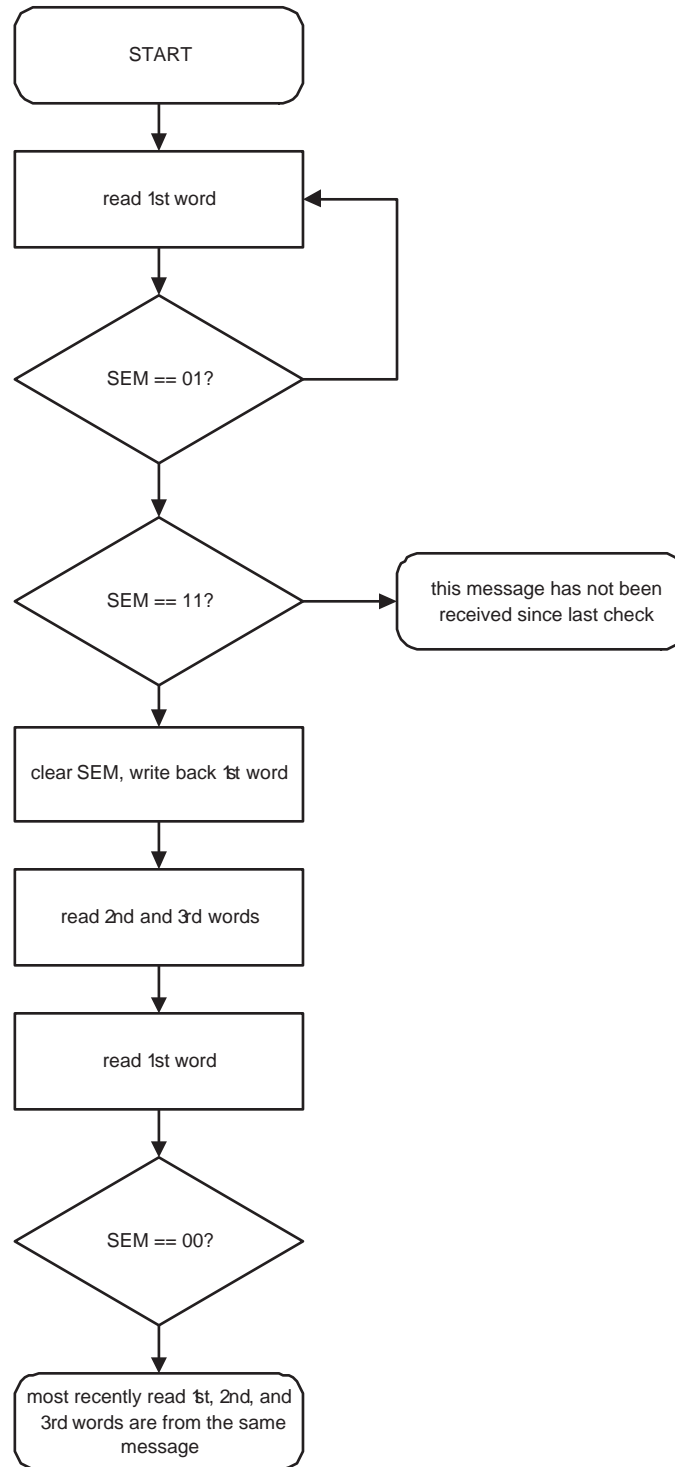


Fig 86. Semaphore procedure for reading an auto-stored message

## 20.16.2 FullCAN interrupts

The CAN Gateway Block contains a 2 kB ID Look-up Table RAM. With this size a maximum number of 146 FullCAN objects can be defined if the whole Look-up Table RAM is used for FullCAN objects only. Only the first 64 FullCAN objects can be configured to participate in the interrupt scheme. It is still possible to define more than 64 FullCAN objects. The only difference is, that the remaining FullCAN objects will not provide a FullCAN interrupt.

The FullCAN Interrupt Register-set contains interrupt flags (IntPndx) for (pending) FullCAN receive interrupts. As soon as a FullCAN message is received, the according interrupt bit (IntPndx) in the FCAN Interrupt Register gets asserted. In case that the Global FullCAN Interrupt Enable bit is set, the FullCAN Receive Interrupt is passed to the Vectored Interrupt Controller.

Application Software has to solve the following:

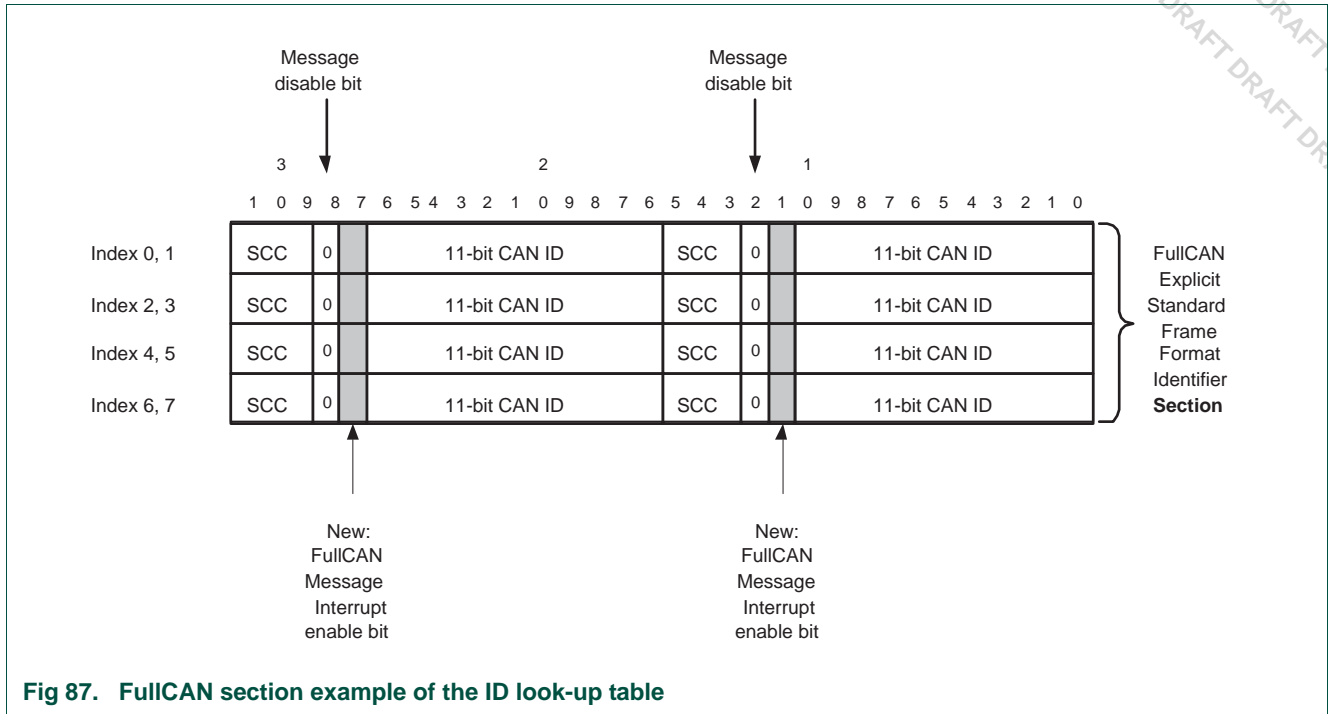
1. Index/Object number calculation based on the bit position in the FCANIC Interrupt Register for more than one pending interrupt.
2. Interrupt priority handling if more than one FullCAN receive interrupt is pending.

The software that covers the interrupt priority handling has to assign a receive interrupt priority to every FullCAN object. If more than one interrupt is pending, then the software has to decide, which received FullCAN object has to be served next.

To each FullCAN object a new FullCAN Interrupt Enable bit (FCANIntxEn) is added, so that it is possible to enable or disable FullCAN interrupts for each object individually. The new Message Lost flag (MsgLstx) is introduced to indicate whether more than one FullCAN message has been received since last time this message object was read by the CPU. The Interrupt Enable and the Message Lost bits reside in the existing Look-up Table RAM.

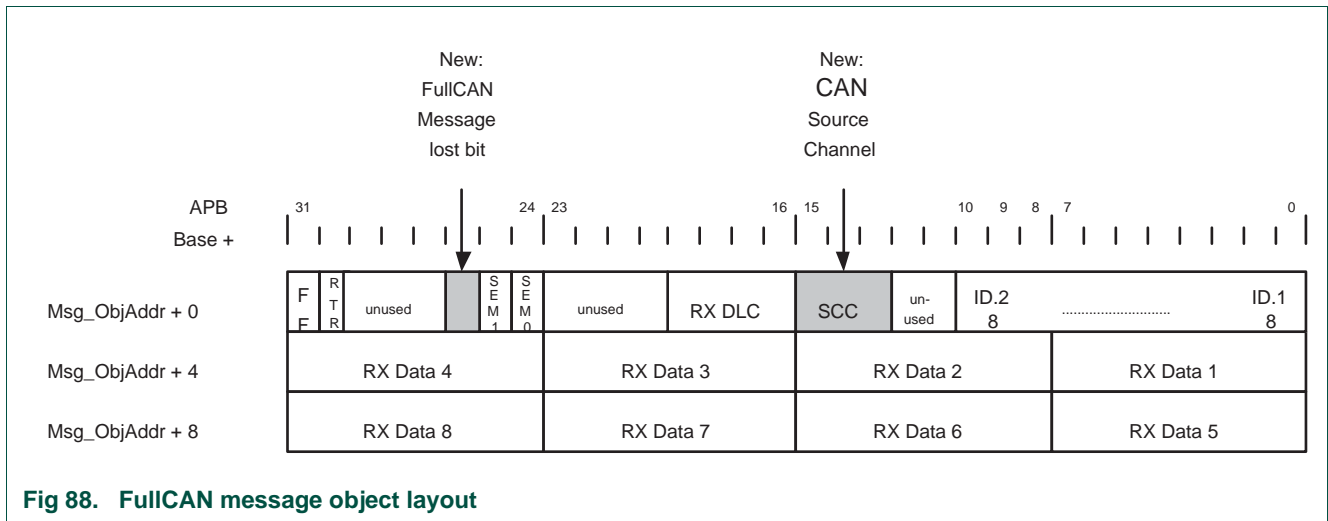
### 20.16.2.1 FullCAN message interrupt enable bit

In [Figure 87](#) 8 FullCAN Identifiers with their Source CAN Channel are defined in the FullCAN, Section. The new introduced FullCAN Message Interrupt enable bit can be used to enable for each FullCAN message an Interrupt.



**20.16.2.2 Message lost bit and CAN channel number**

Figure 88 is the detailed layout structure of one FullCAN message stored in the FullCAN message object section of the Look-up Table.



The new message lost bit (MsgLst) is introduced to indicate whether more than one FullCAN message has been received since last time this message object was read. For more information the CAN Source Channel (SCC) of the received FullCAN message is added to Message Object.

### 20.16.2.3 Setting the interrupt pending bits (IntPnd 63 to 0)

The interrupt pending bit (IntPndx) gets asserted in case of an accepted FullCAN message and if the interrupt of the according FullCAN Object is enabled (enable bit FCANIntxEn) is set).

During the **last write access** from the data storage of a FullCAN message object the interrupt pending bit of a FullCAN object (IntPndx) gets asserted.

### 20.16.2.4 Clearing the interrupt pending bits (IntPnd 63 to 0)

Each of the FullCAN Interrupt Pending requests gets cleared when the semaphore bits of a message object are cleared by Software (ARM CPU).

### 20.16.2.5 Setting the message lost bit of a FullCAN message object (MsgLost 63 to 0)

The Message Lost bit of a FullCAN message object gets asserted in case of an accepted FullCAN message and when the FullCAN Interrupt of the same object is asserted already.

During the **first write access** from the data storage of a FullCAN message object the Message Lost bit of a FullCAN object (MsgLostx) gets asserted if the interrupt pending bit is set already.

### 20.16.2.6 Clearing the message lost bit of a FullCAN message object (MsgLost 63 to 0)

The Message Lost bit of a FullCAN message object gets cleared when the FullCAN Interrupt of the same object is not asserted.

During the **first write access** from the data storage of a FullCAN message object the Message Lost bit of a FullCAN object (MsgLostx) gets cleared if the interrupt pending bit is not set.

## 20.16.3 Set and clear mechanism of the FullCAN interrupt

Special precaution is needed for the built-in set and clear mechanism of the FullCAN Interrupts. The following text illustrates how the already existing Semaphore Bits (see [Section 20.16.1 "FullCAN message layout"](#) for more details) and how the new introduced features (IntPndx, MsgLstx) will behave.

### 20.16.3.1 Scenario 1: Normal case, no message lost

[Figure 89](#) below shows a typical "normal" scenario in which an accepted FullCAN message is stored in the FullCAN Message Object Section. After storage the message is read out by Software (ARM CPU).

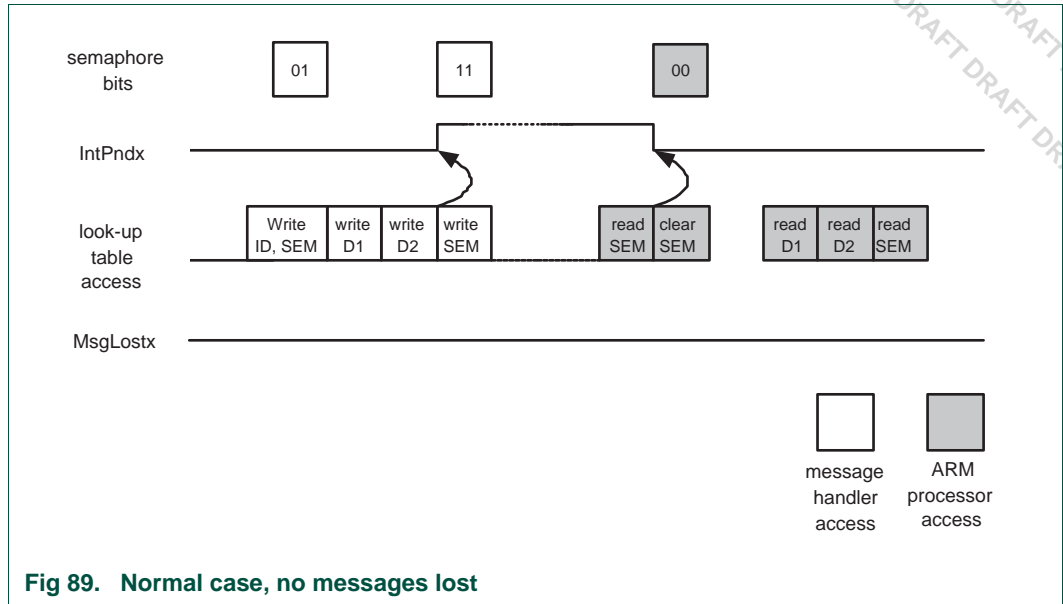


Fig 89. Normal case, no messages lost

20.16.3.2 Scenario 2: Message lost

In this scenario a first FullCAN Message is stored and read out by Software (1<sup>st</sup> Object write and read). In a second course a second message is stored (2<sup>nd</sup> Object write) but not read out before a third message gets stored (3<sup>rd</sup> Object write). Since the FullCAN Interrupt of that Object (IntPndx) is already asserted, the Message Lost Signal gets asserted.

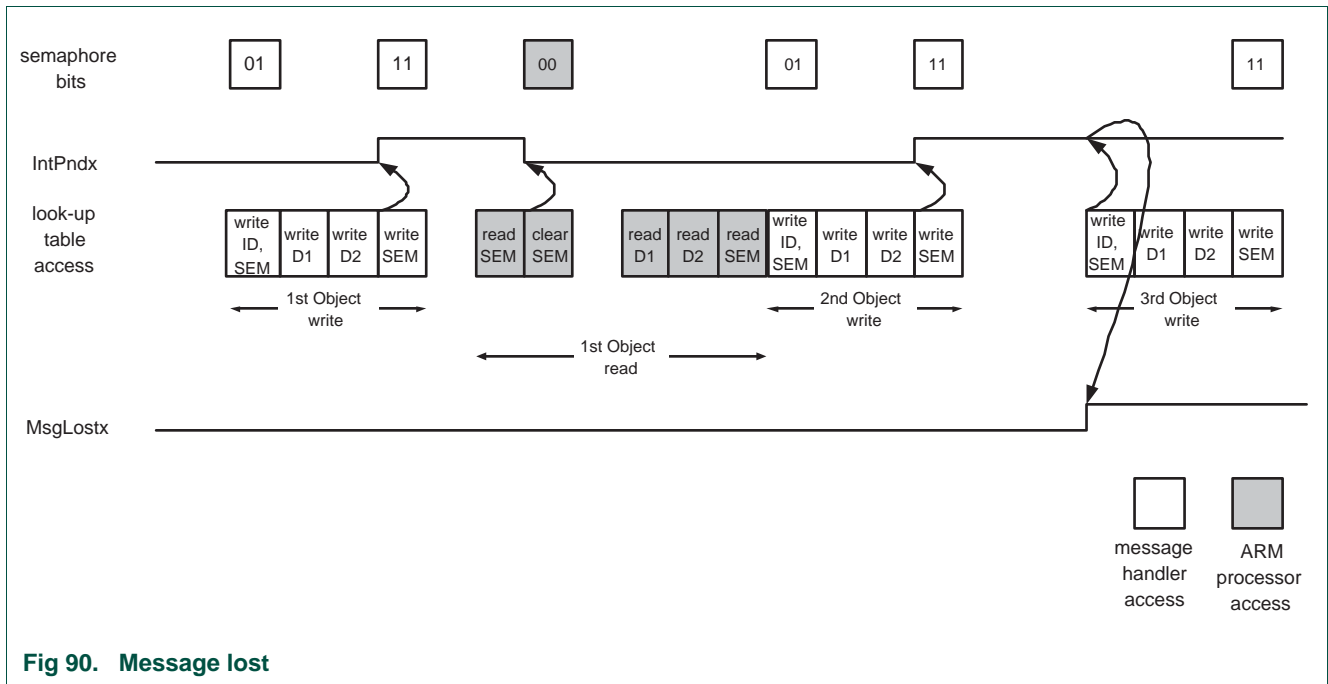


Fig 90. Message lost



20.16.3.3 Scenario 3: Message gets overwritten indicated by Semaphore bits

This scenario is a special case in which the lost message is indicated by the existing semaphore bits. The scenario is entered, if during a Software read of a message object another new message gets stored by the message handler. In this case, the FullCAN Interrupt bit gets set for a second time with the 2<sup>nd</sup> Object write.

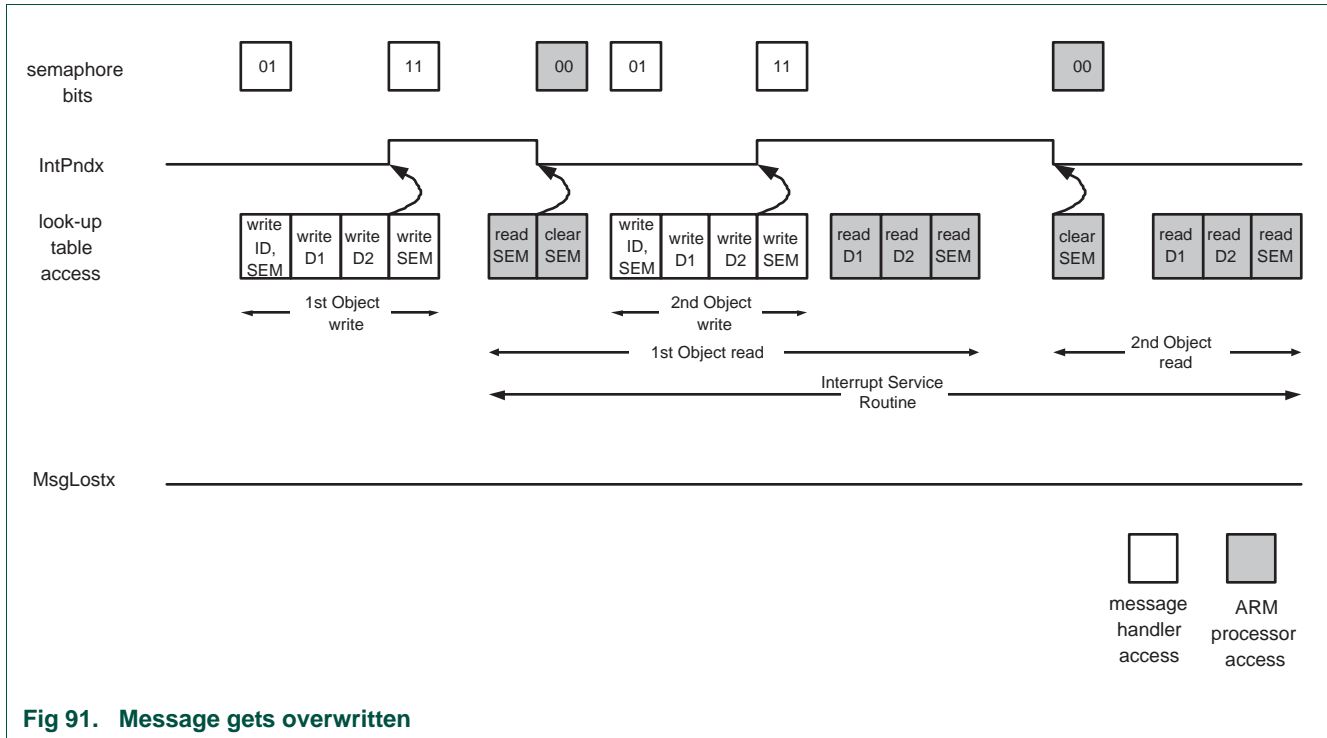


Fig 91. Message gets overwritten

20.16.3.4 Scenario 3.1: Message gets overwritten indicated by Semaphore bits and Message Lost

This scenario is a sub-case to Scenario 3 in which the lost message is indicated by the existing semaphore bits and by Message Lost.

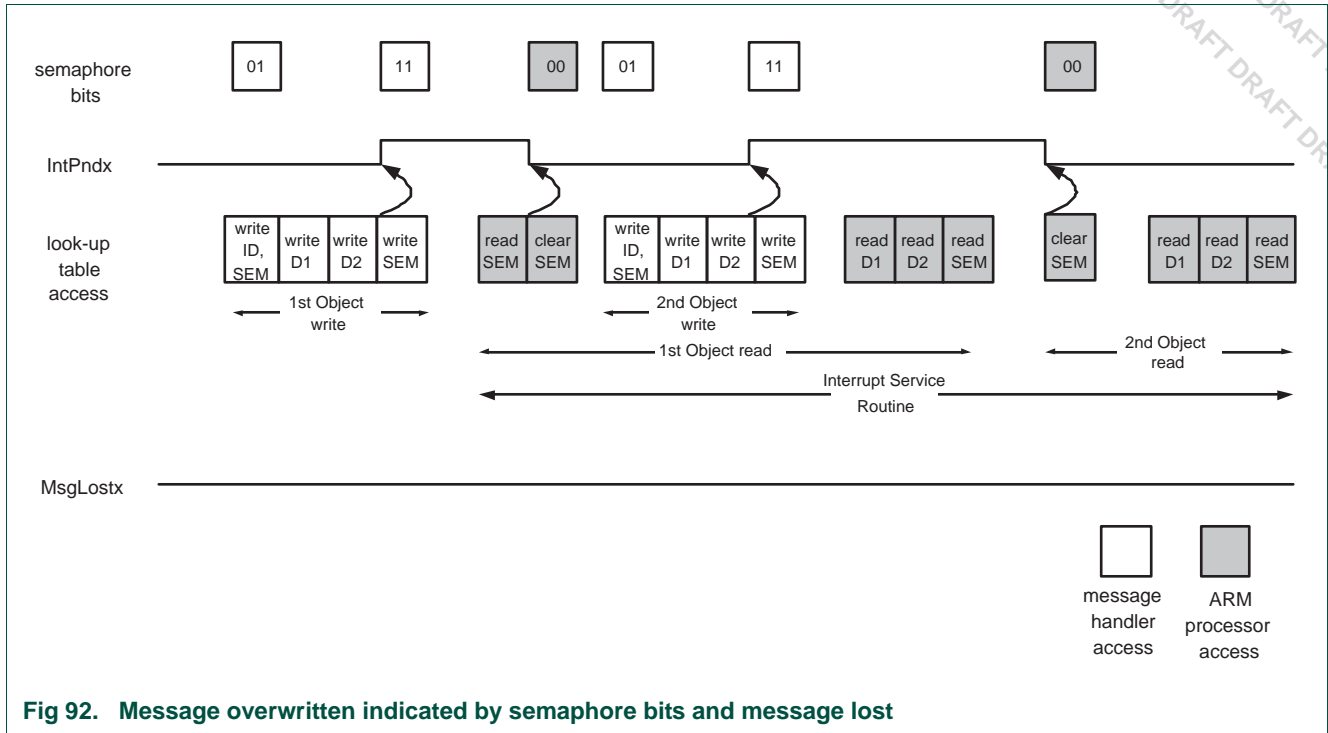


Fig 92. Message overwritten indicated by semaphore bits and message lost

### 20.16.3.5 Scenario 3.2: Message gets overwritten indicated by Message Lost

This scenario is a sub-case to Scenario 3 in which the lost message is indicated by Message Lost.

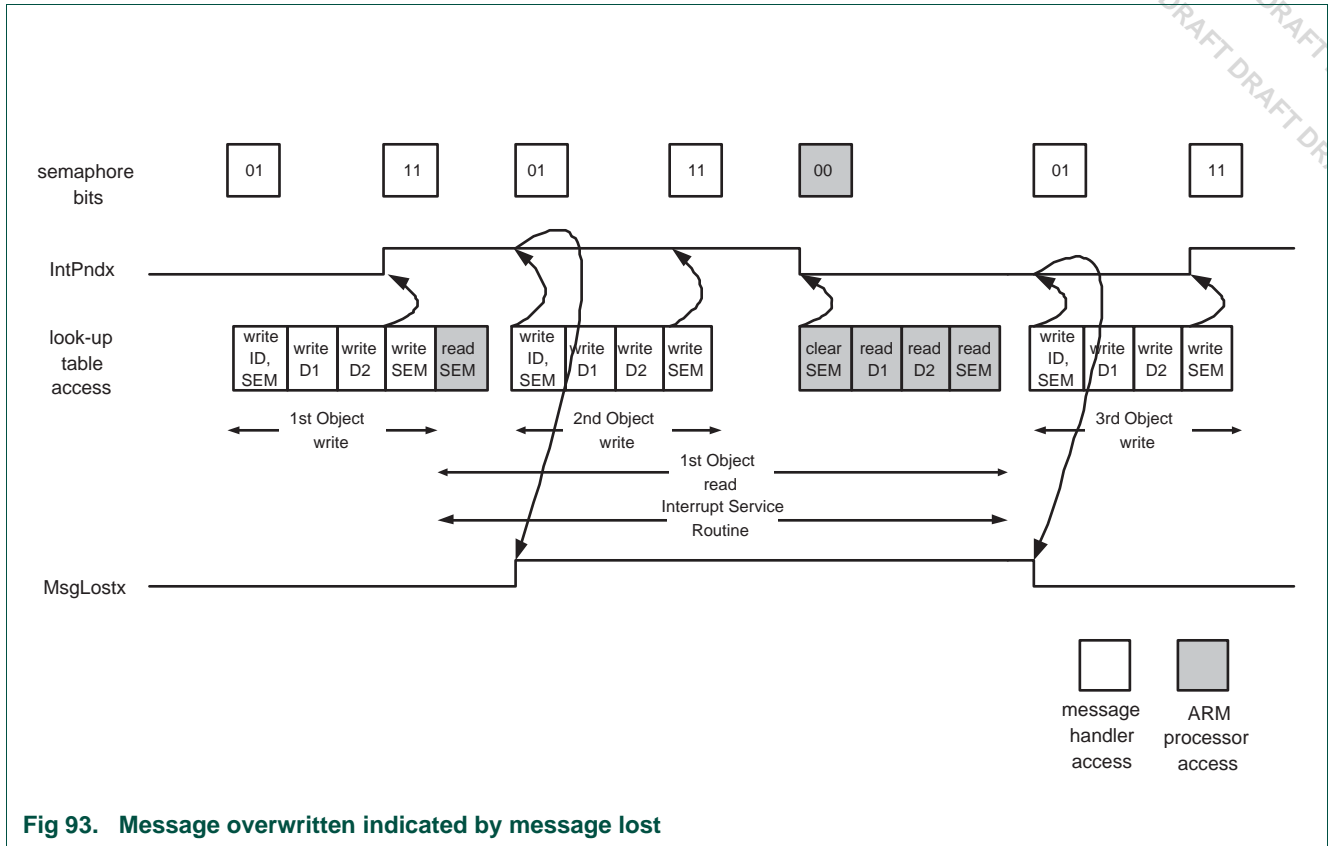


Fig 93. Message overwritten indicated by message lost

### 20.16.3.6 Scenario 4: Clearing Message Lost bit

This scenario is a special case in which the lost message bit of an object gets set during an overwrite of a none read message object (2<sup>nd</sup> Object write). The subsequent read out of that object by Software (1<sup>st</sup> Object read) clears the pending Interrupt. The 3<sup>rd</sup> Object write clears the Message Lost bit. Every “write ID, SEM” clears Message Lost bit if no pending Interrupt of that object is set.

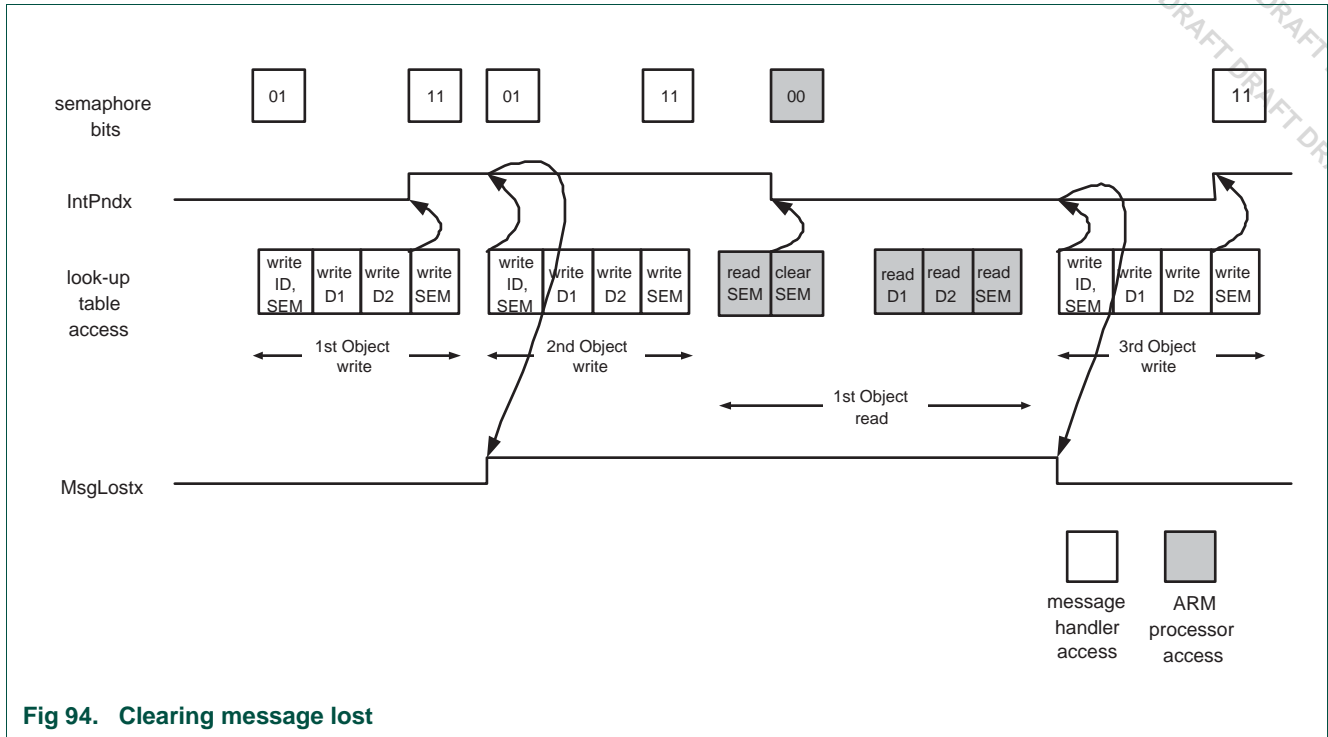


Fig 94. Clearing message lost

## 20.17 Examples of acceptance filter tables and ID index values

### 20.17.1 Example 1: only one section is used

```
SFF_sa < ENDofTable OR
SFF_GRP_sa < ENDofTable OR
EFF_sa < ENDofTable OR
EFF_GRP_sa < ENDofTable
```

The start address of a section is lower than the end address of all programmed CAN identifiers.

### 20.17.2 Example 2: all sections are used

```
SFF_sa < SFF_GRP_sa AND
SFF_GRP_sa < EFF_sa AND
EFF_sa < EFF_GRP_sa AND
EFF_GRP_sa < ENDofTable
```

In cases of a section not being used, the start address has to be set onto the value of the next section start address.

### 20.17.3 Example 3: more than one but not all sections are used

If the SFF group is not used, the start address of the SFF Group Section (SFF\_GRP\_sa register) has to be set to the same value of the next section start address, in this case the start address of the Explicit SFF Section (SFF\_sa register).

In cases where explicit identifiers as well as groups of the identifiers are programmed, a CAN identifier search has to start in the explicit identifier section first. If no match is found, it continues the search in the group of identifier section. By this order it can be guaranteed that in case where an explicit identifier match is found, the succeeding software can directly proceed on this certain message whereas in case of a group of identifier match the succeeding software needs more steps to identify the message.

### 20.17.4 Configuration example 4

Suppose that the five Acceptance Filter address registers contain the values shown in the third column below. In this case each table contains the decimal number of words and entries shown in the next two columns, and the ID Index field of the CANRFS register can return the decimal values shown in the column ID Indexes for CAN messages whose Identifiers match the entries in that table.

**Table 476. Example of Acceptance Filter Tables and ID index Values**

Table	Register	Value	# Words	# Entire	ID Indexes
Standard Individual	SFF_sa	0x040	8 <sub>10</sub>	16 <sub>10</sub>	0-15 <sub>10</sub>
Standard Group	SFF_GRP_sa	0x060	4 <sub>10</sub>	4 <sub>10</sub>	16-19 <sub>10</sub>
Extended Individual	EFF_sa	0x070	8 <sub>10</sub>	16 <sub>10</sub>	20-55 <sub>10</sub>
Extended Group	EFF_GRP_sa	0x100	8 <sub>10</sub>	16 <sub>10</sub>	56-57 <sub>10</sub>
	ENDofTable	0x110			

### 20.17.5 Configuration example 5

[Figure 95](#) below is a more detailed and graphic example of the address registers, table layout, and ID Index values. It shows:

- A Standard Individual table starting at the start of Acceptance Filter RAM and containing 26 Identifiers, followed by:
- A Standard Group table containing 12 ranges of Identifiers, followed by:
- An Extended Individual table containing 3 Identifiers, followed by:
- An Extended Group table containing 2 ranges of Identifiers.

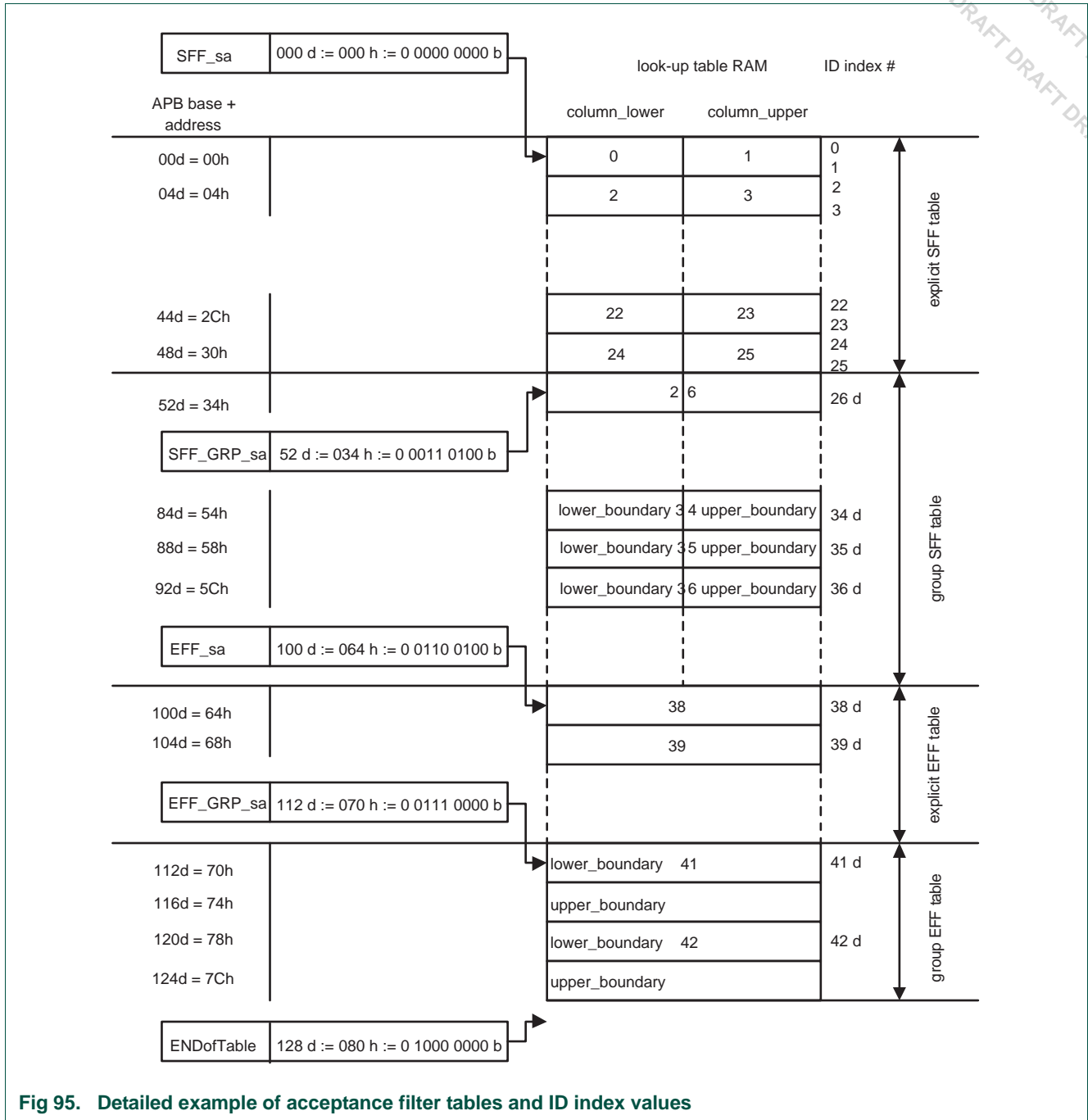


Fig 95. Detailed example of acceptance filter tables and ID index values

### 20.17.6 Configuration example 6

The Table below shows which sections and therefore which types of CAN identifiers are used and activated. The ID-Look-up Table configuration of this example is shown in [Figure 96](#).

**Table 477. Used ID-Look-up Table sections**

ID-Look-up Table Section	Status
FullCAN	not activated
Explicit Standard Frame Format	activated
Group of Standard Frame Format	activated
Explicit Extended Frame Format	activated
Group of Extended Frame Format	activated

**Explicit standard frame format identifier section (11-bit CAN ID):**

The start address of the Explicit Standard Frame Format section is defined in the SFF\_sa register with the value of 0x00. The end of this section is defined in the SFF\_GRP\_sa register. In the Explicit Standard Frame Format section of the ID Look-up Table two CAN Identifiers with their Source CAN Channels (SCC) share one 32-bit word. Not used or disabled CAN Identifiers can be marked by setting the message disable bit.

**Group of standard frame format identifier section (11-bit CAN ID):**

The start address of the Group of Standard Frame Format section is defined with the SFF\_GRP\_sa register with the value of 0x10. The end of this section is defined with the EFF\_sa register. In the Group of Standard Frame Format section two CAN Identifiers with the same Source CAN Channel (SCC) share one 32-bit word and represent a range of CAN Identifiers to be accepted. Bit 31 down to 16 represents the lower boundary and bit 15 down to 0 represents the upper boundary of the range of CAN Identifiers. All Identifiers within this range (including the boundary identifiers) will be accepted. A whole group can be disabled and not used by the acceptance filter by setting the message disable bit in the upper and lower boundary identifier. To provide memory space for four Groups of Standard Frame Format identifiers, the EFF\_sa register value is set to 0x20. The identifier group with the Index 9 of this section is not used and therefore disabled.

**Explicit extended frame format identifier section (29-bit CAN ID, [Figure 96](#))**

The start address of the Explicit Extended Frame Format section is defined with the EFF\_sa register with the value of 0x20. The end of this section is defined with the EFF\_GRP\_sa register. In the explicit Extended Frame Format section only one CAN Identifier with its Source CAN Channel (SCC) is programmed per address line. To provide memory space for four Explicit Extended Frame Format identifiers, the EFF\_GRP\_sa register value is set to 0x30.

**Group of extended frame format identifier section (29-bit CAN ID, [Figure 96](#))**

The start address of the Group of Extended Frame Format is defined with the EFF\_GRP\_sa register with the value of 0x30. The end of this section is defined with the End of Table address register (ENDofTable). In the Group of Extended Frame Format section the boundaries are programmed with a pair of address lines; the first is the lower boundary, the second the upper boundary. To provide memory space for two Groups of Extended Frame Format Identifiers, the ENDofTable register value is set to 0x40.

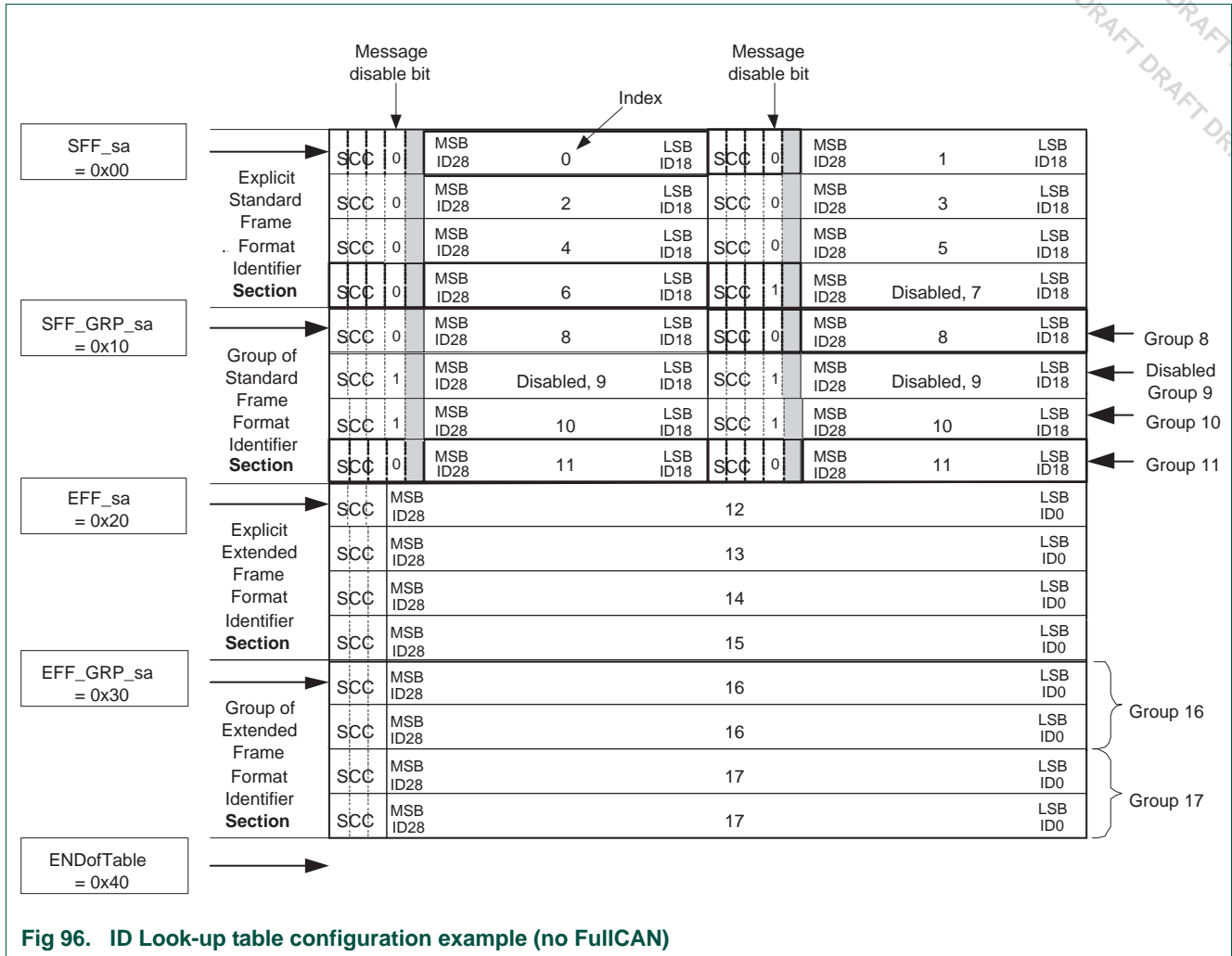


Fig 96. ID Look-up table configuration example (no FullCAN)

### 20.17.7 Configuration example 7

The Table below shows which sections and therefore which types of CAN identifiers are used and activated. The ID-Look-up Table configuration of this example is shown in [Figure 97](#).

This example uses a typical configuration in which FullCAN as well as Explicit Standard Frame Format messages are defined. As described in [Section 20.15.1 "Acceptance filter search algorithm"](#), acceptance filtering takes place in a certain order. With the enabled FullCAN section, the identifier screening process of the acceptance filter starts always in the FullCAN section first, before it continues with the rest of enabled sections.

Table 478. Used ID-Look-up Table sections

ID-Look-up Table Section	Status
FullCAN	activated and enabled
Explicit Standard Frame Format	activated
Group of Standard Frame Format	not activated
Explicit Extended Frame Format	not activated
Group of Extended Frame Format	not activated



### FullCAN explicit standard frame format identifier section (11-bit CAN ID)

The start address of the FullCAN Explicit Standard Frame Format Identifier section is (automatically) set to 0x00. The end of this section is defined in the SFF\_sa register. In the FullCAN ID section only identifiers of FullCAN Object are stored for acceptance filtering. In this section two CAN Identifiers with their Source CAN Channels (SCC) share one 32-bit word. Not used or disabled CAN Identifiers can be marked by setting the message disable bit. The FullCAN Object data for each defined identifier can be found in the FullCAN Message Object section. In case of an identifier match during the acceptance filter process, the received FullCAN message object data is moved from the Receive Buffer of the appropriate CAN Controller into the FullCAN Message Object section. To provide memory space for eight FullCAN, Explicit Standard Frame Format identifiers, the SFF\_sa register value is set to 0x10. The identifier with the Index 1 of this section is not used and therefore disabled.

### Explicit standard frame format identifier section (11-bit CAN ID)

The start address of the Explicit Standard Frame Format section is defined in the SFF\_sa register with the value of 0x10. The end of this section is defined in the End of Table address register (ENDofTable). In the explicit Standard Frame Format section of the ID Look-up Table two CAN Identifiers with their Source CAN Channel (SCC) share one 32-bit word. Not used or disabled CAN Identifiers can be marked by setting the message disable bit. To provide memory space for eight Explicit Standard Frame Format identifiers, the ENDofTable register value is set to 0x20.

### FullCAN message object data section

The start address of the FullCAN Message Object Data section is defined with the ENDofTable register. The number of enabled FullCAN identifiers is limited to the available memory space in the FullCAN Message Object Data section. Each defined FullCAN Message needs three address lines for the Message Data in the FullCAN Message Object Data section. The FullCAN Message Object section is organized in that way, that each Index number of the FullCAN Identifier section corresponds to a Message Object Number in the FullCAN Message Object section.

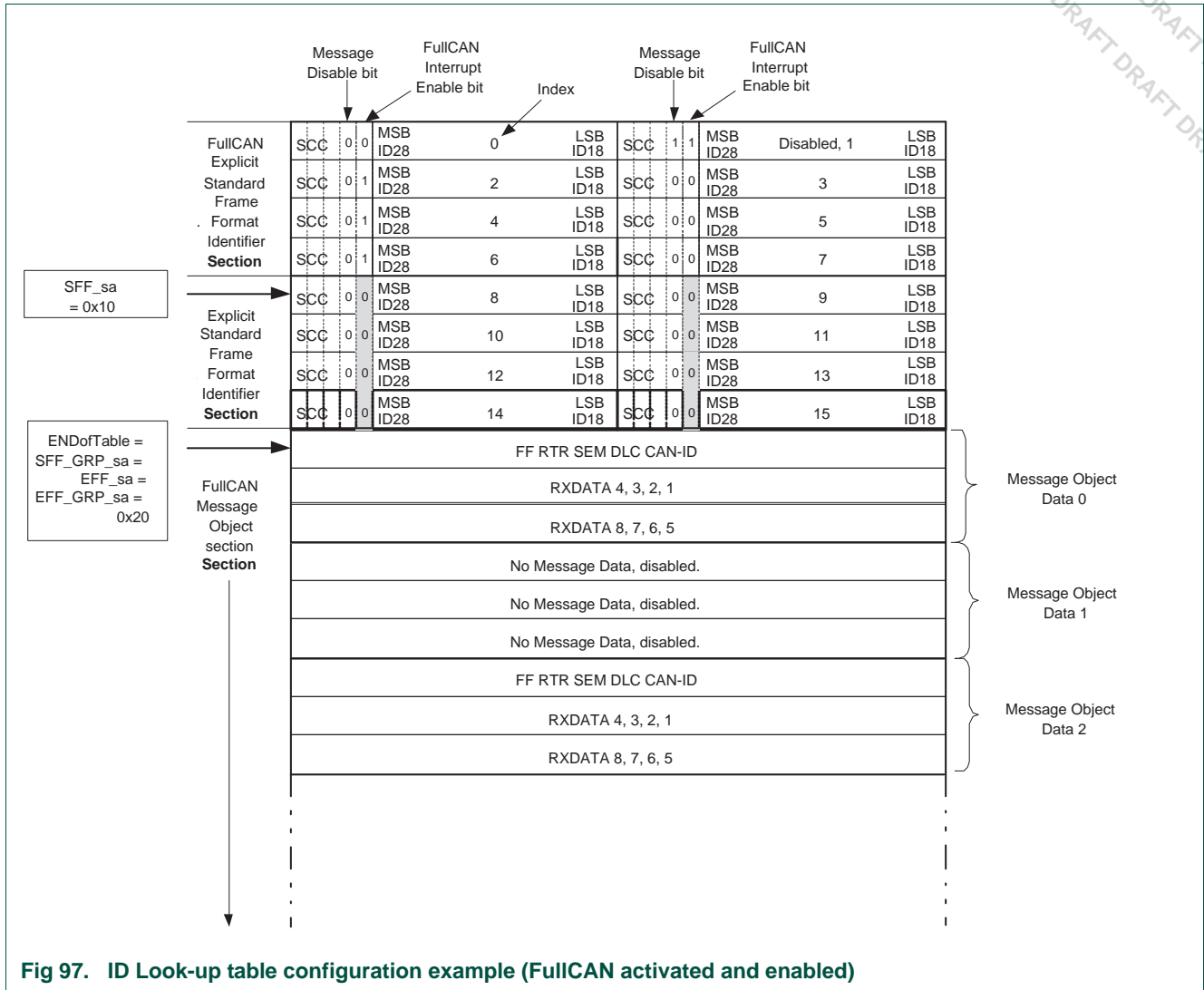


Fig 97. ID Look-up table configuration example (FullCAN activated and enabled)

### 20.17.8 Look-up table programming guidelines

All identifier sections of the ID Look-up Table have to be programmed in such a way, that each active section is organized as a sorted list or table with an increasing order of the Source CAN Channel (SCC) together with CAN Identifier in each section.

SCC value equals CAN\_controller - 1, i.e., SCC = 0 matches CAN1 and SCC = 1 matches CAN2.

In cases, where a syntax error in the ID Look-up Table is encountered, the Look-up Table address of the incorrect line is made available in the Look-up Table Error Address Register (LUTerrAd).

The reporting process in the Look-up Table Error Address Register (LUTerrAd) is a “run-time” process. Only those address lines with syntax error are reported, which were passed through the acceptance filtering process.

The following general rules for programming the Look-up Table apply:

- Each section has to be organized as a sorted list or table with an increasing order of the Source CAN Channel (SCC) in conjunction with the CAN Identifier (there is no exception for disabled identifiers).
- The upper and lower bound in a Group of Identifiers definition has to be from the same Source CAN Channel.
- To disable a Group of Identifiers the message disable bit has to be set for both, the upper and lower bound.

### 21.1 Basic configuration

The three SSP interfaces, SSP0, SSP1, and SSP2 are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCSSP0 to enable SSP0 and bit PCSSP1 to enable SSP1.  
**Remark:** On reset, SSP interfaces 0 and 1 are enabled (PCSSP0/1 = 1), while SSP2 is disabled (PCSSP2 = 0).
2. Peripheral clock: The SSPs operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#). In master mode, the clock must be scaled down (see [Section 21.6.5](#)).
3. Pins: Select the SSP pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: Interrupts are enabled in the SSP0IMSC register for SSP0 and SSP1IMSC register for SSP1 [Table 486](#). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register, see [Table 41](#).
5. Initialization: There are two control registers for each of the SSP ports to be configured: SSP0CR0 and SSP0CR1 for SSP0, SSP1CR0 and SSP1CR1 for SSP1, SSP2CR0 and SSP2CR1 for SSP2. See [Section 21.6.1](#) and [Section 21.6.2](#).
6. DMA: The Rx and Tx FIFOs of the SSP interfaces can be connected to the GPDMA controller (see [Section 21.6.10](#)). For GPDMA system connections, see [Table 655](#).

### 21.2 Features

- Compatible with Motorola SPI, 4-wire TI SSI, and National Semiconductor Microwire buses.
- Synchronous Serial Communication.
- Master or slave operation.
- 8 frame FIFOs for both transmit and receive.
- 4 to 16 bit data frame.
- DMA transfers supported by GPDMA.

### 21.3 Description

The SSP is a Synchronous Serial Port (SSP) controller capable of operation on a SPI, 4-wire SSI, or Microwire bus. It can interact with multiple masters and slaves on the bus. Only a single master and a single slave can communicate on the bus during a given data transfer. Data transfers are in principle full duplex, with frames of 4 to 16 bits of data flowing from the master to the slave and from the slave to the master. In practice it is often the case that only one of these data flows carries meaningful data.

The LPC178x/177x has three Synchronous Serial Port controllers -- SSP0, SSP1, and SSP2.

## 21.4 Pin descriptions

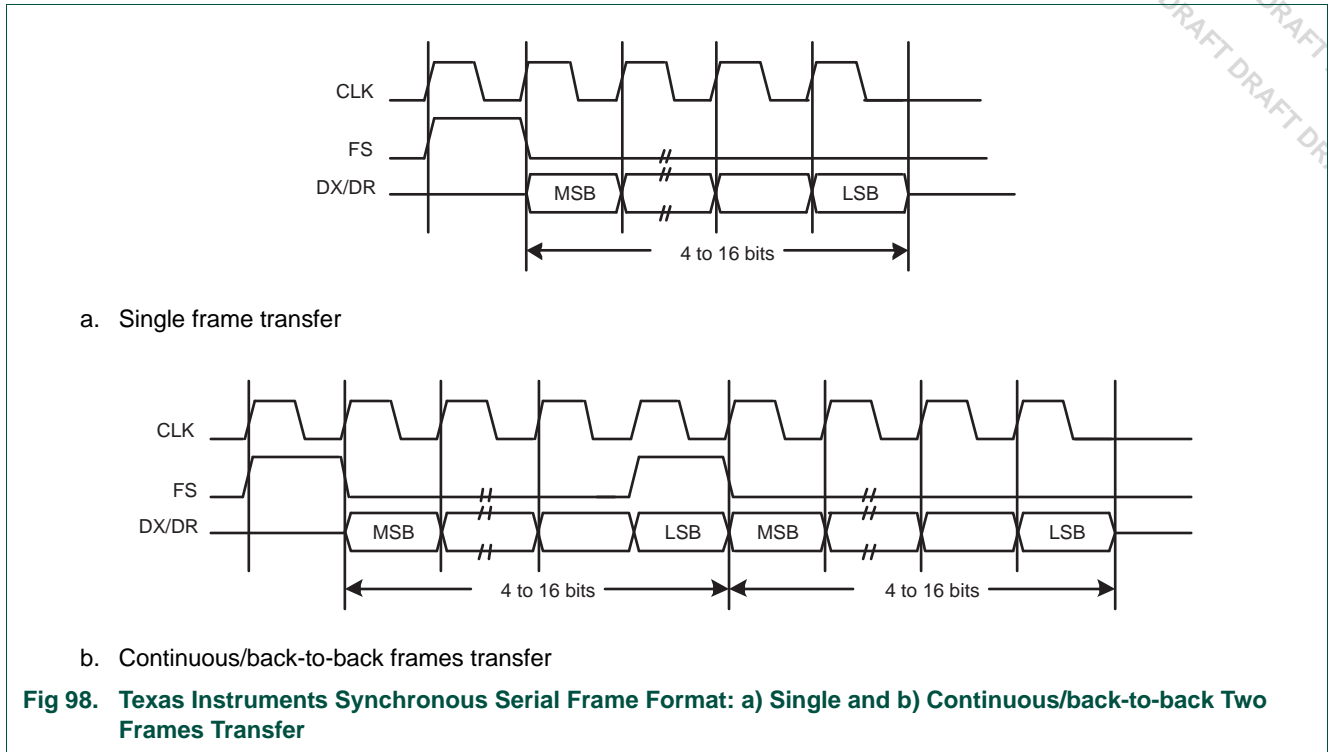
Table 479. SSP pin descriptions

Pin Name	Type	Interface pin name/function			Pin Description
		SPI	SSI	Microwire	
SCK0/1/2	I/O	SCK	CLK	SK	<b>Serial Clock.</b> SCK/CLK/SK is a clock signal used to synchronize the transfer of data. It is driven by the master and received by the slave. When the SPI interface is used, the clock is programmable to be active-high or active-low, otherwise it is always active-high. SCK only switches during a data transfer. Any other time, the SSPn interface either holds it in its inactive state, or does not drive it (leaves it in high-impedance state).
SSEL0/1/2	I/O	SSEL	FS	CS	<b>Frame Sync/Slave Select.</b> When the SSPn interface is a bus master, it drives this signal to an active state before the start of serial data, and then releases it to an inactive state after the serial data has been sent. The active state of this signal can be high or low depending upon the selected bus and mode. When the SSPn is a bus slave, this signal qualifies the presence of data from the Master, according to the protocol in use.  When there is just one bus master and one bus slave, the Frame Sync or Slave Select signal from the Master can be connected directly to the slave's corresponding input. When there is more than one slave on the bus, further qualification of their Frame Select/Slave Select inputs will typically be necessary to prevent more than one slave from responding to a transfer.
MISO0/1/2	I/O	MISO	DR(M) DX(S)	SI(M) SO(S)	<b>Master In Slave Out.</b> The MISO signal transfers serial data from the slave to the master. When the SSPn is a slave, serial data is output on this signal. When the SSPn is a master, it clocks in serial data from this signal. When the SSPn is a slave and is not selected by FS/SSEL, it does not drive this signal (leaves it in high-impedance state).
MOSI0/1/2	I/O	MOSI	DX(M) DR(S)	SO(M) SI(S)	<b>Master Out Slave In.</b> The MOSI signal transfers serial data from the master to the slave. When the SSPn is a master, it outputs serial data on this signal. When the SSPn is a slave, it clocks in serial data from this signal.

## 21.5 Bus description

### 21.5.1 Texas Instruments synchronous serial frame format

[Figure 98](#) shows the 4-wire Texas Instruments synchronous serial frame format supported by the SSP module.



**Fig 98. Texas Instruments Synchronous Serial Frame Format: a) Single and b) Continuous/back-to-back Two Frames Transfer**

For device configured as a master in this mode, CLK and FS are forced LOW, and the transmit data line DX is tri-stated whenever the SSP is idle. Once the bottom entry of the transmit FIFO contains data, FS is pulsed HIGH for one CLK period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of CLK, the MSB of the 4-bit to 16-bit data frame is shifted out on the DX pin. Likewise, the MSB of the received data is shifted onto the DR pin by the off-chip serial slave device.

Both the SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each CLK. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of CLK after the LSB has been latched.

### 21.5.2 SPI frame format

The SPI interface is a four-wire interface where the SSEL signal behaves as a slave select. The main feature of the SPI format is that the inactive state and phase of the SCK signal are programmable through the CPOL and CPHA bits within the SSPCR0 control register.

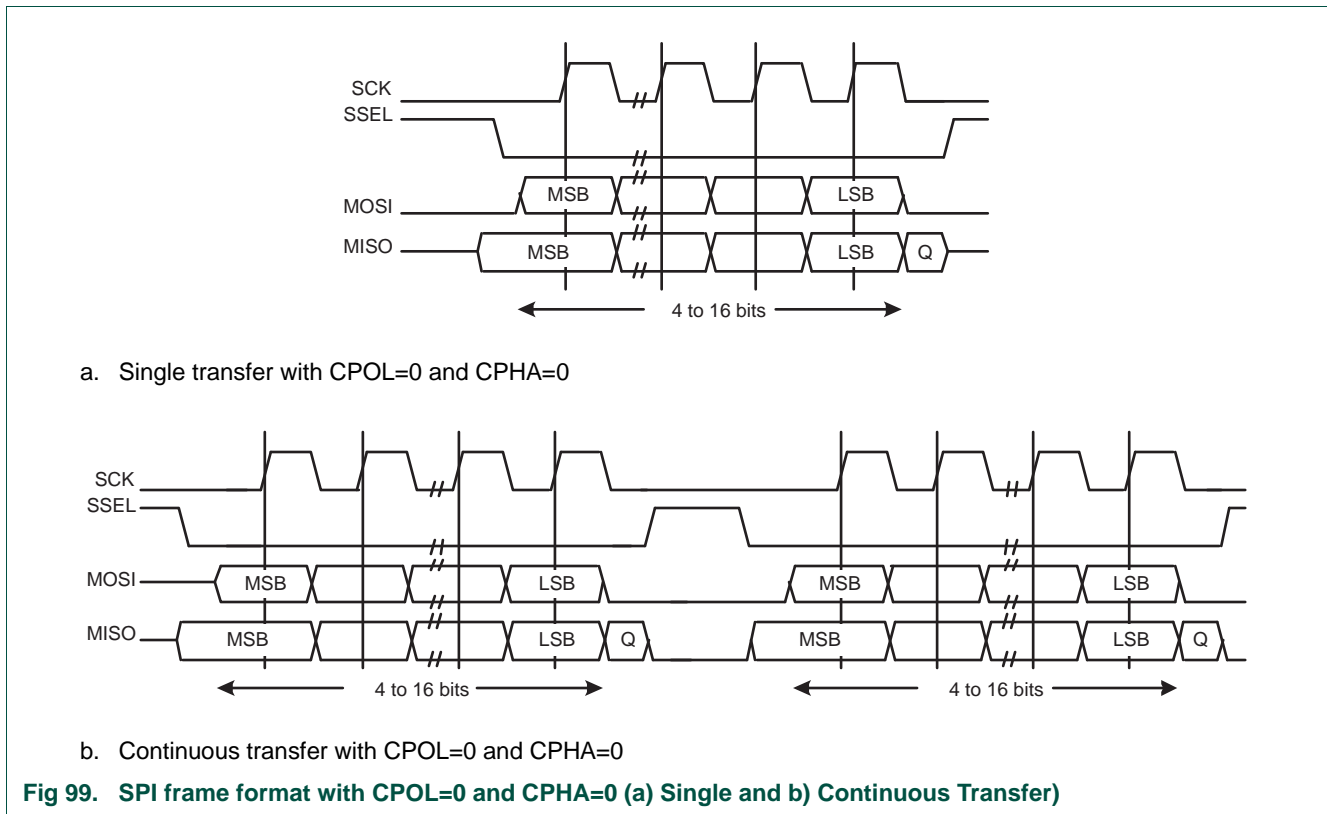
#### 21.5.2.1 Clock Polarity (CPOL) and Phase (CPHA) control

When the CPOL clock polarity control bit is 0, it produces a steady state low value on the SCK pin. If the CPOL clock polarity control bit is 1, a steady state high value is placed on the CLK pin when data is not being transferred.

The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is 0, data is captured on the first clock edge transition. If the CPHA clock phase control bit is 1, data is captured on the second clock edge transition.

**21.5.2.2 SPI format with CPOL=0,CPHA=0**

Single and continuous transmission signal sequences for SPI format with CPOL = 0, CPHA = 0 are shown in [Figure 99](#).



**Fig 99. SPI frame format with CPOL=0 and CPHA=0 (a) Single and b) Continuous Transfer)**

In this configuration, during idle periods:

- The CLK signal is forced LOW.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. This causes slave data to be enabled onto the MISO input line of the master. Master's MOSI is enabled.

One half SCK period later, valid master data is transferred to the MOSI pin. Now that both the master and slave data have been set, the SCK master clock pin goes HIGH after one further half SCK period.

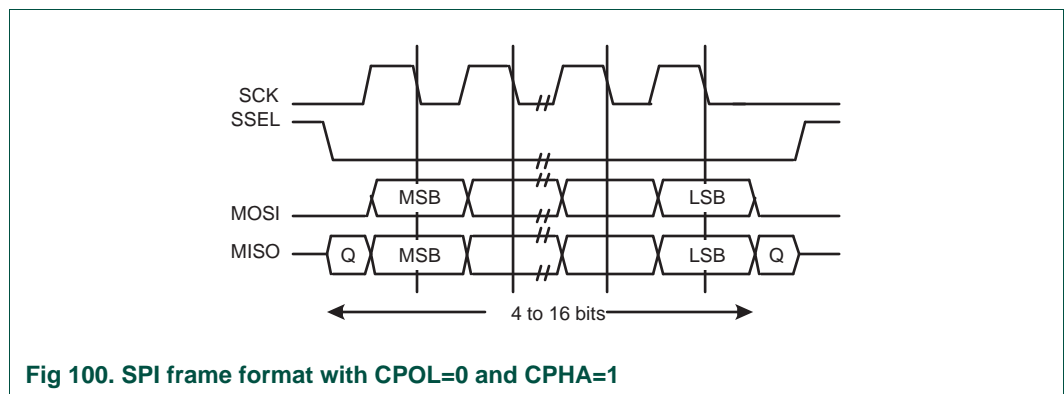
The data is now captured on the rising and propagated on the falling edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

**21.5.2.3 SPI format with CPOL=0,CPHA=1**

The transfer signal sequence for SPI format with CPOL = 0, CPHA = 1 is shown in [Figure 100](#), which covers both single and continuous transfers.



**Fig 100. SPI frame format with CPOL=0 and CPHA=1**

In this configuration, during idle periods:

- The CLK signal is forced LOW.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master’s MOSI pin is enabled. After a further one half SCK period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the SCK is enabled with a rising edge transition.

Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

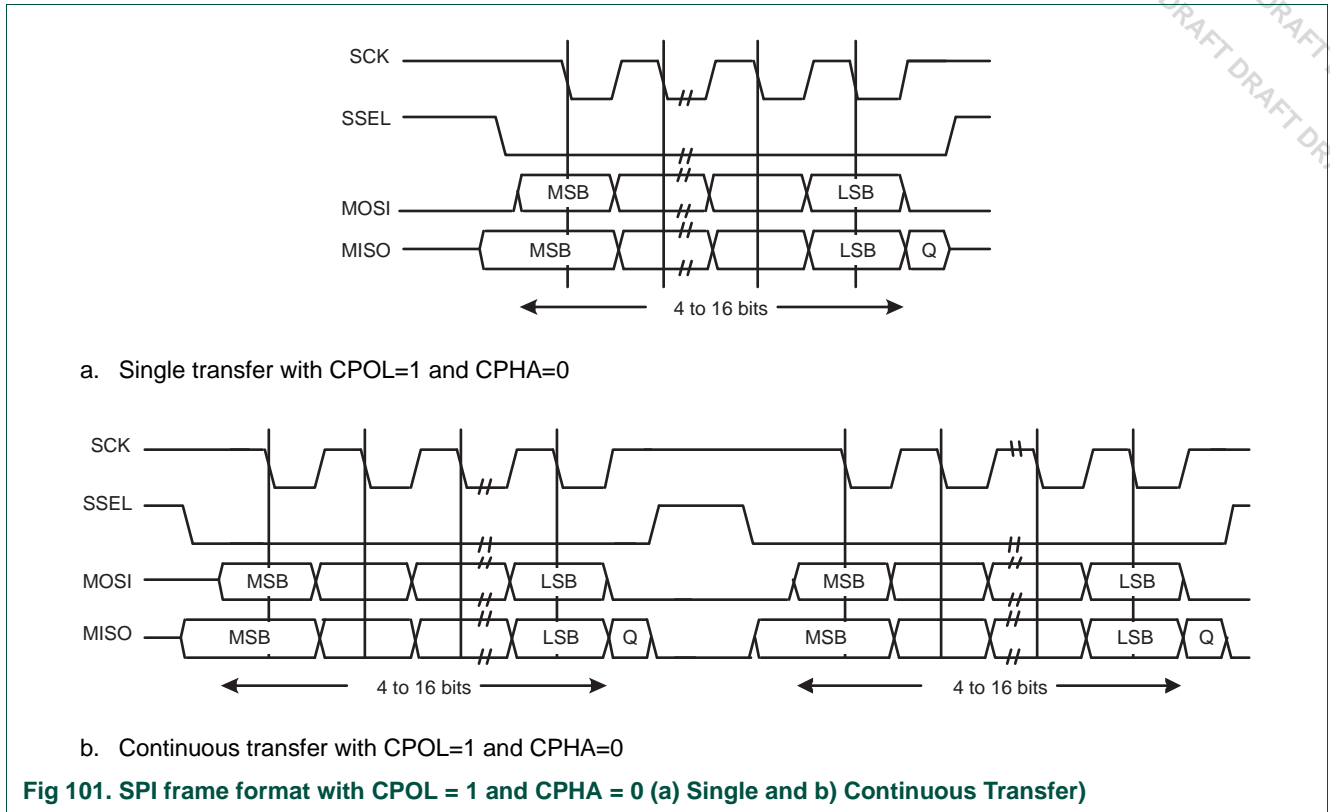
In the case of a single word transfer, after all bits have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

For continuous back-to-back transfers, the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

**21.5.2.4 SPI format with CPOL = 1,CPHA = 0**

Single and continuous transmission signal sequences for SPI format with CPOL=1, CPHA=0 are shown in [Figure 101](#).





In this configuration, during idle periods:

- The CLK signal is forced HIGH.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW, which causes slave data to be immediately transferred onto the MISO line of the master. Master's MOSI pin is enabled.

One half period later, valid master data is transferred to the MOSI line. Now that both the master and slave data have been set, the SCK master clock pin becomes LOW after one further half SCK period. This means that data is captured on the falling edges and be propagated on the rising edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word are transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

21.5.2.5 SPI format with CPOL = 1, CPHA = 1

The transfer signal sequence for SPI format with CPOL = 1, CPHA = 1 is shown in [Figure 102](#), which covers both single and continuous transfers.

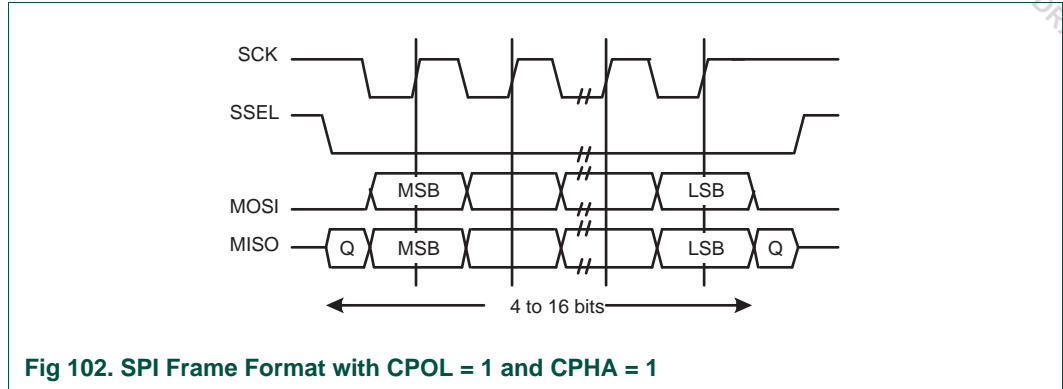


Fig 102. SPI Frame Format with CPOL = 1 and CPHA = 1

In this configuration, during idle periods:

- The CLK signal is forced HIGH.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master’s MOSI is enabled. After a further one half SCK period, both master and slave data are enabled onto their respective transmission lines. At the same time, the SCK is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the SCK signal.

After all bits have been transferred, in the case of a single word transmission, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured. For continuous back-to-back transmissions, the SSEL pins remains in its active LOW state, until the final bit of the last word has been captured, and then returns to its idle state as described above. In general, for continuous back-to-back transfers the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

21.5.3 National Semiconductor Microwire frame format

[Figure 103](#) shows the Microwire frame format for a single frame. [Figure 104](#) shows the same format when back-to-back frames are transmitted.

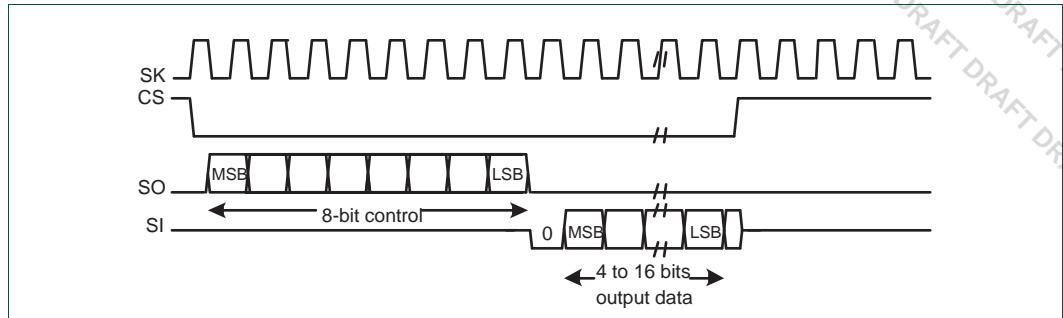


Fig 103. Microwire frame format (single transfer)

Microwire format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSP to the off-chip slave device. During this transmission, no incoming data is received by the SSP. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- The SK signal is forced LOW.
- CS is forced HIGH.
- The transmit data line SO is arbitrarily forced LOW.

A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of CS causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the SO pin. CS remains LOW for the duration of the frame transmission. The SI pin remains tri-stated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SK. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSP. Each bit is driven onto SI line on the falling edge of SK. The SSP in turn latches each bit on the rising edge of SK. At the end of the frame, for single transfers, the CS signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

**Note:** The off-chip slave device can tri-state the receive line either on the falling edge of SK after the LSB has been latched by the receive shifter, or when the CS pin goes HIGH.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the CS line is continuously asserted (held LOW) and transmission of data occurs back to back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge SK, after the LSB of the frame has been latched into the SSP.

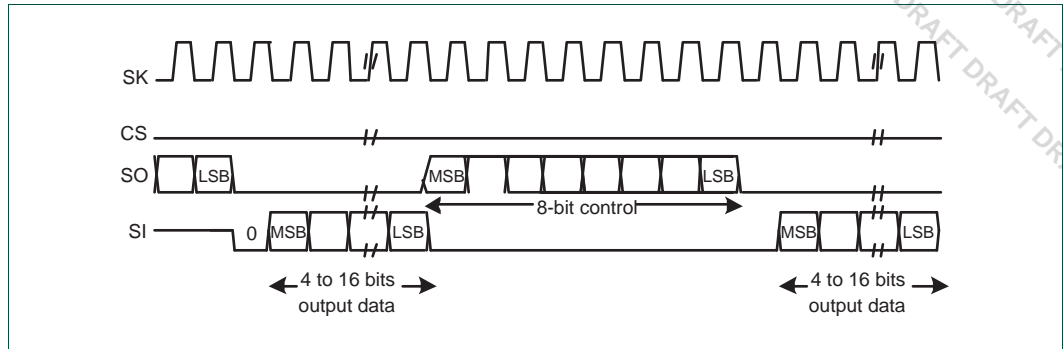


Fig 104. Microwire frame format (continuous transfers)

### 21.5.3.1 Setup and hold time requirements on CS with respect to SK in Microwire mode

In the Microwire mode, the SSP slave samples the first bit of receive data on the rising edge of SK after CS has gone LOW. Masters that drive a free-running SK must ensure that the CS signal has sufficient setup and hold margins with respect to the rising edge of SK.

Figure 105 illustrates these setup and hold time requirements. With respect to the SK rising edge on which the first bit of receive data is to be sampled by the SSP slave, CS must have a setup of at least two times the period of SK on which the SSP operates. With respect to the SK rising edge previous to this edge, CS must have a hold of at least one SK period.

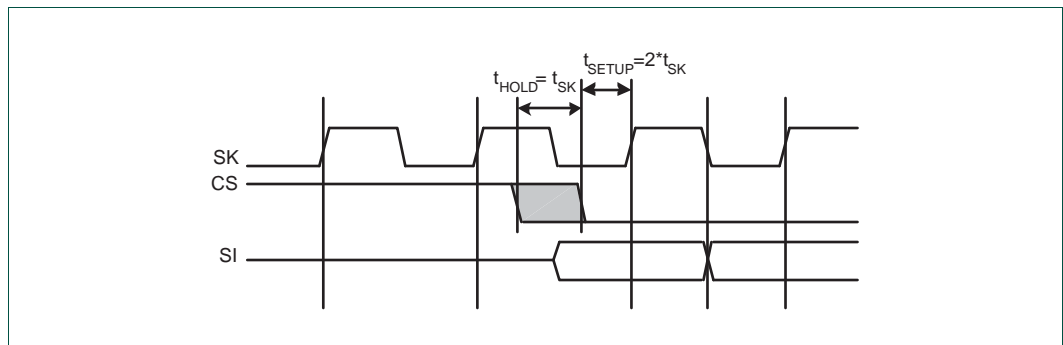


Fig 105. Microwire frame format setup and hold details

## 21.6 Register description

The register addresses of the SSP controllers addresses are shown in [Table 480](#).

**Table 480. SSP Register Map**

Generic Name	Description	Access	Reset Value <sup>[1]</sup>	SSPn Register Name & Address	Table
CR0	Control Register 0. Selects the serial clock rate, bus type, and data size.	R/W	0	SSP0CR0 - 0x4008 8000 SSP1CR0 - 0x4003 0000 SSP2CR0 - 0x400A C000	<a href="#">481</a>
CR1	Control Register 1. Selects master/slave and other modes.	R/W	0	SSP0CR1 - 0x4008 8004 SSP1CR1 - 0x4003 0004 SSP2CR1 - 0x400A 4004	<a href="#">482</a>
DR	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	R/W	0	SSP0DR - 0x4008 8008 SSP1DR - 0x4003 0008 SSP2DR - 0x400A C008	<a href="#">483</a>
SR	Status Register	RO		SSP0SR - 0x4008 800C SSP1SR - 0x4003 000C SSP2SR - 0x400A C00C	<a href="#">484</a>
CPSR	Clock Prescale Register	R/W	0	SSP0CPSR - 0x4008 8010 SSP1CPSR - 0x4003 0010 SSP2CPSR - 0x400A C010	<a href="#">485</a>
IMSC	Interrupt Mask Set and Clear Register	R/W	0	SSP0IMSC - 0x4008 8014 SSP1IMSC - 0x4003 0014 SSP2IMSC - 0x400A C014	<a href="#">486</a>
RIS	Raw Interrupt Status Register	R/W		SSP0RIS - 0x4008 8018 SSP1RIS - 0x4003 0018 SSP2RIS - 0x400A C018	<a href="#">487</a>
MIS	Masked Interrupt Status Register	R/W	0	SSP0MIS - 0x4008 801C SSP1MIS - 0x4003 001C SSP2MIS - 0x400A C01C	<a href="#">488</a>
ICR	SSPICR Interrupt Clear Register	R/W	NA	SSP0ICR - 0x4008 8020 SSP1ICR - 0x4003 0020 SSP2ICR - 0x400A C020	<a href="#">489</a>
DMACR	DMA Control Register	R/W	0	SSP0DMACR - 0x4008 8024 SSP1DMACR - 0x4003 0024 SSP2DMACR - 0x400A C024	<a href="#">490</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

**21.6.1 SSPn Control Register 0 (SSP0CR0 - 0x4008 8000, SSP1CR0 - 0x4003 0000, SSP2CR0 - 0x400A C000)**

This register controls the basic operation of the SSP controller.

**Table 481: SSPn Control Register 0 (SSP0CR0 - address 0x4008 8000, SSP1CR0 - 0x4003 0000, SSP2CR0 - 0x400A C000) bit description**

Bit	Symbol	Value	Description	Reset Value
3:0	DSS		Data Size Select. This field controls the number of bits transferred in each frame. Values 0000-0010 are not supported and should not be used.	0000
		0011	4-bit transfer	
		0100	5-bit transfer	
		0101	6-bit transfer	
		0110	7-bit transfer	
		0111	8-bit transfer	
		1000	9-bit transfer	
		1001	10-bit transfer	
		1010	11-bit transfer	
		1011	12-bit transfer	
		1100	13-bit transfer	
		1101	14-bit transfer	
		1110	15-bit transfer	
5:4	FRF		Frame Format.	00
		00	SPI	
		01	TI	
		10	Microwire	
		11	This combination is not supported and should not be used.	
6	CPOL		Clock Out Polarity. This bit is only used in SPI mode.	0
		0	SSP controller maintains the bus clock low between frames.	
	1	SSP controller maintains the bus clock high between frames.		
7	CPHA		Clock Out Phase. This bit is only used in SPI mode.	0
		0	SSP controller captures serial data on the first clock transition of the frame, that is, the transition <b>away from</b> the inter-frame state of the clock line.	
	1	SSP controller captures serial data on the second clock transition of the frame, that is, the transition <b>back to</b> the inter-frame state of the clock line.		
15:8	SCR		Serial Clock Rate. The number of prescaler-output clocks per bit on the bus, minus one. Given that CPSDVSR is the prescale divider, and the APB clock PCLK clocks the prescaler, the bit frequency is $PCLK / (CPSDVSR \times [SCR+1])$ .	0
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

**21.6.2 SSPn Control Register 1 (SSP0CR1 - 0x4008 8004, SSP1CR1 - 0x4003 0004, SSP2CR1 - 0x400A C004)**

This register controls certain aspects of the operation of the SSP controller.

**Table 482: SSPn Control Register 1 (SSP0CR1 - address 0x4008 8004, SSP1CR1 - 0x4003 0004, SSP2CR1 - 0x400A C004) bit description**

Bit	Symbol	Value	Description	Reset Value
0	LBM		Loop Back Mode.	0
		0	During normal operation.	
		1	Serial input is taken from the serial output (MOSI or MISO) rather than the serial input pin (MISO or MOSI respectively).	
1	SSE		SSP Enable.	0
		0	The SSP controller is disabled.	
		1	The SSP controller will interact with other devices on the serial bus. Software should write the appropriate control information to the other SSP registers and interrupt controller registers, before setting this bit.	
2	MS		Master/Slave Mode. This bit can only be written when the SSE bit is 0.	0
		0	The SSP controller acts as a master on the bus, driving the SCLK, MOSI, and SSEL lines and receiving the MISO line.	
		1	The SSP controller acts as a slave on the bus, driving MISO line and receiving SCLK, MOSI, and SSEL lines.	
3	SOD		Slave Output Disable. This bit is relevant only in slave mode (MS = 1). If it is 1, this blocks this SSP controller from driving the transmit data line (MISO).	0
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 21.6.3 SSPn Data Register (SSP0DR - 0x4008 8008, SSP1DR - 0x4003 0008, SSP2DR - 0x400A C008)

Software can write data to be transmitted to this register, and read data that has been received.

**Table 483: SSPn Data Register (SSP0DR - address 0x4008 8008, SSP1DR - 0x4003 0008, SSP2DR - 0x400A C008) bit description**

Bit	Symbol	Description	Reset Value
15:0	DATA	<p><b>Write:</b> software can write data to be sent in a future frame to this register whenever the TNF bit in the Status register is 1, indicating that the Tx FIFO is not full. If the Tx FIFO was previously empty and the SSP controller is not busy on the bus, transmission of the data will begin immediately. Otherwise the data written to this register will be sent as soon as all previous data has been sent (and received). If the data length is less than 16 bits, software must right-justify the data written to this register.</p> <p><b>Read:</b> software can read data from this register whenever the RNE bit in the Status register is 1, indicating that the Rx FIFO is not empty. When software reads this register, the SSP controller returns data from the least recent frame in the Rx FIFO. If the data length is less than 16 bits, the data is right-justified in this field with higher order bits filled with 0s.</p>	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

### 21.6.4 SSPn Status Register (SSP0SR - 0x4008 800C, SSP1SR - 0x4003 000C, SSP2SR - 0x400A C00C)

This read-only register reflects the current status of the SSP controller.

**Table 484: SSPn Status Register (SSP0SR - address 0x4008 800C, SSP1SR - 0x4003 000C, SSP2SR - 0x400A C00C) bit description**

Bit	Symbol	Description	Reset Value
0	TFE	Transmit FIFO Empty. This bit is 1 if the Transmit FIFO is empty, 0 if not.	1
1	TNF	Transmit FIFO Not Full. This bit is 0 if the Tx FIFO is full, 1 if not.	1
2	RNE	Receive FIFO Not Empty. This bit is 0 if the Receive FIFO is empty, 1 if not.	0
3	RFF	Receive FIFO Full. This bit is 1 if the Receive FIFO is full, 0 if not.	0
4	BSY	Busy. This bit is 0 if the SSPn controller is idle, or 1 if it is currently sending/receiving a frame and/or the Tx FIFO is not empty.	0
31:5	-	Reserved. The value read from a reserved bit is not defined.	NA

### 21.6.5 SSPn Clock Prescale Register (SSP0CPSR - 0x4008 8010, SSP1CPSR - 0x4003 0010, SSP2CPSR - 0x400A C010)

This register controls the factor by which the Prescaler divides PCLK to yield the prescaler clock that is, in turn, divided by the SCR factor in SSPnCR0, to determine the bit clock.

**Table 485: SSPn Clock Prescale Register (SSP0CPSR - address 0x4008 8010, SSP1CPSR - 0x4003 0010, SSP2CPSR - 0x400A C010) bit description**

Bit	Symbol	Description	Reset Value
7:0	CPSDVSR	This even value between 2 and 254, by which PCLK is divided to yield the prescaler output clock. Bit 0 always reads as 0.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

**Important:** the SSPnCPSR value must be properly initialized or the SSP controller will not be able to transmit data correctly.

In Slave mode, the SSP clock rate provided by the master must not exceed 1/12 of the peripheral clock selected in [Section 4.6.4](#). The content of the SSPnCPSR register is not relevant.

In master mode,  $CPSDVSR_{min} = 2$  or larger (even numbers only).

### 21.6.6 SSPn Interrupt Mask Set/Clear Register (SSP0IMSC - 0x4008 8014, SSP1IMSC - 0x4003 0014, SSP2IMSC - 0x400A C014)

This register controls whether each of the four possible interrupt conditions in the SSP controller are enabled. Note that ARM uses the word “masked” in the opposite sense from classic computer terminology, in which “masked” meant “disabled”. ARM uses the word “masked” to mean “enabled”. To avoid confusion we will not use the word “masked”.

**Table 486: SSPn Interrupt Mask Set/Clear register (SSP0IMSC - address 0x4008 8014, SSP1IMSC - 0x4003 0014, SSP2IMSC - 0x400A C014) bit description**

Bit	Symbol	Description	Reset Value
0	RORIM	Software should set this bit to enable interrupt when a Receive Overrun occurs, that is, when the Rx FIFO is full and another frame is completely received. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs.	0
1	RTIM	Software should set this bit to enable interrupt when a Receive Timeout condition occurs. A Receive Timeout occurs when the Rx FIFO is not empty, and has not been read for a period of 32 bit times.	0



**Table 486: SSPn Interrupt Mask Set/Clear register (SSP0IMSC - address 0x4008 8014, SSP1IMSC - 0x4003 0014, SSP2IMSC - 0x400A C014) bit description**

Bit	Symbol	Description	Reset Value
2	RXIM	Software should set this bit to enable interrupt when the Rx FIFO is at least half full.	0
3	TXIM	Software should set this bit to enable interrupt when the Tx FIFO is at least half empty.	0
31:4	-	Reserved. Read value is undefined, only zero should be written.	NA

### 21.6.7 SSPn Raw Interrupt Status Register (SSP0RIS - 0x4008 8018, SSP1RIS - 0x4003 0018, SSP2RIS - 0x400A C018)

This read-only register contains a 1 for each interrupt condition that is asserted, regardless of whether or not the interrupt is enabled in the SSPnIMSC.

**Table 487: SSPn Raw Interrupt Status register (SSP0RIS - address 0x4008 8018, SSP1RIS - 0x4003 0018, SSP2RIS - 0x400A C018) bit description**

Bit	Symbol	Description	Reset Value
0	RORRIS	This bit is 1 if another frame was completely received while the Rx FIFO was full. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs.	0
1	RTRIS	This bit is 1 if the Rx FIFO is not empty, and has not been read for a period of 32 bit times.	0
2	RXRIS	This bit is 1 if the Rx FIFO is at least half full.	0
3	TXRIS	This bit is 1 if the Tx FIFO is at least half empty.	1
31:4	-	Reserved. The value read from a reserved bit is not defined.	NA

### 21.6.8 SSPn Masked Interrupt Status Register (SSP0MIS - 0x4008 801C, SSP1MIS - 0x4003 001C, SSP2MIS - 0x400A C01C)

This read-only register contains a 1 for each interrupt condition that is asserted and enabled in the SSPnIMSC. When an SSP interrupt occurs, the interrupt service routine should read this register to determine the cause(s) of the interrupt.

**Table 488: SSPn Masked Interrupt Status register (SSP0MIS - address 0x4008 801C, SSP1MIS - 0x4003 001C, SSP2MIS - 0x400A C01C) bit description**

Bit	Symbol	Description	Reset Value
0	RORMIS	This bit is 1 if another frame was completely received while the Rx FIFO was full, and this interrupt is enabled.	0
1	RTMIS	This bit is 1 if the Rx FIFO is not empty, has not been read for a period of 32 bit times, and this interrupt is enabled.	0
2	RXMIS	This bit is 1 if the Rx FIFO is at least half full, and this interrupt is enabled.	0
3	TXMIS	This bit is 1 if the Tx FIFO is at least half empty, and this interrupt is enabled.	0
31:4	-	Reserved. The value read from a reserved bit is not defined.	NA

### 21.6.9 SSPn Interrupt Clear Register (SSP0ICR - 0x4008 8020, SSP1ICR - 0x4003 0020, SSP2ICR - 0x400A C020)

Software can write one or more one(s) to this write-only register, to clear the corresponding interrupt condition(s) in the SSP controller. Note that the other two interrupt conditions can be cleared by writing or reading the appropriate FIFO, or disabled by clearing the corresponding bit in SSPnIMSC.

**Table 489: SSPn interrupt Clear Register (SSP0ICR - address 0x4008 8020, SSP1ICR - 0x4003 0020, SSP2ICR - 0x400A C020) bit description**

Bit	Symbol	Description	Reset Value
0	RORIC	Writing a 1 to this bit clears the "frame was received when RxFIFO was full" interrupt.	NA
1	RTIC	Writing a 1 to this bit clears the "Rx FIFO was not empty and has not been read for a period of 32 bit times" interrupt.	NA
31:2	-	Reserved. Read value is undefined, only zero should be written.	NA

### 21.6.10 SSPn DMA Control Register (SSP0DMACR - 0x4008 8024, SSP1DMACR - 0x4003 0024, SSP2DMACR - 0x400A C024)

The SSPnDMACR register is the DMA control register. It is a read/write register.

**Table 490: SSPn DMA Control Register (SSP0DMACR - address 0x4008 8024, SSP1DMACR - 0x4003 0024, SSP2DMACR - 0x400A C024) bit description**

Bit	Symbol	Description	Reset Value
0	Receive DMA Enable (RXDMAE)	When this bit is set to one 1, DMA for the receive FIFO is enabled, otherwise receive DMA is disabled.	0
1	Transmit DMA Enable (TXDMAE)	When this bit is set to one 1, DMA for the transmit FIFO is enabled, otherwise transmit DMA is disabled	0
31:2	-	Reserved. Read value is undefined, only zero should be written.	NA

### 22.1 Basic configuration

The I<sup>2</sup>C0/1/2 interfaces are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCI2C0/1/2.

**Remark:** On reset, all I<sup>2</sup>C interfaces are enabled (PCI2C0/1/2 = 1).

2. Peripheral clock: The I<sup>2</sup>C interfaces operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select I<sup>2</sup>C0, I<sup>2</sup>C1, or I<sup>2</sup>C2 pins and pin modes using the relevant IOCON registers (See [Section 8.4.1](#)).

**Remark:** The pins P0[27] and P0[28] are specialized open-drain I<sup>2</sup>C pins that support fully compliant fast mode (400 kHz) and standard mode (100 kHz) I<sup>2</sup>C. These pins have no on-chip pull-up devices at all and must always be pulled up externally when they are outputs (per the I<sup>2</sup>C bus specification). Pins P5[2] and P5[3] are similar, but in addition, also support Fast Mode Plus (1 MHz) I<sup>2</sup>C. Both sets of pins have somewhat different configuration options than most port pins, see [Section 8.4.1](#) for details. For all other pins that can be used for I<sup>2</sup>C communication, see the Remark below.

**Remark:** I<sup>2</sup>C pins that do not use specialized I<sup>2</sup>C pads (as identified in [Table 65](#)) can be configured to an open-drain mode via the relevant IOCON registers, and can be used with fast mode (400 kHz) and standard mode (100 kHz) I<sup>2</sup>C. These pins do not include an analog filter to suppress line glitches, but a similar function is performed by the digital filter in the I<sup>2</sup>C block itself. These pins should be configured as: no pull-up, no pull-down, open drain mode.

4. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
5. Initialization: see [Section 22.9.8.1](#) and [Section 22.10.1](#).

## 22.2 Features

- Supports 1 MHz Fast Mode Plus (some pinouts of I<sup>2</sup>C0 only), 400 kHz fast mode, and 100 kHz standard mode.
- Standard I<sup>2</sup>C compliant bus interfaces may be configured as Master, Slave, or Master/Slave.
- Arbitration is handled between simultaneously transmitting masters without corruption of serial data on the bus.
- Programmable clock allows adjustment of I<sup>2</sup>C transfer rates.
- Data transfer is bidirectional between masters and slaves.
- Serial clock synchronization allows devices with different bit rates to communicate via one serial bus.
- Serial clock synchronization is used as a handshake mechanism to suspend and resume serial transfer.
- Optional recognition of up to 4 distinct slave addresses.
- Monitor mode allows observing all I<sup>2</sup>C-bus traffic, regardless of slave address, without affecting the actual I<sup>2</sup>C-bus traffic.

## 22.3 Applications

Interfaces to external I<sup>2</sup>C standard parts, such as serial RAMs, LCDs, tone generators, other microcontrollers, etc. Can also be used as a diagnostic/test bus.

## 22.4 Description

A typical I<sup>2</sup>C-bus configuration is shown in [Figure 106](#). Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I<sup>2</sup>C-bus:

- Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte, unless the slave device is unable to accept more data.
- Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. Next follows the data bytes transmitted by the slave to the master. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a “not acknowledge” is returned. The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the I<sup>2</sup>C-bus will not be released.

The LPC178x/177x I<sup>2</sup>C interfaces are byte oriented and have four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

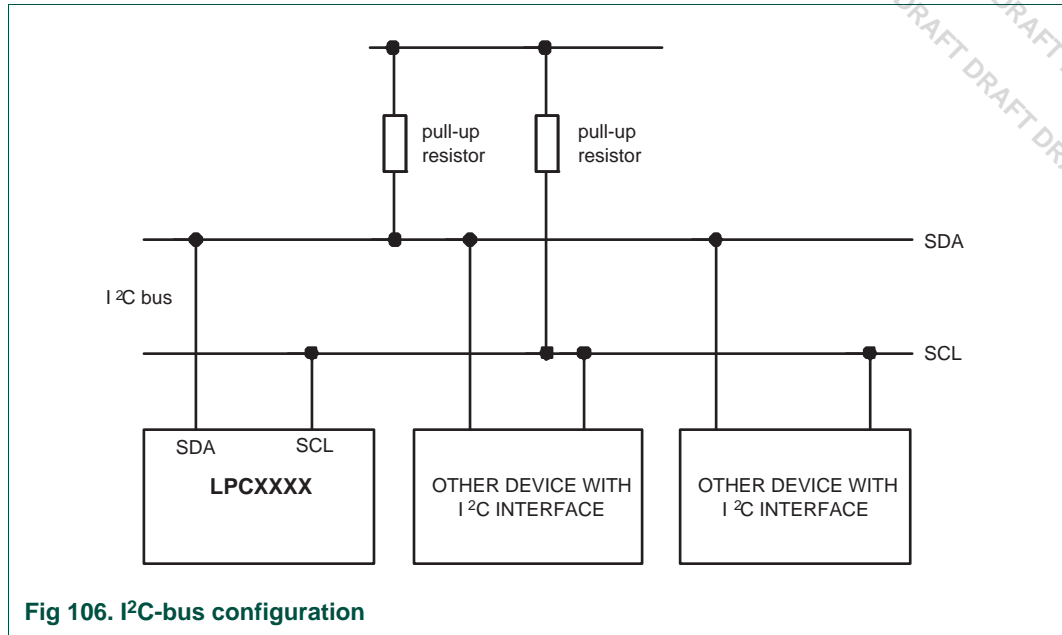


Fig 106. I<sup>2</sup>C-bus configuration

### 22.4.1 I<sup>2</sup>C FAST Mode Plus

Fast Mode Plus is a 1 Mbit/sec transfer rate to communicate with the I<sup>2</sup>C products which the NXP Semiconductors is now providing.

In order to use Fast Mode Plus, the I<sup>2</sup>C0 pins must be configured, then rates above 400 kHz and up to 1 Mhz may be selected, see [Table 505](#). To configure the pins for Fast Mode Plus, the I2CMODE bits in the IOCON\_P5\_02 and IOCON\_P5\_03 registers must be set to binary 10, see [Section 8.4.1](#).

## 22.5 Pin description

**Table 491. I<sup>2</sup>C Pin Description**

Pin	Type	Description
I2C0_SDA	Input/Output	I <sup>2</sup> C0 Serial Data
I2C0_SCL	Input/Output	I <sup>2</sup> C0 Serial Clock
I2C1_SDA	Input/Output	I <sup>2</sup> C1 Serial Data
I2C1_SCL	Input/Output	I <sup>2</sup> C1 Serial Clock
I2C2_SDA	Input/Output	I <sup>2</sup> C2 Serial Data
I2C2_SCL	Input/Output	I <sup>2</sup> C2 Serial Clock

The internal logic of the 3 I<sup>2</sup>C interfaces is identical. These interfaces can be brought out to device pins in several ways, some of which have different pin I/O characteristics. I2C0 on pins P0[27] and P0[28] use specialized I<sup>2</sup>C pads that support fully spec compliant fast mode and standard mode I<sup>2</sup>C. I2C0 on pins P5[2] and P5[3] also use specialized I<sup>2</sup>C pads. These pads support Fast Mode Plus in addition to the previously mentioned modes.

Any of the I<sup>2</sup>C interfaces brought out to pins other than those just mentioned use standard I/O pins. These pins also support I<sup>2</sup>C operation in fast mode and standard mode. The primary difference is that these pins do not include an analog spike suppression filter that exists on the specialized I<sup>2</sup>C pads. The I<sup>2</sup>C interfaces all include a digital filter that can serve the same purpose.

## 22.6 I<sup>2</sup>C operating modes

In a given application, the I<sup>2</sup>C block may operate as a master, a slave, or both. In the slave mode, the I<sup>2</sup>C hardware looks for any one of its four slave addresses and the General Call address. If one of these addresses is detected, an interrupt is requested. If the processor wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave operation is not interrupted. If bus arbitration is lost in the master mode, the I<sup>2</sup>C block switches to the slave mode immediately and can detect any of its own configured slave addresses in the same serial transfer.

### 22.6.1 Master Transmitter mode

In this mode data is transmitted from master to slave. Before the master transmitter mode can be entered, the I2CONSET register must be initialized as shown in [Table 492](#). I2EN must be set to 1 to enable the I<sup>2</sup>C function. If the AA bit is 0, the I<sup>2</sup>C interface will not acknowledge any address when another device is master of the bus, so it can not enter slave mode. The STA, STO and SI bits must be 0. The SI bit is cleared by writing 1 to the SIC bit in the I2CONCLR register. The STA bit should be cleared after writing the slave address.

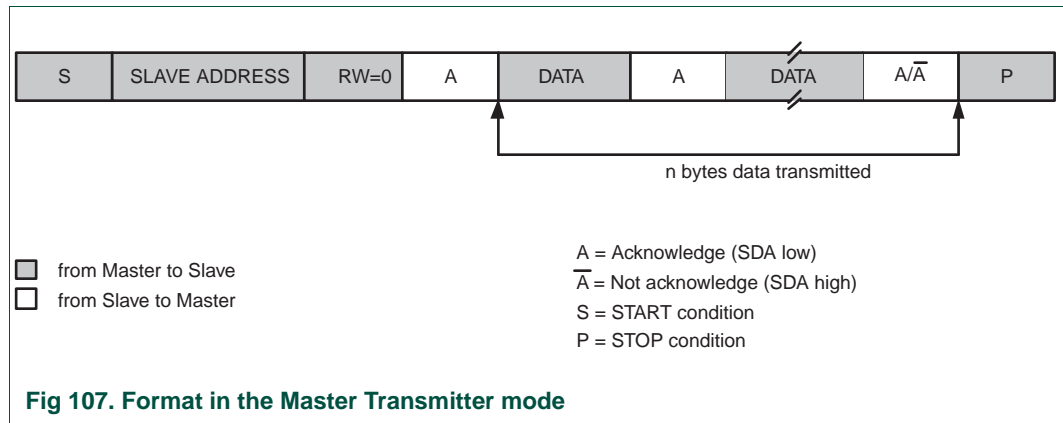
**Table 492. I2C0CONSET and I2C1CONSET used to configure Master mode**

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	0	-	-

The first byte transmitted contains the slave address of the receiving device (7 bits) and the data direction bit. In this mode the data direction bit (R/W) should be 0 which means Write. The first byte transmitted contains the slave address and Write bit. Data is transmitted 8 bits at a time. After each byte is transmitted, an acknowledge bit is received. START and STOP conditions are output to indicate the beginning and the end of a serial transfer.

The I<sup>2</sup>C interface will enter master transmitter mode when software sets the STA bit. The I<sup>2</sup>C logic will send the START condition as soon as the bus is free. After the START condition is transmitted, the SI bit is set, and the status code in the I2STAT register is 0x08. This status code is used to vector to a state service routine which will load the slave address and Write bit to the I2DAT register, and then clear the SI bit. SI is cleared by writing a 1 to the SIC bit in the I2CONCLR register.

When the slave address and R/W bit have been transmitted and an acknowledgment bit has been received, the SI bit is set again, and the possible status codes now are 0x18, 0x20, or 0x38 for the master mode, or 0x68, 0x78, or 0xB0 if the slave mode was enabled (by setting AA to 1). The appropriate actions to be taken for each of these status codes are shown in [Table 509](#) to [Table 512](#).



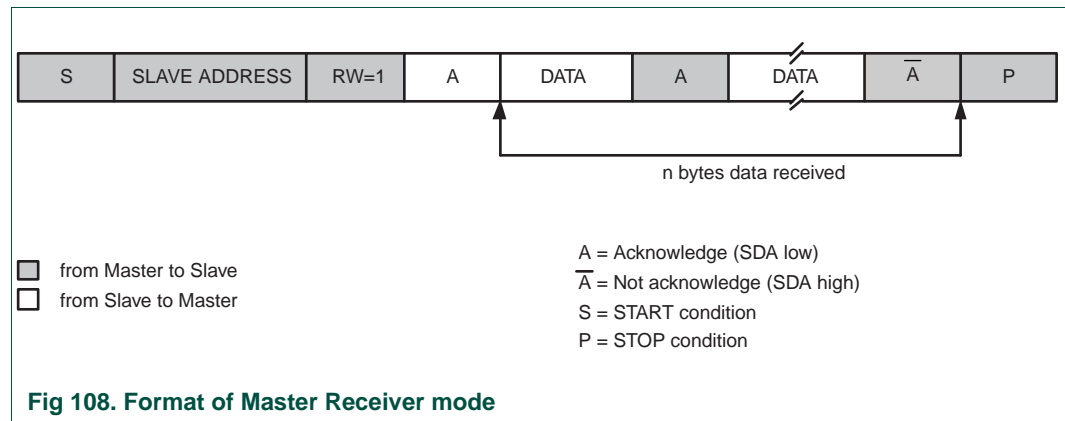
### 22.6.2 Master Receiver mode

In the master receiver mode, data is received from a slave transmitter. The transfer is initiated in the same way as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load the slave address and the data direction bit to the I<sup>2</sup>C Data register (I2DAT), and then clear the SI bit. In this case, the data direction bit (R/W) should be 1 to indicate a read.

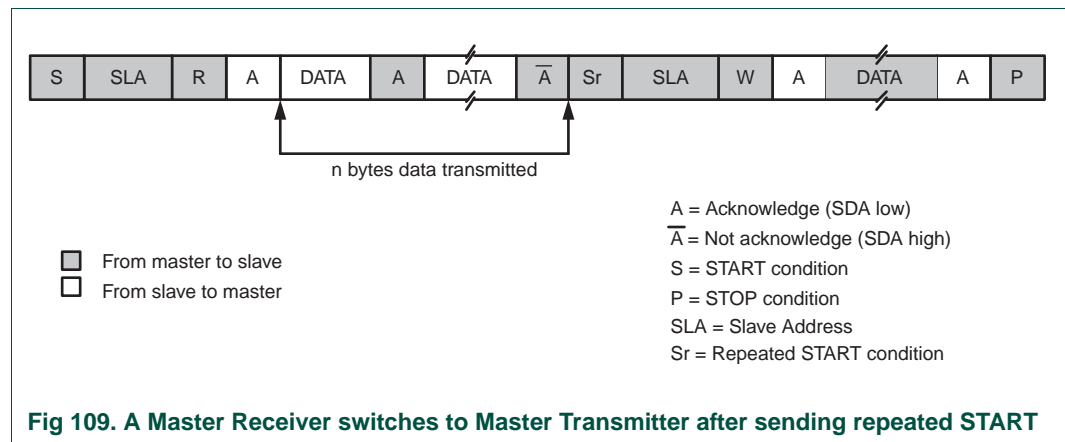
When the slave address and data direction bit have been transmitted and an acknowledge bit has been received, the SI bit is set, and the Status Register will show the status code. For master mode, the possible status codes are 0x40, 0x48, or 0x38. For slave mode, the possible status codes are 0x68, 0x78, or 0xB0. For details, refer to [Table 510](#).

When the LPC178x/177x needs to acknowledge a received byte, the AA bit needs to be set accordingly prior to clearing the SI bit and initiating the byte read. When the LPC178x/177x needs to not acknowledge a received byte, the AA bit needs to be cleared prior to clearing the SI bit and initiating the byte read.

Note that the last received byte is always followed by a "Not Acknowledge" from the LPC178x/177x so that the master can signal the slave that the reading sequence is finished and that it needs to issue a STOP or repeated START Command. Once the "Not Acknowledge" has been sent and the SI bit is set, the LPC178x/177x can send either a STOP (STO bit is set) or a repeated START (STA bit is set). Then the SI bit is cleared to initiate the requested operation.



After a repeated START condition, I<sup>2</sup>C may switch to the master transmitter mode.



### 22.6.3 Slave Receiver mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, write any of the Slave Address registers (I2ADR0-3) and Slave Mask registers (I2MASK0-3) and write the I<sup>2</sup>C Control Set register (I2CONSET) as shown in [Table 493](#).

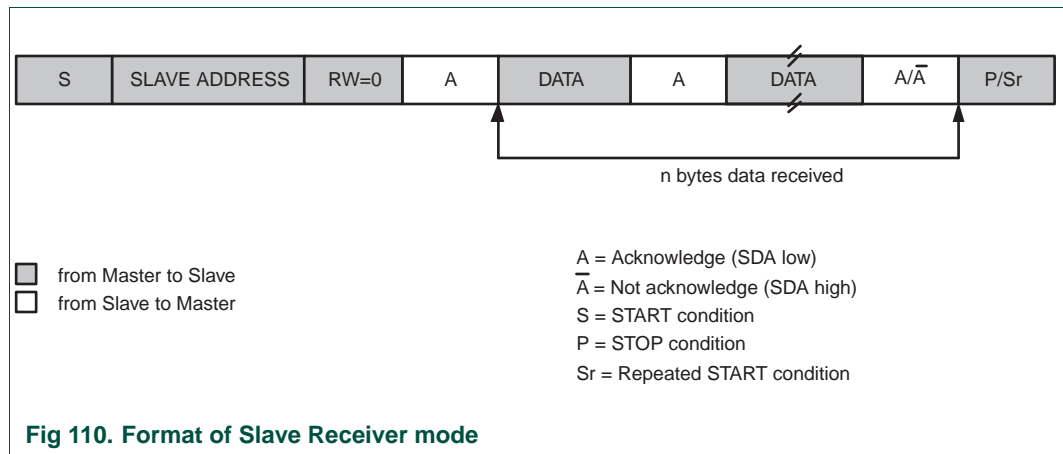


**Table 493. I2C0CONSET and I2C1CONSET used to configure Slave mode**

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	1	-	-

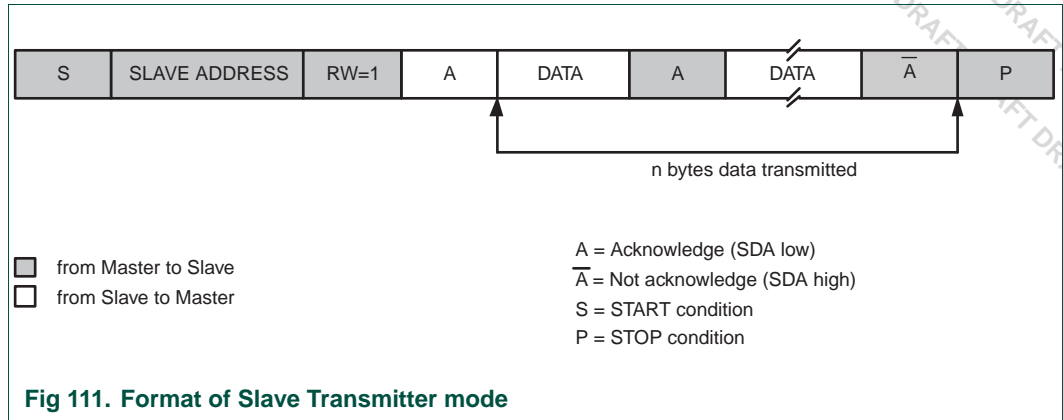
I2EN must be set to 1 to enable the I<sup>2</sup>C function. AA bit must be set to 1 to acknowledge any of its own slave addresses or the General Call address. The STA, STO and SI bits are set to 0.

After I2ADR and I2CONSET are initialized, the I<sup>2</sup>C interface waits until it is addressed by its any of its own slave addresses or General Call address followed by the data direction bit. If the direction bit is 0 (W), it enters slave receiver mode. If the direction bit is 1 (R), it enters slave transmitter mode. After the address and direction bit have been received, the SI bit is set and a valid status code can be read from the Status register (I2STAT). Refer to [Table 511](#) for the status codes and actions.



### 22.6.4 Slave Transmitter mode

The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will be 1, indicating a read operation. Serial data is transmitted via SDA while the serial clock is input through SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. In a given application, I<sup>2</sup>C may operate as a master and as a slave. In the slave mode, the I<sup>2</sup>C hardware looks for any of its own slave addresses and the General Call address. If one of these addresses is detected, an interrupt is requested. When the microcontrollers wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave action is not interrupted. If bus arbitration is lost in the master mode, the I<sup>2</sup>C interface switches to the slave mode immediately and can detect any of its own slave addresses in the same serial transfer.



## 22.7 I<sup>2</sup>C implementation and operation

[Figure 112](#) shows how the on-chip I<sup>2</sup>C-bus interface is implemented, and the following text describes the individual blocks.

### 22.7.1 Input filters and output stages

Input signals are synchronized with the internal clock, and spikes shorter than three clocks are filtered out.

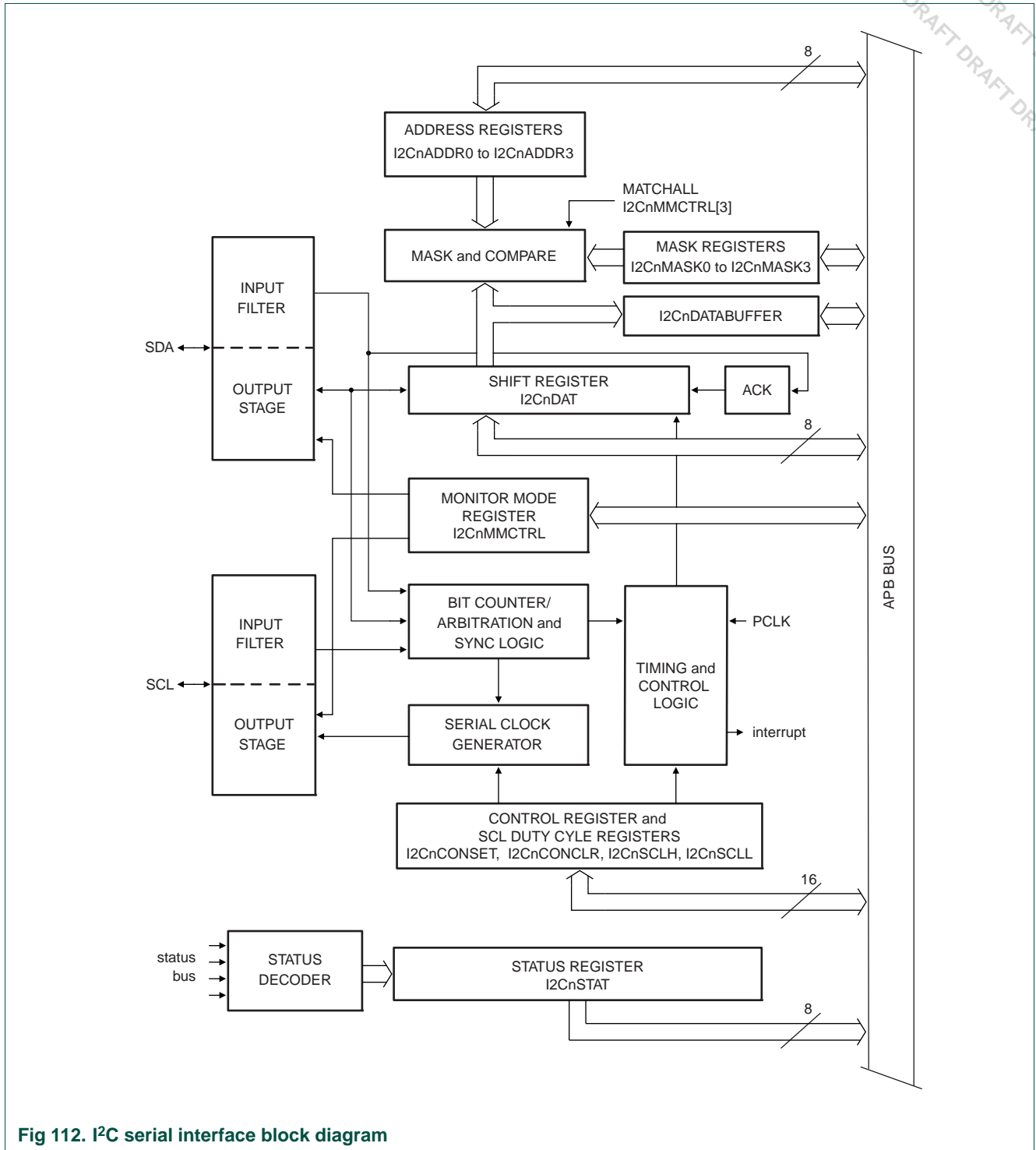


Fig 112. I2C serial interface block diagram

### 22.7.2 Address Registers, I2ADR0 to I2ADR3

These registers may be loaded with the 7-bit slave address (7 most significant bits) to which the I<sup>2</sup>C block will respond when programmed as a slave transmitter or receiver. The LSB (GC) is used to enable General Call address (0x00) recognition. When multiple slave addresses are enabled, the actual address received may be read from the I2DAT register at the state where the “own slave address” has just been received.

**Remark:** in the remainder of this chapter, when the phrase “own slave address” is used, it refers to any of the four configured slave addresses after address masking.

### 22.7.3 Address mask registers, I2MASK0 to I2MASK3

The four mask registers each contain seven active bits (7:1). Any bit in these registers which is set to ‘1’ will cause an automatic compare on the corresponding bit of the received address when it is compared to the I2ADRn register associated with that mask register. In other words, bits in an I2ADRn register which are masked are not taken into account in determining an address match.

When an address-match interrupt occurs, the processor will have to read the data register (I2DAT) to determine which received address actually caused the match.

### 22.7.4 Comparator

The comparator compares the received 7-bit slave address with any of the four configured slave addresses in I2ADR0 through I2ADR3 after masking. It also compares the first received 8-bit byte with the General Call address (0x00). If an a match is found, the appropriate status bits are set and an interrupt is requested.

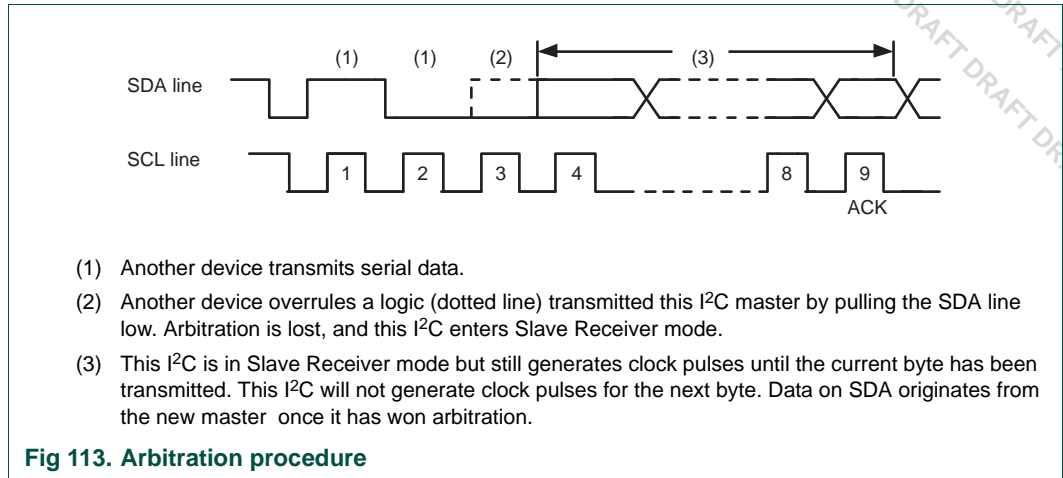
### 22.7.5 Shift register, I2DAT

This 8-bit register contains a byte of serial data to be transmitted or a byte which has just been received. Data in I2DAT is always shifted from right to left; the first bit to be transmitted is the MSB (bit 7) and, after a byte has been received, the first bit of received data is located at the MSB of I2DAT. While data is being shifted out, data on the bus is simultaneously being shifted in; I2DAT always contains the last byte present on the bus. Thus, in the event of lost arbitration, the transition from master transmitter to slave receiver is made with the correct data in I2DAT.

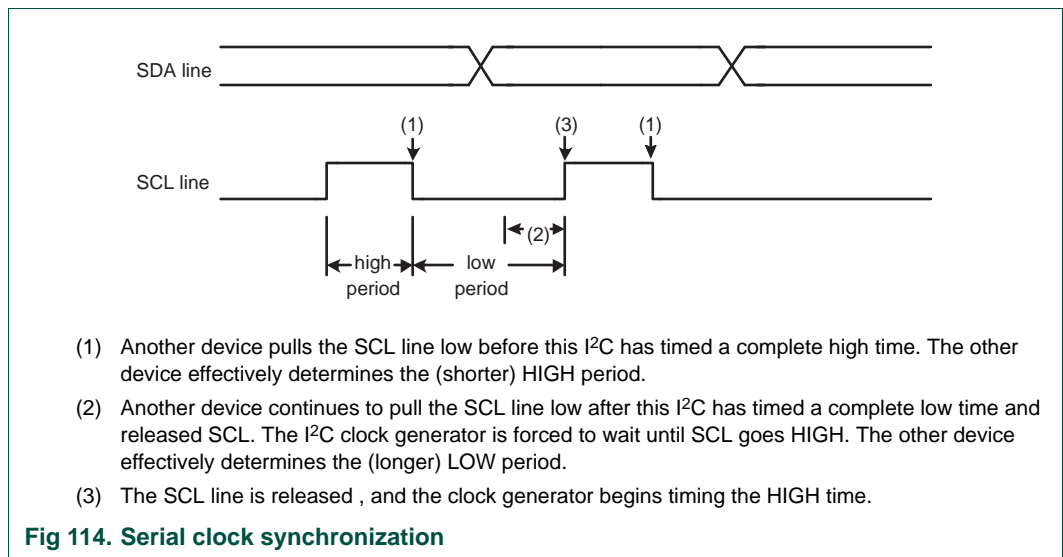
### 22.7.6 Arbitration and synchronization logic

In the master transmitter mode, the arbitration logic checks that every transmitted logic 1 actually appears as a logic 1 on the I<sup>2</sup>C-bus. If another device on the bus overrules a logic 1 and pulls the SDA line low, arbitration is lost, and the I<sup>2</sup>C block immediately changes from master transmitter to slave receiver. The I<sup>2</sup>C block will continue to output clock pulses (on SCL) until transmission of the current serial byte is complete.

Arbitration may also be lost in the master receiver mode. Loss of arbitration in this mode can only occur while the I<sup>2</sup>C block is returning a “not acknowledge: (logic 1) to the bus. Arbitration is lost when another device on the bus pulls this signal low. Since this can occur only at the end of a serial byte, the I<sup>2</sup>C block generates no further clock pulses. [Figure 113](#) shows the arbitration procedure.



The synchronization logic will synchronize the serial clock generator with the clock pulses on the SCL line from another device. If two or more master devices generate clock pulses, the “mark” duration is determined by the device that generates the shortest “marks,” and the “space” duration is determined by the device that generates the longest “spaces”. [Figure 114](#) shows the synchronization procedure.



A slave may stretch the space duration to slow down the bus master. The space duration may also be stretched for handshaking purposes. This can be done after each bit or after a complete byte transfer. the I<sup>2</sup>C block will stretch the SCL space duration after a byte has been transmitted or received and the acknowledge bit has been transferred. The serial interrupt flag (SI) is set, and the stretching continues until the serial interrupt flag is cleared.

### 22.7.7 Serial clock generator

This programmable clock pulse generator provides the SCL clock pulses when the I<sup>2</sup>C block is in the master transmitter or master receiver mode. It is switched off when the I<sup>2</sup>C block is in a slave mode. The I<sup>2</sup>C output clock frequency and duty cycle is programmable

via the I<sup>2</sup>C Clock Control Registers. See the description of the I2CSCLL and I2CSCLH registers for details. The output clock pulses have a duty cycle as programmed unless the bus is synchronizing with other SCL clock sources as described above.

### 22.7.8 Timing and control

The timing and control logic generates the timing and control signals for serial byte handling. This logic block provides the shift pulses for I2DAT, enables the comparator, generates and detects START and STOP conditions, receives and transmits acknowledge bits, controls the master and slave modes, contains interrupt request logic, and monitors the I<sup>2</sup>C-bus status.

### 22.7.9 Control register, I2CONSET and I2CONCLR

The I<sup>2</sup>C control register contains bits used to control the following I<sup>2</sup>C block functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.

The contents of the I<sup>2</sup>C control register may be read as I2CONSET. Writing to I2CONSET will set bits in the I<sup>2</sup>C control register that correspond to ones in the value written. Conversely, writing to I2CONCLR will clear bits in the I<sup>2</sup>C control register that correspond to ones in the value written.

### 22.7.10 Status decoder and status register

The status decoder takes all of the internal status bits and compresses them into a 5-bit code. This code is unique for each I<sup>2</sup>C-bus status. The 5-bit code may be used to generate vector addresses for fast processing of the various service routines. Each service routine processes a particular bus status. There are 26 possible bus states if all four modes of the I<sup>2</sup>C block are used. The 5-bit status code is latched into the five most significant bits of the status register when the serial interrupt flag is set (by hardware) and remains stable until the interrupt flag is cleared by software. The three least significant bits of the status register are always zero. If the status code is used as a vector to service routines, then the routines are displaced by eight address locations. Eight bytes of code is sufficient for most of the service routines (see the software example in this section).

## 22.8 Register description

Each I<sup>2</sup>C interface contains 16 registers as shown in [Table 494](#) below.

**Remark:** In the LPC178x/177x, the following registers have been added to support response to multiple addresses in Slave mode and a new Monitor mode: I2ADR1 to 3, I2MASK0 To 3, MMCTRL, and I2DATA\_BUFFER.

**Table 494. I<sup>2</sup>C register map**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	I <sup>2</sup> Cn Name & Address	Table
I2CONSET	<b>I2C Control Set Register.</b> When a one is written to a bit of this register, the corresponding bit in the I <sup>2</sup> C control register is set. Writing a zero has no effect on the corresponding bit in the I <sup>2</sup> C control register.	R/W	0x00	I2C0CONSET - 0x4001 C000 I2C1CONSET - 0x4005 C000 I2C2CONSET - 0x400A 0000	<a href="#">495</a>
I2STAT	<b>I2C Status Register.</b> During I <sup>2</sup> C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8	I2C0STAT - 0x4001 C004 I2C1STAT - 0x4005 C004 I2C2STAT - 0x400A 0004	<a href="#">497</a>
I2DAT	<b>I2C Data Register.</b> During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register.	R/W	0x00	I2C0DAT - 0x4001 C008 I2C1DAT - 0x4005 C008 I2C2DAT - 0x400A 0008	<a href="#">498</a>
I2ADR0	<b>I2C Slave Address Register 0.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR0 - 0x4001 C00C I2C1ADR0 - 0x4005 C00C I2C2ADR0 - 0x400A 000C	<a href="#">501</a>
I2SCLH	<b>SCH Duty Cycle Register High Half Word.</b> Determines the high time of the I <sup>2</sup> C clock.	R/W	0x04	I2C0SCLH - 0x4001 C010 I2C1SCLH - 0x4005 C010 I2C2SCLH - 0x400A 0010	<a href="#">503</a>
I2SCLL	<b>SCL Duty Cycle Register Low Half Word.</b> Determines the low time of the I <sup>2</sup> C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I <sup>2</sup> C master and certain times used in slave mode.	R/W	0x04	I2C0SCLL - 0x4001 C014 I2C1SCLL - 0x4005 C014 I2C2SCLL - 0x400A 0014	<a href="#">504</a>
I2CONCLR	<b>I2C Control Clear Register.</b> When a one is written to a bit of this register, the corresponding bit in the I <sup>2</sup> C control register is cleared. Writing a zero has no effect on the corresponding bit in the I <sup>2</sup> C control register.	WO	NA	I2C0CONCLR - 0x4001 C018 I2C1CONCLR - 0x4005 C018 I2C2CONCLR - 0x400A 0018	<a href="#">496</a>
MMCTRL	<b>Monitor mode control register.</b>	R/W	0x00	I2C0MMCTRL - 0x4001 C01C I2C1MMCTRL - 0x4005 C01C I2C2MMCTRL - 0x400A 001C	<a href="#">499</a>
I2ADR1	<b>I2C Slave Address Register 1.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR1 - 0x4001 C020 I2C1ADR1 - 0x4005 C020 I2C2ADR1 - 0x400A 0020	<a href="#">501</a>

Table 494. I2C register map

Generic Name	Description	Access	Reset value <sup>[1]</sup>	I2Cn Name & Address	Table
I2ADR2	<b>I2C Slave Address Register 2.</b> Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR2 - 0x4001 C024 I2C1ADR2 - 0x4005 C024 I2C2ADR2 - 0x400A 0024	<a href="#">501</a>
I2ADR3	<b>I2C Slave Address Register 3.</b> Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR3 - 0x4001 C028 I2C1ADR3 - 0x4005 C028 I2C2ADR3 - 0x400A 0028	<a href="#">501</a>
I2DATA_BUFFER	<b>Data buffer register.</b> The contents of the 8 MSBs of the I2DAT shift register will be transferred to the I2DATA_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus.	RO	0x00	I2C0DATA_BUFFER - 0x4001 C02C I2C1DATA_BUFFER - 0x4005 C02C I2C2DATA_BUFFER - 0x400A 002C	<a href="#">500</a>
I2MASK0	<b>I2C Slave address mask register 0.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK0 - 0x4001 C030 I2C1MASK0 - 0x4005 C030 I2C2MASK0 - 0x400A 0030	<a href="#">502</a>
I2MASK1	<b>I2C Slave address mask register 1.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK1 - 0x4001 C034 I2C1MASK1 - 0x4005 C034 I2C2MASK1 - 0x400A 0034	<a href="#">502</a>
I2MASK2	<b>I2C Slave address mask register 2.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK2 - 0x4001 C038 I2C1MASK2 - 0x4005 C038 I2C2MASK2 - 0x400A 0038	<a href="#">502</a>
I2MASK3	<b>I2C Slave address mask register 3.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK3 - 0x4001 C03C I2C1MASK3 - 0x4005 C03C I2C2MASK3 - 0x400A 003C	<a href="#">502</a>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

### 22.8.1 I2C Control Set register (I2CONSET: I2C0, I2C0CONSET - 0x4001 C000; I2C1, I2C1CONSET - 0x4005 C000; I2C2, I2C2CONSET - 0x400A 0000)

The I2CONSET registers control setting of bits in the I2CON register that controls operation of the I2C interface. Writing a one to a bit of this register causes the corresponding bit in the I2C control register to be set. Writing a zero has no effect. Reading this register provides the current values of the control and flag bits.



**Table 495. I<sup>2</sup>C Control Set register (I2CONSET: I<sup>2</sup>C0, I2C0CONSET - address 0x4001 C000, I<sup>2</sup>C1, I2C1CONSET - address 0x4005 C000, I<sup>2</sup>C2, I2C2CONSET - address 0x400A 0000) bit description**

Bit	Symbol	Description	Reset value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
2	AA	Assert acknowledge flag.	0
3	SI	I <sup>2</sup> C interrupt flag.	0
4	STO	STOP flag.	0
5	STA	START flag.	0
6	I2EN	I <sup>2</sup> C interface enable.	0
31:7	-	Reserved. Read value is undefined, only zero should be written.	NA

**I2EN** I<sup>2</sup>C Interface Enable. When I2EN is 1, the I<sup>2</sup>C interface is enabled. I2EN can be cleared by writing 1 to the I2ENC bit in the I2CONCLR register. When I2EN is 0, the I<sup>2</sup>C interface is disabled.

When I2EN is “0”, the SDA and SCL input signals are ignored, the I<sup>2</sup>C block is in the “not addressed” slave state, and the STO bit is forced to “0”.

I2EN should not be used to temporarily release the I<sup>2</sup>C-bus since, when I2EN is reset, the I<sup>2</sup>C-bus status is lost. The AA flag should be used instead.

**STA** is the START flag. Setting this bit causes the I<sup>2</sup>C interface to enter master mode and transmit a START condition or transmit a repeated START condition if it is already in master mode.

When STA is 1 and the I<sup>2</sup>C interface is not already in master mode, it enters master mode, checks the bus and generates a START condition if the bus is free. If the bus is not free, it waits for a STOP condition (which will free the bus) and generates a START condition after a delay of a half clock period of the internal clock generator. If the I<sup>2</sup>C interface is already in master mode and data has been transmitted or received, it transmits a repeated START condition. STA may be set at any time, including when the I<sup>2</sup>C interface is in an addressed slave mode.

STA can be cleared by writing 1 to the STAC bit in the I2CONCLR register. When STA is 0, no START condition or repeated START condition will be generated.

If STA and STO are both set, then a STOP condition is transmitted on the I<sup>2</sup>C-bus if the interface is in master mode, and transmits a START condition thereafter. If the I<sup>2</sup>C interface is in slave mode, an internal STOP condition is generated, but is not transmitted on the bus.

**STO** is the STOP flag. Setting this bit causes the I<sup>2</sup>C interface to transmit a STOP condition in master mode, or recover from an error condition in slave mode. When STO is 1 in master mode, a STOP condition is transmitted on the I<sup>2</sup>C-bus. When the bus detects the STOP condition, STO is cleared automatically.

In slave mode, setting this bit can recover from an error condition. In this case, no STOP condition is transmitted to the bus. The hardware behaves as if a STOP condition has been received and it switches to “not addressed” slave receiver mode. The STO flag is cleared by hardware automatically.

**SI** is the I<sup>2</sup>C Interrupt Flag. This bit is set when the I<sup>2</sup>C state changes. However, entering state F8 does not set SI since there is nothing for an interrupt service routine to do in that case.

While SI is set, the low period of the serial clock on the SCL line is stretched, and the serial transfer is suspended. When SCL is HIGH, it is unaffected by the state of the SI flag. SI must be reset by software, by writing a 1 to the SIC bit in I2CONCLR register. The SI bit should be cleared only after the required bit(s) has (have) been set and the value in I2DAT has been loaded or read.

**AA** is the Assert Acknowledge Flag. When set to 1, an acknowledge (low level to SDA) will be returned during the acknowledge clock pulse on the SCL line on the following situations:

1. A matching address defined by registers I2ADR0 through I2ADR3, masked by I2MASK0 though I2MASK3, has been received.
2. The General Call address has been received while the General Call bit (GC) in I2ADR is set.
3. A data byte has been received while the I<sup>2</sup>C is in the master receiver mode.
4. A data byte has been received while the I<sup>2</sup>C is in the addressed slave receiver mode

The AA bit can be cleared by writing 1 to the AAC bit in the I2CONCLR register. When AA is 0, a not acknowledge (HIGH level to SDA) will be returned during the acknowledge clock pulse on the SCL line on the following situations:

1. A data byte has been received while the I<sup>2</sup>C is in the master receiver mode.
2. A data byte has been received while the I<sup>2</sup>C is in the addressed slave receiver mode.

### 22.8.2 I<sup>2</sup>C Control Clear register (I2CONCLR: I<sup>2</sup>C0, I2C0CONCLR - 0x4001 C018; I<sup>2</sup>C1, I2C1CONCLR - 0x4005 C018; I<sup>2</sup>C2, I2C2CONCLR - 0x400A 0018)

The I2CONCLR registers control clearing of bits in the I2CON register that controls operation of the I<sup>2</sup>C interface. Writing a one to a bit of this register causes the corresponding bit in the I<sup>2</sup>C control register to be cleared. Writing a zero has no effect. I2CONCLR is a write-only register. The value of the related bits can be read from the I2CONSET register.

**Table 496. I<sup>2</sup>C Control Clear register (I2CONCLR: I<sup>2</sup>C0, I2C0CONCLR - 0x4001 C018; I<sup>2</sup>C1, I2C1CONCLR - 0x4005 C018; I<sup>2</sup>C2, I2C2CONCLR - 0x400A 0018) bit description**

Bit	Symbol	Description
1:0	-	Reserved. Read value is undefined, only zero should be written.
2	AAC	Assert acknowledge Clear bit.
3	SIC	I <sup>2</sup> C interrupt Clear bit.
4	-	Reserved. Read value is undefined, only zero should be written.
5	STAC	START flag Clear bit.
6	I2ENC	I <sup>2</sup> C interface Disable bit.
31:7	-	Reserved. Read value is undefined, only zero should be written.

**AAC** is the Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the I2CONSET register. Writing 0 has no effect.

**SIC** is the I<sup>2</sup>C Interrupt Clear bit. Writing a 1 to this bit clears the SI bit in the I2CONSET register. Writing 0 has no effect.

**STAC** is the START flag Clear bit. Writing a 1 to this bit clears the STA bit in the I2CONSET register. Writing 0 has no effect.

**I2ENC** is the I<sup>2</sup>C Interface Disable bit. Writing a 1 to this bit clears the I2EN bit in the I2CONSET register. Writing 0 has no effect.

### 22.8.3 I<sup>2</sup>C Status register (I2STAT: I<sup>2</sup>C0, I2C0STAT - 0x4001 C004; I<sup>2</sup>C1, I2C1STAT - 0x4005 C004; I<sup>2</sup>C2, I2C2STAT - 0x400A 0004)

Each I<sup>2</sup>C Status register reflects the condition of the corresponding I<sup>2</sup>C interface. The I<sup>2</sup>C Status register is read-only.

**Table 497. I<sup>2</sup>C Status register (I2STAT: I<sup>2</sup>C0, I2C0STAT - 0x4001 C004; I<sup>2</sup>C1, I2C1STAT - 0x4005 C004; I<sup>2</sup>C2, I2C2STAT - 0x400A 0004) bit description**

Bit	Symbol	Description	Reset value
2:0	-	These bits are unused and are always 0.	0
7:3	Status	These bits give the actual status information about the I <sup>2</sup> C interface.	0x1F
31:8	-	Reserved. The value read from a reserved bit is not defined.	NA

The three least significant bits are always 0. Taken as a byte, the status register contents represent a status code. There are 26 possible status codes. When the status code is 0xF8, there is no relevant information available and the SI bit is not set. All other 25 status codes correspond to defined I<sup>2</sup>C states. When any of these states entered, the SI bit will be set. For a complete list of status codes, refer to tables from [Table 509](#) to [Table 512](#).

### 22.8.4 I<sup>2</sup>C Data register (I2DAT: I<sup>2</sup>C0, I2C0DAT - 0x4001 C008; I<sup>2</sup>C1, I2C1DAT - 0x4005 C008; I<sup>2</sup>C2, I2C2DAT - 0x400A 0008)

This register contains the data to be transmitted or the data just received. The CPU can read and write to this register only while it is not in the process of shifting a byte, when the SI bit is set. Data in I2DAT remains stable as long as the SI bit is set. Data in I2DAT is always shifted from right to left: the first bit to be transmitted is the MSB (bit 7), and after a byte has been received, the first bit of received data is located at the MSB of I2DAT.

**Table 498. I<sup>2</sup>C Data register (I2DAT: I<sup>2</sup>C0, I2C0DAT - 0x4001 C008; I<sup>2</sup>C1, I2C1DAT - 0x4005 C008; I<sup>2</sup>C2, I2C2DAT - 0x400A 0008) bit description**

Bit	Symbol	Description	Reset value
7:0	Data	This register holds data values that have been received or are to be transmitted.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 22.8.5 I<sup>2</sup>C Monitor mode control register (I2MMCTRL: I<sup>2</sup>C0, I2C0MMCTRL - 0x4001 C01C; I<sup>2</sup>C1, I2C1MMCTRL- 0x4005 C01C; I<sup>2</sup>C2, I2C2MMCTRL- 0x400A 001C)

This register controls the Monitor mode which allows the I<sup>2</sup>C module to monitor traffic on the I<sup>2</sup>C-bus without actually participating in traffic or interfering with the I<sup>2</sup>C-bus.

**Table 499. I<sup>2</sup>C Monitor mode control register (I2MMCTRL: I<sup>2</sup>C0, I2C0MMCTRL - 0x4001 C01C; I<sup>2</sup>C1, I2C1MMCTRL- 0x4005 C01C; I<sup>2</sup>C2, I2C2MMCTRL- 0x400A 001C) bit description**

Bit	Symbol	Value	Description	Reset value
0	MM_ENA		Monitor mode enable.	0
		0	Monitor mode disabled.	
		1	The I <sup>2</sup> C module will enter monitor mode. In this mode the SDA output will be put in high impedance mode. This prevents the I <sup>2</sup> C module from outputting data of any kind (including ACK) onto the I <sup>2</sup> C data bus. Depending on the state of the ENA_SCL bit, the output may be also forced high, preventing the module from having control over the I <sup>2</sup> C clock line.	
1	ENA_SCL		SCL output enable.	0
		0	When this bit is cleared to '0', the SCL output will be forced high when the module is in monitor mode. As described above, this will prevent the module from having any control over the I <sup>2</sup> C clock line.	
		1	When this bit is set, the I <sup>2</sup> C module may exercise the same control over the clock line that it would in normal operation. This means that, acting as a slave peripheral, the I <sup>2</sup> C module can "stretch" the clock line (hold it low) until it has had time to respond to an I <sup>2</sup> C interrupt. <sup>[1]</sup>	
2	MATCH_ALL		Select interrupt register match.	0
		0	When this bit is cleared, an interrupt will only be generated when a match occurs to one of the (up-to) four address registers, I2ADR0 through I2ADR3. That is, the module will respond as a normal slave as far as address-recognition is concerned.	
		1	When this bit is set to '1' and the I <sup>2</sup> C is in monitor mode, an interrupt will be generated on ANY address received. This will enable the part to monitor all traffic on the bus.	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] When the ENA\_SCL bit is cleared and the I<sup>2</sup>C no longer has the ability to stretch the clock, interrupt response time becomes important. To give the part more time to respond to an I<sup>2</sup>C interrupt under these conditions, an I2DATA\_BUFFER register is used (Section 22.8.6) to hold received data for a full 9-bit word transmission time.

**Remark:** The ENA\_SCL and MATCH\_ALL bits have no effect if the MM\_ENA is '0' (i.e. if the module is NOT in monitor mode).

### 22.8.5.1 Interrupt in Monitor mode

All interrupts will occur as normal when the module is in monitor mode. This means that the first interrupt will occur when an address-match is detected (any address received if the MATCH\_ALL bit is set, otherwise an address matching one of the four address registers).

Subsequent to an address-match detection, interrupts will be generated after each data byte is received for a slave-write transfer, or after each byte that the module believes it has transmitted for a slave-read transfer. In this second case, the data register will actually contain data transmitted by some other slave on the bus which was actually addressed by the master.

Following all of these interrupts, the processor may read the data register to see what was actually transmitted on the bus.

### 22.8.5.2 Loss of arbitration in Monitor mode

In monitor mode, the I<sup>2</sup>C module will not be able to respond to a request for information by the bus master or issue an ACK. Some other slave on the bus will respond instead.

Software should be aware of the fact that the module is in monitor mode and should not respond to any loss of arbitration state that is detected.

### 22.8.6 I<sup>2</sup>C Data buffer register (I2DATA\_BUFFER: I<sup>2</sup>C0, I2CDATA\_BUFFER - 0x4001 C02C; I<sup>2</sup>C1, I2C1DATA\_BUFFER- 0x4005 C02C; I<sup>2</sup>C2, I2C2DATA\_BUFFER- 0x400A 002C)

In monitor mode, the I<sup>2</sup>C module may lose the ability to stretch the clock if the ENA\_SCL bit is not set. This means that the processor will have a limited amount of time to read the contents of the data received on the bus. If the processor reads the I2DAT shift register, as it ordinarily would, it could have only one bit-time to respond to the interrupt before the received data is overwritten by new data.

To give the processor more time to respond, a new 8-bit, read-only I2DATA\_BUFFER register has been added. The contents of the 8 MSBs of the I2DAT shift register are transferred to the I2DATA\_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus. This means that the processor will have 9 bit transmission times to respond to the interrupt and read the data before it is overwritten.

The processor will still have the ability to read I2DAT directly, as usual, and the behavior of I2DAT will not be altered in any way.

Although the I2DATA\_BUFFER register is primarily intended for use in monitor mode with the ENA\_SCL bit = '0', it is available for reading at any time under any mode of operation.

**Table 500. I<sup>2</sup>C Data buffer register (I2DATA\_BUFFER: I<sup>2</sup>C0, I2CDATA\_BUFFER - 0x4001 C02C; I<sup>2</sup>C1, I2C1DATA\_BUFFER- 0x4005 C02C; I<sup>2</sup>C2, I2C2DATA\_BUFFER- 0x400A 002C) bit description**

Bit	Symbol	Description	Reset value
7:0	Data	This register holds contents of the 8 MSBs of the I2DAT shift register.	0
31:8	-	Reserved. The value read from a reserved bit is not defined.	NA

**22.8.7 I<sup>2</sup>C Slave Address registers (I2ADR0 to 3: I<sup>2</sup>C0, I2C0ADR[0, 1, 2, 3]- 0x4001 C0[0C, 20, 24, 28]; I<sup>2</sup>C1, I2C1ADR[0, 1, 2, 3] - address 0x4005 C0[0C, 20, 24, 28]; I<sup>2</sup>C2, I2C2ADR[0, 1, 2, 3] - address 0x400A 00[0C, 20, 24, 28])**

These registers are readable and writable and are only used when an I<sup>2</sup>C interface is set to slave mode. In master mode, this register has no effect. The LSB of I2ADR is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

If these registers contain 0x00, the I<sup>2</sup>C will not acknowledge any address on the bus. All four registers will be cleared to this disabled state on reset.

**Table 501. I<sup>2</sup>C Slave Address registers (I2ADR0 to 3: I<sup>2</sup>C0, I2C0ADR[0, 1, 2, 3]- 0x4001 C0[0C, 20, 24, 28]; I<sup>2</sup>C1, I2C1ADR[0, 1, 2, 3] - address 0x4005 C0[0C, 20, 24, 28]; I<sup>2</sup>C2, I2C2ADR[0, 1, 2, 3] - address 0x400A 00[0C, 20, 24, 28]) bit description**

Bit	Symbol	Description	Reset value
0	GC	General Call enable bit.	0
7:1	Address	The I <sup>2</sup> C device address for slave mode.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

**22.8.8 I<sup>2</sup>C Mask registers (I2MASK0 to 3: I<sup>2</sup>C0, I2C0MASK[0, 1, 2, 3] - 0x4001 C0[30, 34, 38, 3C]; I<sup>2</sup>C1, I2C1MASK[0, 1, 2, 3] - address 0x4005 C0[30, 34, 38, 3C]; I<sup>2</sup>C2, I2C2MASK[0, 1, 2, 3] - address 0x400A 00[30, 34, 38, 3C])**

The four mask registers each contain seven active bits (7:1). Any bit in these registers which is set to '1' will cause an automatic compare on the corresponding bit of the received address when it is compared to the I2ADRn register associated with that mask register. In other words, bits in an I2ADRn register which are masked are not taken into account in determining an address match.

The mask register has no effect on comparison to the General Call address ("0000000").

When an address-match interrupt occurs, the processor will have to read the data register (I2DAT) to determine which received address actually caused the match.

**Table 502. I<sup>2</sup>C Mask registers (I2MASK0 to 3: I<sup>2</sup>C0, I2C0MASK[0, 1, 2, 3] - 0x4001 C0[30, 34, 38, 3C]; I<sup>2</sup>C1, I2C1MASK[0, 1, 2, 3] - address 0x4005 C0[30, 34, 38, 3C]; I<sup>2</sup>C2, I2C2MASK[0, 1, 2, 3] - address 0x400A 00[30, 34, 38, 3C]) bit description**

Bit	Symbol	Description	Reset value
0	-	Reserved, only zero should be written. This bit always reads back as 0.	0
7:1	MASK	Mask bits.	0
31:8	-	Reserved, only zero should be written. These bits always read back as 0.	0

**22.8.9 I<sup>2</sup>C SCL HIGH duty cycle register (I2SCLH: I<sup>2</sup>C0, I2C0SCLH - 0x4001 C010; I<sup>2</sup>C1, I2C1SCLH - 0x4005 C010; I<sup>2</sup>C2, I2C2SCLH - 0x400A 0010)**

Table 503. I<sup>2</sup>C SCL HIGH Duty Cycle register (I2SCLH: I<sup>2</sup>C0, I2C0SCLH - address 0x4001 C010; I<sup>2</sup>C1, I2C1SCLH - address 0x4005 C010; I<sup>2</sup>C2, I2C2SCLH - 0x400A 0010) bit description

Bit	Symbol	Description	Reset value
15:0	SCLH	Count for SCL HIGH time period selection.	0x0004
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

**22.8.10 I<sup>2</sup>C SCL Low duty cycle register (I2SCLL: I<sup>2</sup>C0 - I2C0SCLL: 0x4001 C014; I<sup>2</sup>C1 - I2C1SCLL: 0x4005 C014; I<sup>2</sup>C2 - I2C2SCLL: 0x400A 0014)**

Table 504. I<sup>2</sup>C SCL Low duty cycle register (I2SCLL: I<sup>2</sup>C0 - I2C0SCLL: 0x4001 C014; I<sup>2</sup>C1 - I2C1SCLL: 0x4005 C014; I<sup>2</sup>C2 - I2C2SCLL: 0x400A 0014) bit description

Bit	Symbol	Description	Reset value
15:0	SCLL	Count for SCL low time period selection.	0x0004
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

**22.8.11 Selecting the appropriate I<sup>2</sup>C data rate and duty cycle**

Software must set values for the registers I2SCLH and I2SCLL to select the appropriate data rate and duty cycle. I2SCLH defines the number of PCLK cycles for the SCL HIGH time, I2SCLL defines the number of PCLK cycles for the SCL low time. The frequency is determined by the following formula (PCLK is the frequency of the peripheral bus APB):

(13)

$$I^2C_{bitfrequency} = \frac{PCLK_{I2C}}{I2CSCLH + I2CSCLL}$$

The values for I2SCLL and I2SCLH must ensure that the data rate is in the appropriate I<sup>2</sup>C data rate range. Each register value must be greater than or equal to 4. [Table 505](#) gives some examples of I<sup>2</sup>C-bus rates based on PCLK frequency and I2SCLL and I2SCLH values.

Table 505. Example I<sup>2</sup>C clock rates

I <sup>2</sup> C Rate	I2SCLL + I2SCLH values at PCLK (MHz)													
	6	8	10	12	16	20	30	40	50	60	70	80	90	100
100 kHz (Standard)	60	80	100	120	160	200	300	400	500	600	700	800	900	1000
400 kHz (Fast Mode)	15	20	25	30	40	50	75	100	125	150	175	200	225	250
1 MHz (Fast Mode Plus)	-	8	10	12	16	20	30	40	50	60	70	80	90	100

I2SCLL and I2SCLH values should not necessarily be the same. Software can set different duty cycles on SCL by setting these two registers. For example, the I<sup>2</sup>C-bus specification defines the SCL low time and high time at different values for Fast Mode and Fast Mode Plus I<sup>2</sup>C.



## 22.9 Details of I<sup>2</sup>C operating modes

The four operating modes are:

- Master Transmitter
- Master Receiver
- Slave Receiver
- Slave Transmitter

Data transfers in each mode of operation are shown in [Figure 115](#), [Figure 116](#), [Figure 117](#), [Figure 118](#), and [Figure 119](#). [Table 506](#) lists abbreviations used in these figures when describing the I<sup>2</sup>C operating modes.

**Table 506. Abbreviations used to describe an I<sup>2</sup>C operation**

Abbreviation	Explanation
S	START condition
SLA	7-bit slave address
R	Read bit (HIGH level at SDA)
W	Write bit (LOW level at SDA)
A	Acknowledge bit (LOW level at SDA)
$\bar{A}$	Not acknowledge bit (HIGH level at SDA)
Data	8-bit data byte
P	STOP condition
Sr	Repeated START condition

In [Figure 115](#) to [Figure 119](#), circles are used to indicate when the serial interrupt flag is set. The numbers in the circles show the status code held in the I2STAT register. At these points, a service routine must be executed to continue or complete the serial transfer. These service routines are not critical since the serial transfer is suspended until the serial interrupt flag is cleared by software.

When a serial interrupt routine is entered, the status code in I2STAT is used to branch to the appropriate service routine. For each status code, the required software action and details of the following serial transfer are given in tables from [Table 509](#) to [Table 513](#).

### 22.9.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see [Figure 115](#)). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

**Table 507. I2CONSET used to initialize Master Transmitter mode**

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	x	-	-

The I<sup>2</sup>C rate must also be configured in the I2SCLL and I2SCLH registers. I2EN must be set to logic 1 to enable the I<sup>2</sup>C block. If the AA bit is reset, the I<sup>2</sup>C block will not acknowledge its own slave address or the General Call address in the event of another device becoming master of the bus. In other words, if AA is reset, the I<sup>2</sup>C interface cannot enter a slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The I<sup>2</sup>C logic will now test the I<sup>2</sup>C-bus and generate a START condition as soon as the bus becomes free. When a START condition is transmitted, the serial interrupt flag (SI) is set, and the status code in the status register (I2STAT) will be 0x08. This status code is used by the interrupt service routine to enter the appropriate state service routine that loads I2DAT with the slave address and the data direction bit (SLA+W). The SI bit in I2CON must then be reset before the serial transfer can continue.

When the slave address and the direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in I2STAT are possible. There are 0x18, 0x20, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = logic 1). The appropriate action to be taken for each of these status codes is detailed in [Table 509](#). After a repeated START condition (state 0x10). The I<sup>2</sup>C block may switch to the master receiver mode by loading I2DAT with SLA+R).

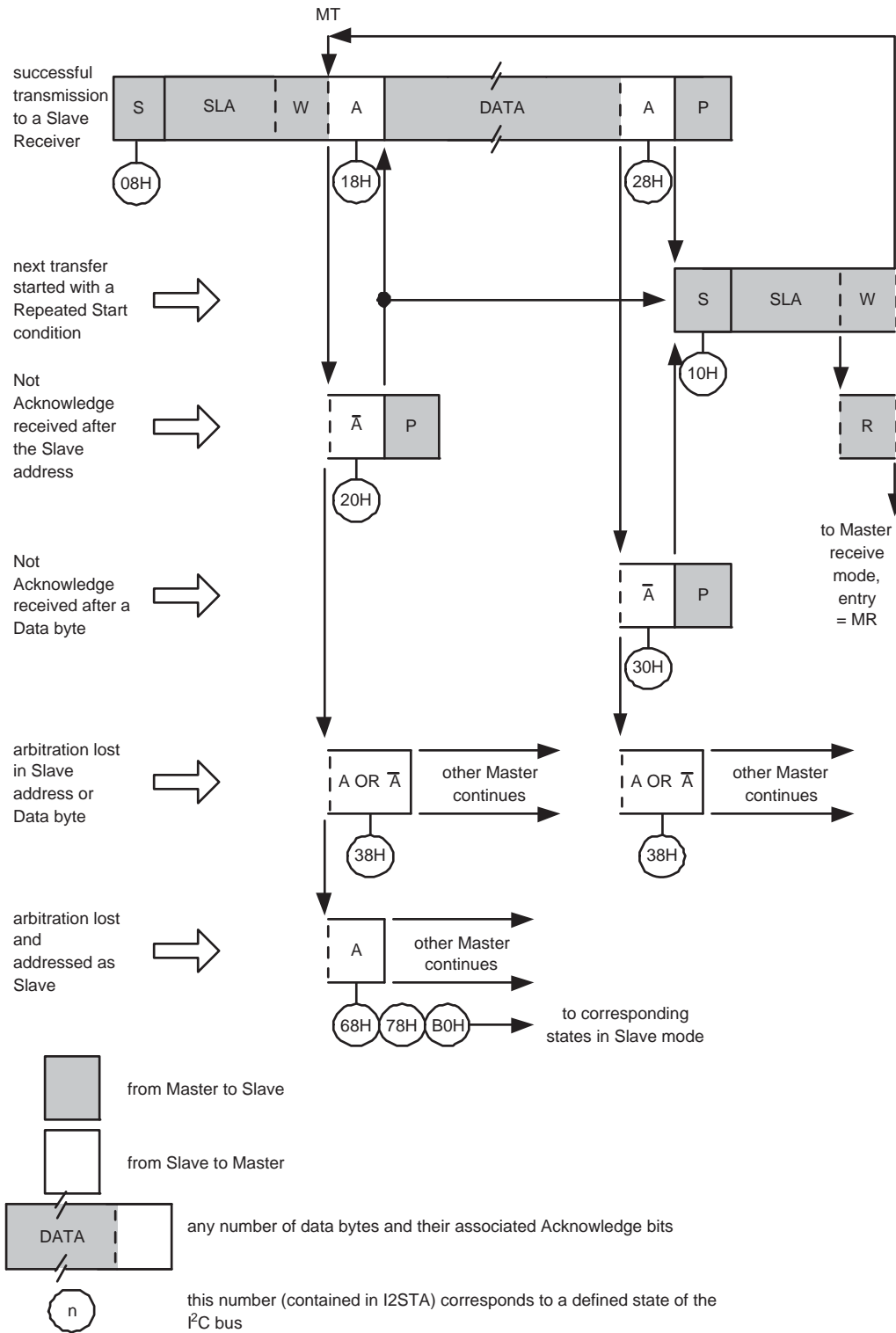
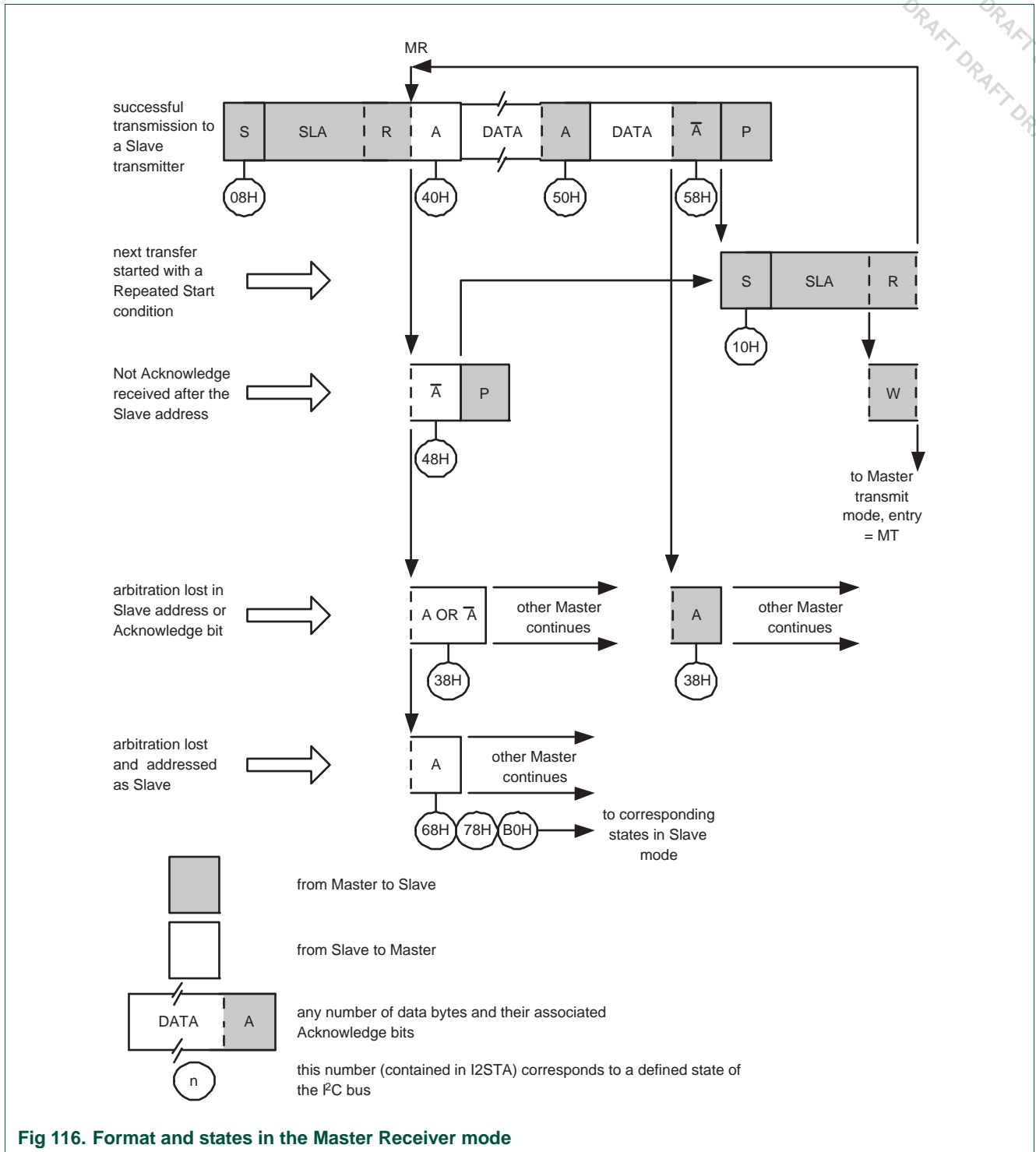


Fig 115. Format and states in the Master Transmitter mode

### 22.9.2 Master Receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see [Figure 116](#)). The transfer is initialized as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load I2DAT with the 7-bit slave address and the data direction bit (SLA+R). The SI bit in I2CON must then be cleared before the serial transfer can continue.

When the slave address and the data direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in I2STAT are possible. These are 0x40, 0x48, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = 1). The appropriate action to be taken for each of these status codes is detailed in [Table 510](#). After a repeated START condition (state 0x10), the I<sup>2</sup>C block may switch to the master transmitter mode by loading I2DAT with SLA+W.



### 22.9.3 Slave Receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see [Figure 117](#)). To initiate the slave receiver mode, I2CON register, the I2ADR registers, and the I2MASK registers must be configured.

The values on the four I2ADR registers combined with the values on the four I2MASK registers determines which address(es) the I<sup>2</sup>C block will respond to when slave functions are enabled. See sections [22.7.2](#), [22.7.3](#), [22.8.7](#), and [22.8.8](#) for details.

**Table 508. I2CONSET used to initialize Slave Receiver mode**

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	1	-	-

The I<sup>2</sup>C-bus rate settings do not affect the I<sup>2</sup>C block in the slave mode. I2EN must be set to logic 1 to enable the I<sup>2</sup>C block. The AA bit must be set to enable the I<sup>2</sup>C block to acknowledge its own slave address or the General Call address. STA, STO, and SI must be reset.

When the I2ADR, I2MASK, and I2CON registers have been initialized, the I<sup>2</sup>C block waits until it is addressed by its own slave address followed by the data direction bit which must be “0” (W) for the I<sup>2</sup>C block to operate in the slave receiver mode. After its own slave address and the W bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from I2STAT. This status code is used to vector to a state service routine. The appropriate action to be taken for each of these status codes is detailed in [Table 511](#). The slave receiver mode may also be entered if arbitration is lost while the I<sup>2</sup>C block is in the master mode (see status 0x68 and 0x78).

If the AA bit is reset during a transfer, the I<sup>2</sup>C block will return a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the I<sup>2</sup>C block does not respond to its own slave address or a General Call address. However, the I<sup>2</sup>C-bus is still monitored and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the I<sup>2</sup>C block from the I<sup>2</sup>C-bus.

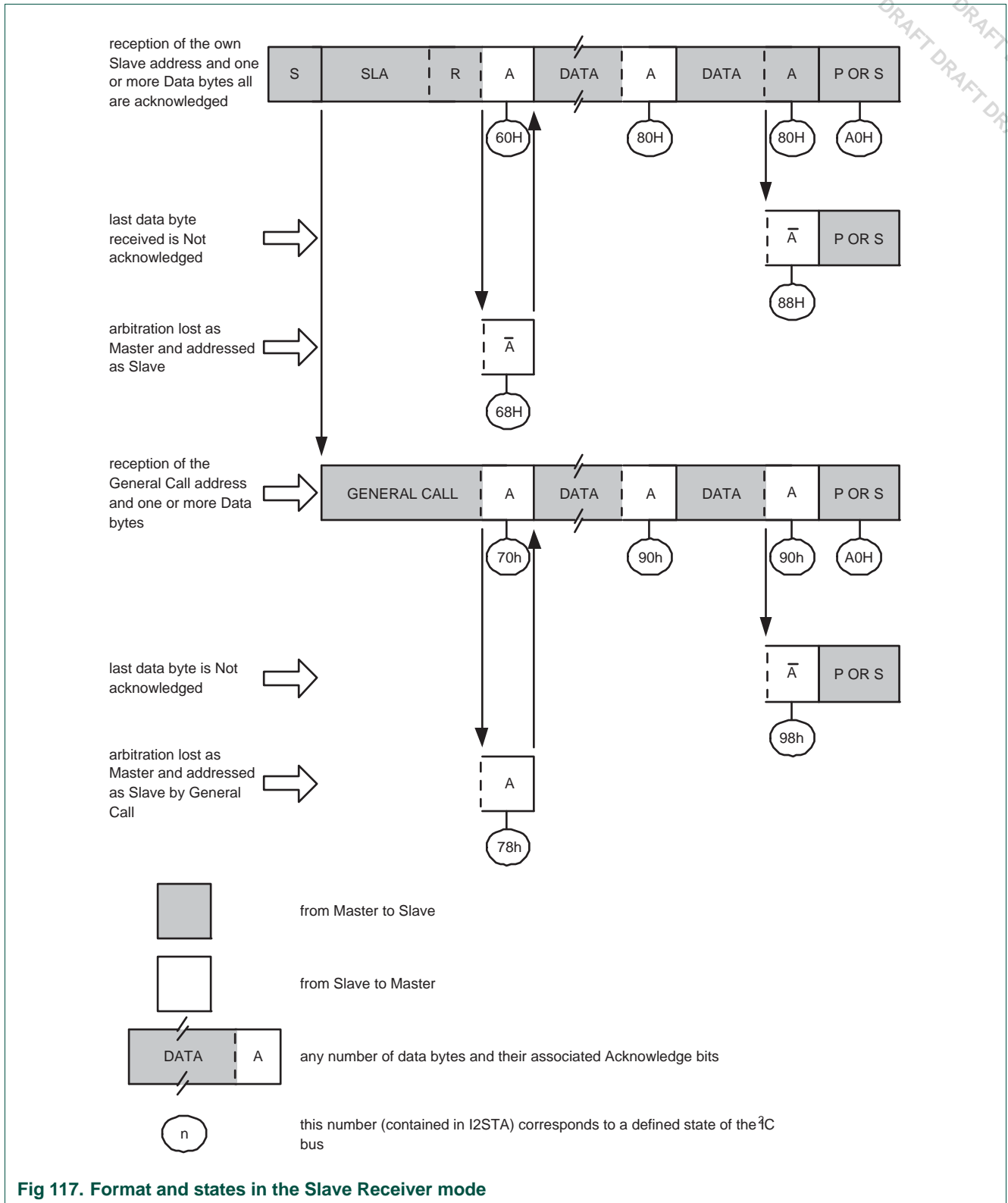


Fig 117. Format and states in the Slave Receiver mode

22.9.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 118). Data transfer is initialized as in the slave receiver mode. When I2ADR and I2CON have been initialized, the I<sup>2</sup>C block waits until it is addressed by its own slave address followed by the data direction bit which must be “1” (R) for the I<sup>2</sup>C block to operate in the slave transmitter mode. After its own slave address and the R bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from I2STAT. This status code is used to vector to a state service routine, and the appropriate action to be taken for each of these status codes is detailed in Table 512. The slave transmitter mode may also be entered if arbitration is lost while the I<sup>2</sup>C block is in the master mode (see state 0xB0).

If the AA bit is reset during a transfer, the I<sup>2</sup>C block will transmit the last byte of the transfer and enter state 0xC0 or 0xC8. The I<sup>2</sup>C block is switched to the not addressed slave mode and will ignore the master receiver if it continues the transfer. Thus the master receiver receives all 1s as serial data. While AA is reset, the I<sup>2</sup>C block does not respond to its own slave address or a General Call address. However, the I<sup>2</sup>C-bus is still monitored, and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the I<sup>2</sup>C block from the I<sup>2</sup>C-bus.

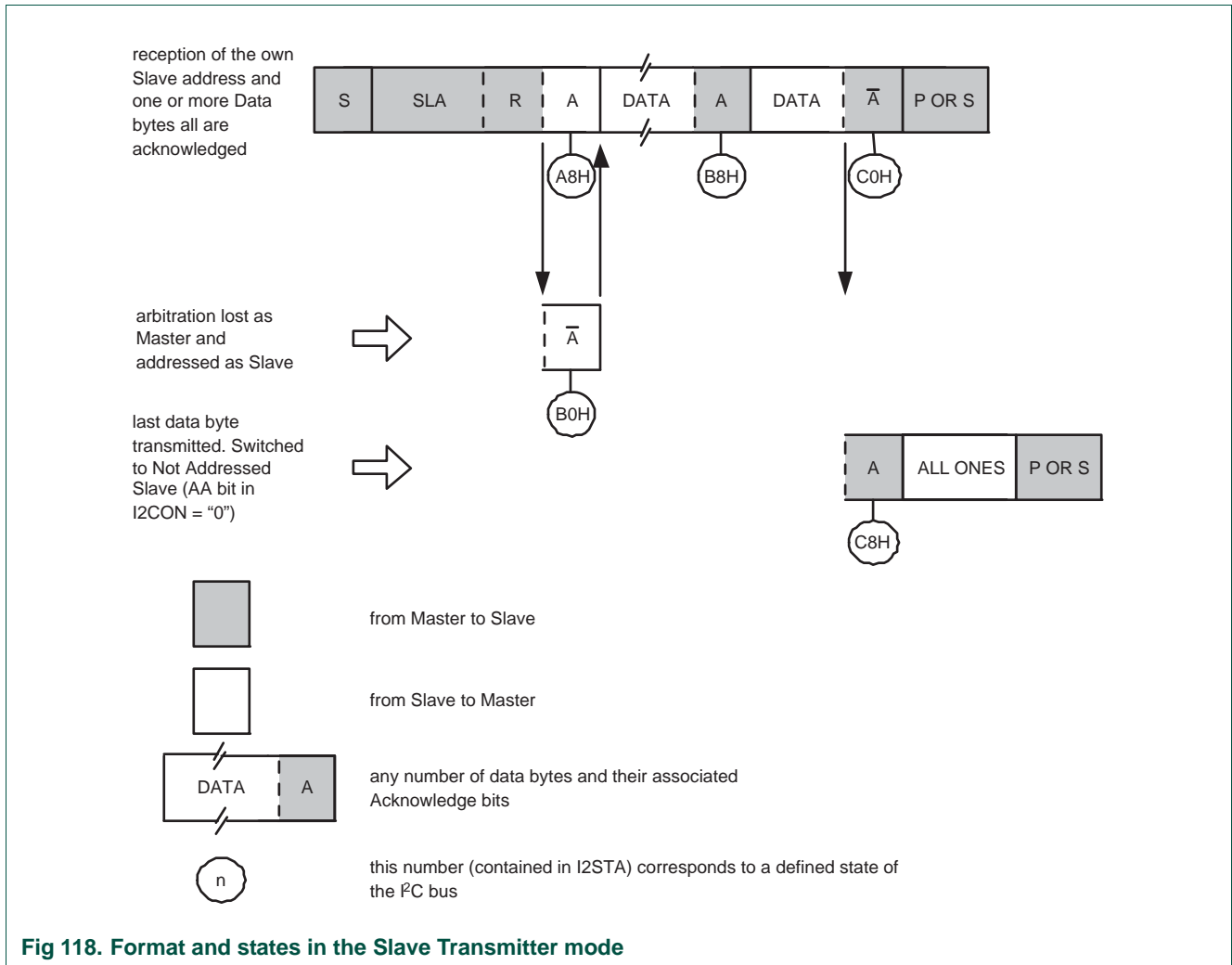


Fig 118. Format and states in the Slave Transmitter mode



22.9.5 Detailed state tables

The following tables show detailed state information for the four I2C operating modes.

Table 509. Master Transmitter mode

I2CSTAT Status Code	Status of the I2C-bus and hardware	Application software response					Next action taken by I2C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0x08	A START condition has been transmitted.	Load SLA+W; clear STA	X	0	0	X	SLA+W will be transmitted; ACK bit will be received.
0x10	A repeated START condition has been transmitted.	Load SLA+W or	X	0	0	X	As above.
		Load SLA+R; Clear STA	X	0	0	X	SLA+W will be transmitted; the I2C block will be switched to MST/REC mode.
0x18	SLA+W has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x20	SLA+W has been transmitted; NOT ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x28	Data byte in I2DAT has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x30	Data byte in I2DAT has been transmitted; NOT ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x38	Arbitration lost in SLA+R/W or Data bytes.	No I2DAT action or	0	0	0	X	I2C-bus will be released; not addressed slave will be entered.
		No I2DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free.

Table 510. Master Receiver mode

I2CSTAT Status Code	Status of the I <sup>2</sup> C-bus and hardware	Application software response To/From I2DAT	To I2CON				Next action taken by I <sup>2</sup> C hardware
			STA	STO	SI	AA	
0x08	A START condition has been transmitted.	Load SLA+R	X	0	0	X	SLA+R will be transmitted; ACK bit will be received.
0x10	A repeated START condition has been transmitted.	Load SLA+R or	X	0	0	X	As above.
		Load SLA+W	X	0	0	X	SLA+W will be transmitted; the I <sup>2</sup> C block will be switched to MST/TRX mode.
0x38	Arbitration lost in NOT ACK bit.	No I2DAT action or	0	0	0	X	I <sup>2</sup> C-bus will be released; the I <sup>2</sup> C block will enter a slave mode.
		No I2DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free.
0x40	SLA+R has been transmitted; ACK has been received.	No I2DAT action or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		No I2DAT action	0	0	0	1	Data byte will be received; ACK bit will be returned.
0x48	SLA+R has been transmitted; NOT ACK has been received.	No I2DAT action or	1	0	0	X	Repeated START condition will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x50	Data byte has been received; ACK has been returned.	Read data byte or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		Read data byte	0	0	0	1	Data byte will be received; ACK bit will be returned.
0x58	Data byte has been received; NOT ACK has been returned.	Read data byte or	1	0	0	X	Repeated START condition will be transmitted.
		Read data byte or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		Read data byte	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.

Table 511. Slave Receiver mode

I2CSTAT Status Code	Status of the I <sup>2</sup> C-bus and hardware	Application software response					Next action taken by I <sup>2</sup> C hardware
		To/From I2DAT	To I2CON	STA	STO	SI	
0x60	Own SLA+W has been received; ACK has been returned.	No I2DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No I2DAT action	X	0	0	1	Data byte will be received and ACK will be returned.
0x68	Arbitration lost in SLA+R/W as master; Own SLA+W has been received, ACK returned.	No I2DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No I2DAT action	X	0	0	1	Data byte will be received and ACK will be returned.
0x70	General Call address (0x00) has been received; ACK has been returned.	No I2DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No I2DAT action	X	0	0	1	Data byte will be received and ACK will be returned.
0x78	Arbitration lost in SLA+R/W as master; General Call address has been received, ACK has been returned.	No I2DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No I2DAT action	X	0	0	1	Data byte will be received and ACK will be returned.
0x80	Previously addressed with own SLA address; DATA has been received; ACK has been returned.	Read data byte or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		Read data byte	X	0	0	1	Data byte will be received and ACK will be returned.
0x88	Previously addressed with own SLA; DATA byte has been received; NOT ACK has been returned.	Read data byte or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address.
		Read data byte or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1.
		Read data byte or	1	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		Read data byte	1	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0x90	Previously addressed with General Call; DATA byte has been received; ACK has been returned.	Read data byte or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		Read data byte	X	0	0	1	Data byte will be received and ACK will be returned.

Table 511. Slave Receiver mode

I2CSTAT Status Code	Status of the I2C-bus and hardware	Application software response To/From I2DAT	To I2CON				Next action taken by I2C hardware
			STA	STO	SI	AA	
0x98	Previously addressed with General Call; DATA byte has been received; NOT ACK has been returned.	Read data byte or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address.
		Read data byte or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1.
		Read data byte or	1	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		Read data byte	1	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave Receiver or Slave Transmitter.	No STDAT action or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address.
		No STDAT action or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1.
		No STDAT action or	1	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No STDAT action	1	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.

Table 512. Slave Transmitter mode

I2CSTAT Status Code	Status of the I <sup>2</sup> C-bus and hardware	Application software response					Next action taken by I <sup>2</sup> C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0xA8	Own SLA+R has been received; ACK has been returned.	Load data byte or	X	0	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	0	1	Data byte will be transmitted; ACK will be received.
0xB0	Arbitration lost in SLA+R/W as master; Own SLA+R has been received, ACK has been returned.	Load data byte or	X	0	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	0	1	Data byte will be transmitted; ACK bit will be received.
0xB8	Data byte in I2DAT has been transmitted; ACK has been received.	Load data byte or	X	0	0	0	Last data byte will be transmitted and ACK bit will be received.
		Load data byte	X	0	0	1	Data byte will be transmitted; ACK bit will be received.
0xC0	Data byte in I2DAT has been transmitted; NOT ACK has been received.	No I2DAT action or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address.
		No I2DAT action or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1.
		No I2DAT action or	1	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No I2DAT action	1	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0xC8	Last data byte in I2DAT has been transmitted (AA = 0); ACK has been received.	No I2DAT action or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address.
		No I2DAT action or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR[0] = logic 1.
		No I2DAT action or	1	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General Call address. A START condition will be transmitted when the bus becomes free.
		No I2DAT action	1	0	0	01	Switched to not addressed SLV mode; Own SLA will be recognized; General Call address will be recognized if I2ADR.0 = logic 1. A START condition will be transmitted when the bus becomes free.

### 22.9.6 Miscellaneous states

There are two I2STAT codes that do not correspond to a defined I2C hardware state (see [Table 513](#)). These are discussed below.

#### 22.9.6.1 I2STAT = 0xF8

This status code indicates that no relevant information is available because the serial interrupt flag, SI, is not yet set. This occurs between other states and when the I2C block is not involved in a serial transfer.

#### 22.9.6.2 I2STAT = 0x00

This status code indicates that a bus error has occurred during an I2C serial transfer. A bus error is caused when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. A bus error may also be caused when external interference disturbs the internal I2C block signals. When a bus error occurs, SI is set. To recover from a bus error, the STO flag must be set and SI must be cleared. This causes the I2C block to enter the “not addressed” slave mode (a defined state) and to clear the STO flag (no other bits in I2CON are affected). The SDA and SCL lines are released (a STOP condition is not transmitted).

**Table 513. Miscellaneous States**

Status Code (I2CSTAT)	Status of the I2C-bus and hardware	Application software response				Next action taken by I2C hardware
		To/From I2DAT	To I2CON			
			STA	STO	SI	AA
0xF8	No relevant state information available; SI = 0.	No I2DAT action	No I2CON action			Wait or proceed current transfer.
0x00	Bus error during MST or selected slave modes, due to an illegal START or STOP condition. State 0x00 can also occur when interference causes the I2C block to enter an undefined state.	No I2DAT action	0	1	0	X

### 22.9.7 Some special cases

The I2C hardware has facilities to handle the following special cases that may occur during a serial transfer:

#### 22.9.7.1 Simultaneous repeated START conditions from two masters

A repeated START condition may be generated in the master transmitter or master receiver modes. A special case occurs if another master simultaneously generates a repeated START condition (see [Figure 119](#)). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

If the I<sup>2</sup>C hardware detects a repeated START condition on the I<sup>2</sup>C-bus before generating a repeated START condition itself, it will release the bus, and no interrupt request is generated. If another master frees the bus by generating a STOP condition, the I<sup>2</sup>C block will transmit a normal START condition (state 0x08), and a retry of the total serial data transfer can commence.

#### 22.9.7.2 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see [Figure 113](#)). Loss of arbitration is indicated by the following states in I2STAT; 0x38, 0x68, 0x78, and 0xB0 (see [Figure 115](#) and [Figure 116](#)).

If the STA flag in I2CON is set by the routines which service these states, then, if the bus is free again, a START condition (state 0x08) is transmitted without intervention by the CPU, and a retry of the total serial transfer can commence.

#### 22.9.7.3 Forced access to the I<sup>2</sup>C-bus

In some applications, it may be possible for an uncontrolled source to cause a bus hang-up. In such situations, the problem may be caused by interference, temporary interruption of the bus or a temporary short-circuit between SDA and SCL.

If an uncontrolled source generates a superfluous START or masks a STOP condition, then the I<sup>2</sup>C-bus stays busy indefinitely. If the STA flag is set and bus access is not obtained within a reasonable amount of time, then a forced access to the I<sup>2</sup>C-bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I<sup>2</sup>C hardware behaves as if a STOP condition was received and is able to transmit a START condition. The STO flag is cleared by hardware [Figure 120](#).

#### 22.9.7.4 I<sup>2</sup>C-bus obstructed by a LOW level on SCL or SDA

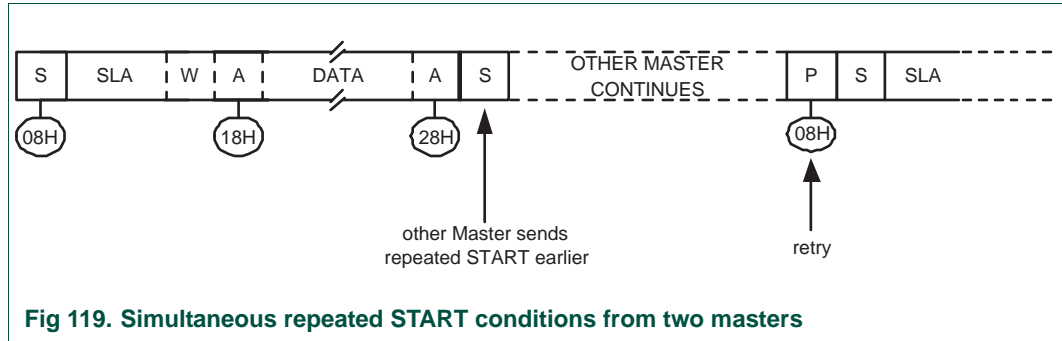
An I<sup>2</sup>C-bus hang-up can occur if either the SDA or SCL line is held LOW by any device on the bus. If the SCL line is obstructed (pulled LOW) by a device on the bus, no further serial transfer is possible, and the problem must be resolved by the device that is pulling the SCL bus line LOW.

Typically, the SDA line may be obstructed by another device on the bus that has become out of synchronization with the current bus master by either missing a clock, or by sensing a noise pulse as a clock. In this case, the problem can be solved by transmitting additional clock pulses on the SCL line [Figure 121](#). The I<sup>2</sup>C interface does not include a dedicated timeout timer to detect an obstructed bus, but this can be implemented using another timer in the system. When detected, software can force clocks (up to 9 may be required) on SCL until SDA is released by the offending device. At that point, the slave may still be out of synchronization, so a START should be generated to insure that all I<sup>2</sup>C peripherals are synchronized.

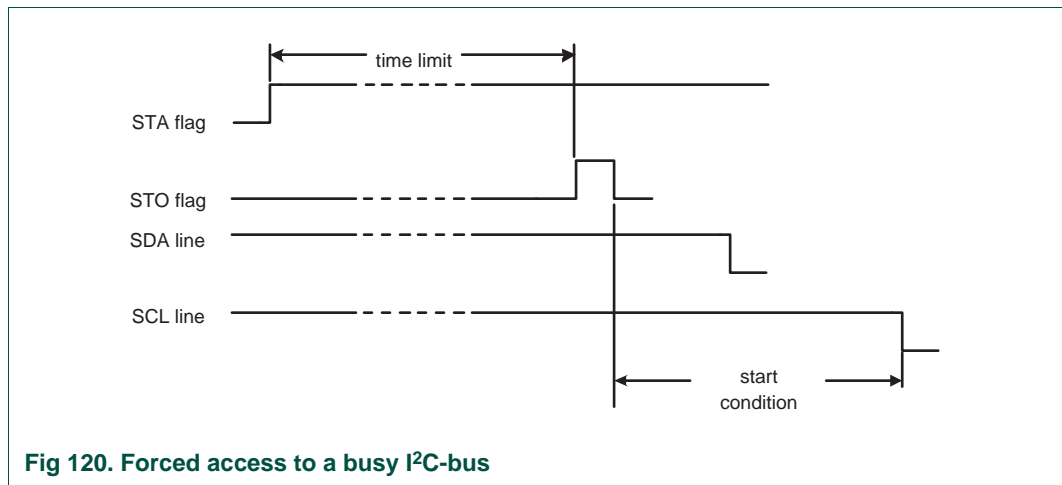
#### 22.9.7.5 Bus error

A bus error occurs when a START or STOP condition is detected at an illegal position in the format frame. Examples of illegal positions are during the serial transfer of an address byte, a data bit, or an acknowledge bit.

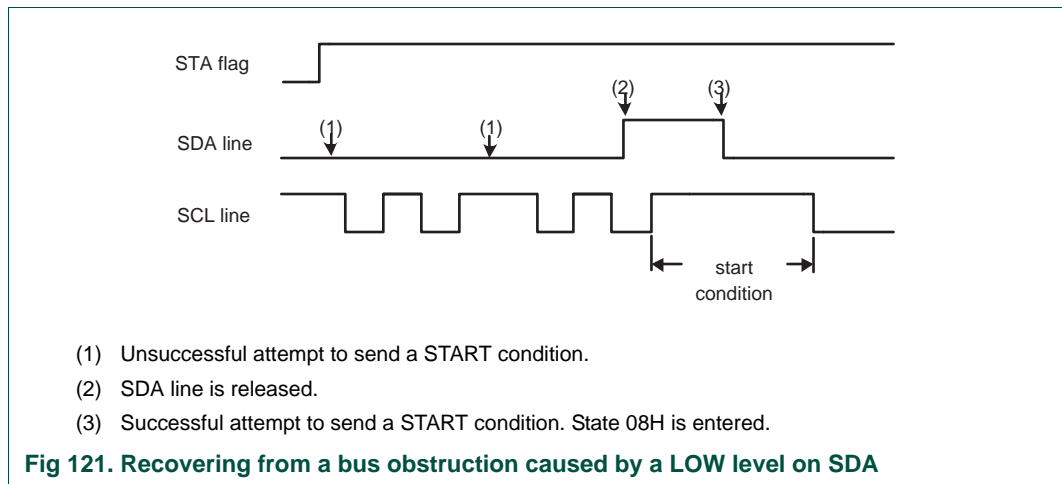
The I<sup>2</sup>C hardware only reacts to a bus error when it is involved in a serial transfer either as a master or an addressed slave. When a bus error is detected, the I<sup>2</sup>C block immediately switches to the not addressed slave mode, releases the SDA and SCL lines, sets the interrupt flag, and loads the status register with 0x00. This status code may be used to vector to a state service routine which either attempts the aborted serial transfer again or simply recovers from the error condition as shown in [Table 513](#).



**Fig 119. Simultaneous repeated START conditions from two masters**



**Fig 120. Forced access to a busy I<sup>2</sup>C-bus**



- (1) Unsuccessful attempt to send a START condition.
- (2) SDA line is released.
- (3) Successful attempt to send a START condition. State 08H is entered.

**Fig 121. Recovering from a bus obstruction caused by a LOW level on SDA**



## 22.9.8 I<sup>2</sup>C state service routines

This section provides examples of operations that must be performed by various I<sup>2</sup>C state service routines. This includes:

- Initialization of the I<sup>2</sup>C block after a Reset.
- I<sup>2</sup>C Interrupt Service
- The 26 state service routines providing support for all four I<sup>2</sup>C operating modes.

### 22.9.8.1 Initialization

In the initialization example, the I<sup>2</sup>C block is enabled for both master and slave modes. For each mode, a buffer is used for transmission and reception. The initialization routine performs the following functions:

- The I2ADR registers and I2MASK registers are loaded with values to configure the part's own slave address(es) and the General Call bit (GC)
- The I<sup>2</sup>C interrupt enable and interrupt priority bits are set
- The slave mode is enabled by simultaneously setting the I2EN and AA bits in I2CON and the serial clock frequency (for master modes) is defined by loading the I2SCLH and I2SCLL registers. The master routines must be started in the main program.

The I<sup>2</sup>C hardware now begins checking the I<sup>2</sup>C-bus for its own slave address and General Call. If the General Call or the own slave address is detected, an interrupt is requested and I2STAT is loaded with the appropriate state information.

### 22.9.8.2 I<sup>2</sup>C interrupt service

When the I<sup>2</sup>C interrupt is entered, I2STAT contains a status code which identifies one of the 26 state services to be executed.

### 22.9.8.3 The state service routines

Each state routine is part of the I<sup>2</sup>C interrupt routine and handles one of the 26 states.

### 22.9.8.4 Adapting state services to an application

The state service examples show the typical actions that must be performed in response to the 26 I<sup>2</sup>C state codes. If one or more of the four I<sup>2</sup>C operating modes are not used, the associated state services can be omitted, as long as care is taken that the those states can never occur.

In an application, it may be desirable to implement some kind of timeout during I<sup>2</sup>C operations, in order to trap an inoperative bus or a lost service routine.

## 22.10 Software example

### 22.10.1 Initialization routine

Example to initialize I<sup>2</sup>C Interface as a Slave and/or Master.

1. Load the I2ADR registers and I2MASK registers with values to configure the own Slave Address, enable General Call recognition if needed.
2. Enable I<sup>2</sup>C interrupt.
3. Write 0x44 to I2CONSET to set the I2EN and AA bits, enabling Slave functions. For Master only functions, write 0x40 to I2CONSET.

### 22.10.2 Start Master Transmit function

Begin a Master Transmit operation by setting up the buffer, pointer, and data count, then initiating a START.

1. Initialize Master data counter.
2. Set up the Slave Address to which data will be transmitted, and add the Write bit.
3. Write 0x20 to I2CONSET to set the STA bit.
4. Set up data to be transmitted in Master Transmit buffer.
5. Initialize the Master data counter to match the length of the message being sent.
6. Exit

### 22.10.3 Start Master Receive function

Begin a Master Receive operation by setting up the buffer, pointer, and data count, then initiating a START.

1. Initialize Master data counter.
2. Set up the Slave Address to which data will be transmitted, and add the Read bit.
3. Write 0x20 to I2CONSET to set the STA bit.
4. Set up the Master Receive buffer.
5. Initialize the Master data counter to match the length of the message to be received.
6. Exit

### 22.10.4 I<sup>2</sup>C interrupt routine

Determine the I<sup>2</sup>C state and which state routine will be used to handle it.

1. Read the I<sup>2</sup>C status from I2STA.
2. Use the status value to branch to one of 26 possible state routines.

### 22.10.5 Non mode specific states

#### 22.10.5.1 State: 0x00

Bus Error. Enter not addressed Slave mode and release bus.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 22.10.5.2 Master States

State 0x08 and State 0x10 are for both Master Transmit and Master Receive modes. The R/W bit decides whether the next state is within Master Transmit mode or Master Receive mode.

#### 22.10.5.3 State: 0x08

A START condition has been transmitted. The Slave Address + R/W bit will now be transmitted.

1. Write Slave Address with R/W bit to I2DAT.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Master Transmit mode data buffer.
5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

#### 22.10.5.4 State: 0x10

A repeated START condition has been transmitted. The Slave Address + R/W bit will now be transmitted.

1. Write Slave Address with R/W bit to I2DAT.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Master Transmit mode data buffer.
5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

### 22.10.6 Master Transmitter states

#### 22.10.6.1 State: 0x18

Previous state was State 0x08 or State 0x10, Slave Address + Write has been transmitted, ACK has been received. The first data byte will be transmitted.

1. Load I2DAT with first data byte from Master Transmit buffer.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment Master Transmit buffer pointer.
5. Exit

### 22.10.6.2 State: 0x20

Slave Address + Write has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 22.10.6.3 State: 0x28

Data has been transmitted, ACK has been received. If the transmitted data was the last data byte then transmit a STOP condition, otherwise transmit the next data byte.

1. Decrement the Master data counter, skip to step 5 if not the last data byte.
2. Write 0x14 to I2CONSET to set the STO and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Exit
5. Load I2DAT with next data byte from Master Transmit buffer.
6. Write 0x04 to I2CONSET to set the AA bit.
7. Write 0x08 to I2CONCLR to clear the SI flag.
8. Increment Master Transmit buffer pointer
9. Exit

### 22.10.6.4 State: 0x30

Data has been transmitted, NOT ACK received. A STOP condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 22.10.6.5 State: 0x38

Arbitration has been lost during Slave Address + Write or data. The bus has been released and not addressed Slave mode is entered. A new START condition will be transmitted when the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

## 22.10.7 Master Receive states

### 22.10.7.1 State: 0x40

Previous state was State 08 or State 10. Slave Address + Read has been transmitted, ACK has been received. Data will be received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.

3. Exit

#### 22.10.7.2 State: 0x48

Slave Address + Read has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 22.10.7.3 State: 0x50

Data has been received, ACK has been returned. Data will be read from I2DAT. Additional data will be received. If this is the last data byte then NOT ACK will be returned, otherwise ACK will be returned.

1. Read data byte from I2DAT into Master Receive buffer.
2. Decrement the Master data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
4. Exit
5. Write 0x04 to I2CONSET to set the AA bit.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment Master Receive buffer pointer
8. Exit

#### 22.10.7.4 State: 0x58

Data has been received, NOT ACK has been returned. Data will be read from I2DAT. A STOP condition will be transmitted.

1. Read data byte from I2DAT into Master Receive buffer.
2. Write 0x14 to I2CONSET to set the STO and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Exit

### 22.10.8 Slave Receiver states

#### 22.10.8.1 State: 0x60

Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 22.10.8.2 State: 0x68

Arbitration has been lost in Slave Address and R/W bit as bus Master. Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK will be returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit.

### 22.10.8.3 State: 0x70

General Call has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 22.10.8.4 State: 0x78

Arbitration has been lost in Slave Address + R/W bit as bus Master. General Call has been received and ACK has been returned. Data will be received and ACK returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 22.10.8.5 State: 0x80

Previously addressed with own Slave Address. Data has been received and ACK has been returned. Additional data will be read.

1. Read data byte from I2DAT into the Slave Receive buffer.
2. Decrement the Slave data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
4. Exit.
5. Write 0x04 to I2CONSET to set the AA bit.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment Slave Receive buffer pointer.
8. Exit

#### 22.10.8.6 State: 0x88

Previously addressed with own Slave Address. Data has been received and NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 22.10.8.7 State: 0x90

Previously addressed with General Call. Data has been received, ACK has been returned. Received data will be saved. Only the first data byte will be received with ACK. Additional data will be received with NOT ACK.

1. Read data byte from I2DAT into the Slave Receive buffer.
2. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
3. Exit

#### 22.10.8.8 State: 0x98

Previously addressed with General Call. Data has been received, NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 22.10.8.9 State: 0xA0

A STOP condition or repeated START has been received, while still addressed as a Slave. Data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 22.10.9 Slave Transmitter states

#### 22.10.9.1 State: 0xA8

Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received.

1. Load I2DAT from Slave Transmit buffer with first data byte.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

### 22.10.9.2 State: 0xB0

Arbitration lost in Slave Address and R/W bit as bus Master. Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received. STA is set to restart Master mode after the bus is free again.

1. Load I2DAT from Slave Transmit buffer with first data byte.
2. Write 0x24 to I2CONSET to set the STA and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

### 22.10.9.3 State: 0xB8

Data has been transmitted, ACK has been received. Data will be transmitted, ACK bit will be received.

1. Load I2DAT from Slave Transmit buffer with data byte.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment Slave Transmit buffer pointer.
5. Exit

### 22.10.9.4 State: 0xC0

Data has been transmitted, NOT ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

### 22.10.9.5 State: 0xC8

The last data byte has been transmitted, ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit



### 23.1 Basic configuration

The I<sup>2</sup>S interface is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCI2S.  
**Remark:** On reset, the I<sup>2</sup>S interface is disabled (PCI2S = 0).
2. Peripheral clock: The functional portion of the I<sup>2</sup>S interface operates from the CPU clock (CCLK), rather than PCLK. The bus interface operates from the common PCLK for APB peripherals. See [Section 4.6.4](#).
3. Pins: Select I<sup>2</sup>S pins and their modes in the relevant IOCON registers (see [Section 8.4.1](#)).
4. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
5. DMA: The I<sup>2</sup>S interface supports two DMA requests, see [Table 521](#) and [Table 522](#), and [Table 655](#).

### 23.2 Features

The I<sup>2</sup>S bus provides a standard communication interface for digital audio applications. The I<sup>2</sup>S bus specification defines a 3-wire serial bus, having one data, one clock, and one word select signal. The basic I<sup>2</sup>S connection has one master, which is always the master, and one slave. The I<sup>2</sup>S interface on the LPC178x/177x provides a separate transmit and receive channel, each of which can operate as either a master or a slave, and an optional oversample master clock output (MCLK).

- The I<sup>2</sup>S input can operate in both master and slave mode.
- The I<sup>2</sup>S output can operate in both master and slave mode, independent of the input.
- Capable of handling 8-bit, 16-bit, and 32-bit word sizes.
- Mono and stereo audio data supported.
- Versatile clocking includes independent transmit and receive fractional rate generators, and an ability to use a single clock input or output for a 4-wire mode.
- Sampling frequencies (fs) supported include standard 16 to 96 kHz ranges (16, 22.05, 32, 44.1, 48, or 96 kHz) for audio applications, and above, depending on the clock frequency.
- Separate Master Clock outputs for both transmit and receive channels support a clock up to 512 times the I<sup>2</sup>S sampling frequency.
- Word Select period in master mode is separately configurable for I<sup>2</sup>S input and output.
- Two 8 word (32 byte) FIFO data buffers, one set each for transmit and receive.
- Generates interrupt requests when buffer levels cross a programmable boundary.
- Two DMA requests, controlled by programmable buffer levels. These are connected to the General Purpose DMA block.
- Controls include reset, stop and mute options separately for I<sup>2</sup>S input and I<sup>2</sup>S output.
- Optional MCLK (oversample) output.

## 23.3 Description

The I<sup>2</sup>S performs serial data out via the transmit channel and serial data in via the receive channel. These support the NXP Inter IC Audio format for 8-bit, 16-bit and 32-bit audio data, both for stereo and mono modes. Configuration, data access and control is performed by a APB register set. Data streams are buffered by FIFOs with a depth of 8 words.

The I<sup>2</sup>S receive and transmit stage can operate independently in either slave or master mode. Within the I<sup>2</sup>S module the difference between these modes lies in the word select (WS) signal which determines the timing of data transmissions. Data words start on the next falling edge of the transmitting clock after a WS change. In stereo mode when WS is low left data is transmitted and right data when WS is high. In mono mode the same data is transmitted twice, once when WS is low and again when WS is high.

- In master mode, word select is generated internally with a 9-bit counter. The half period count value of this counter can be set in the control register.
- In slave mode, word select is input from the relevant bus pin.
- When an I<sup>2</sup>S bus is active, the word select, receive clock and transmit clock signals are sent continuously by the bus master, while data is sent continuously by the transmitter.
- Disabling the I<sup>2</sup>S can be done with the stop or mute control bits separately for the transmit and receive.
- The stop bit will disable accesses by the transmit channel or the receive channel to the FIFOs and will place the transmit channel in mute mode.
- The mute control bit will place the transmit channel in mute mode. In mute mode, the transmit channel FIFO operates normally, but the output is discarded and replaced by zeroes. This bit does not affect the receive channel, data reception can occur normally.

### 23.4 Pin descriptions

Table 514. Pin descriptions

Pin Name	Type	Description
I2S_RX_CLK	Input/ Output	Receive Clock. A clock signal used to synchronize the transfer of data on the receive channel. It is driven by the master and received by the slave. Corresponds to the signal SCK in the I <sup>2</sup> S bus specification.
I2S_RX_WS	Input/ Output	Receive Word Select. Selects the channel from which data is to be received. It is driven by the master and received by the slave. Corresponds to the signal WS in the I <sup>2</sup> S bus specification. WS = 0 indicates that data is being received by channel 1 (left channel). WS = 1 indicates that data is being received by channel 2 (right channel).
I2S_RX_SDA	Input/ Output	Receive Data. Serial data, received MSB first. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I <sup>2</sup> S bus specification.
I2S_RX_MCLK	Output	Optional master clock output for the I <sup>2</sup> S receive function.
I2S_TX_CLK	Input/ Output	Transmit Clock. A clock signal used to synchronize the transfer of data on the transmit channel. It is driven by the master and received by the slave. Corresponds to the signal SCK in the I <sup>2</sup> S bus specification.
I2S_TX_WS	Input/ Output	Transmit Word Select. Selects the channel to which data is being sent. It is driven by the master and received by the slave. Corresponds to the signal WS in the I <sup>2</sup> S bus specification. WS = 0 indicates that data is being sent to channel 1 (left channel). WS = 1 indicates that data is being sent to channel 2 (right channel).
I2S_TX_SDA	Input/ Output	Transmit Data. Serial data, sent MSB first. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I <sup>2</sup> S bus specification.
I2S_TX_MCLK	Output	Optional master clock output for the I <sup>2</sup> S transmit function.

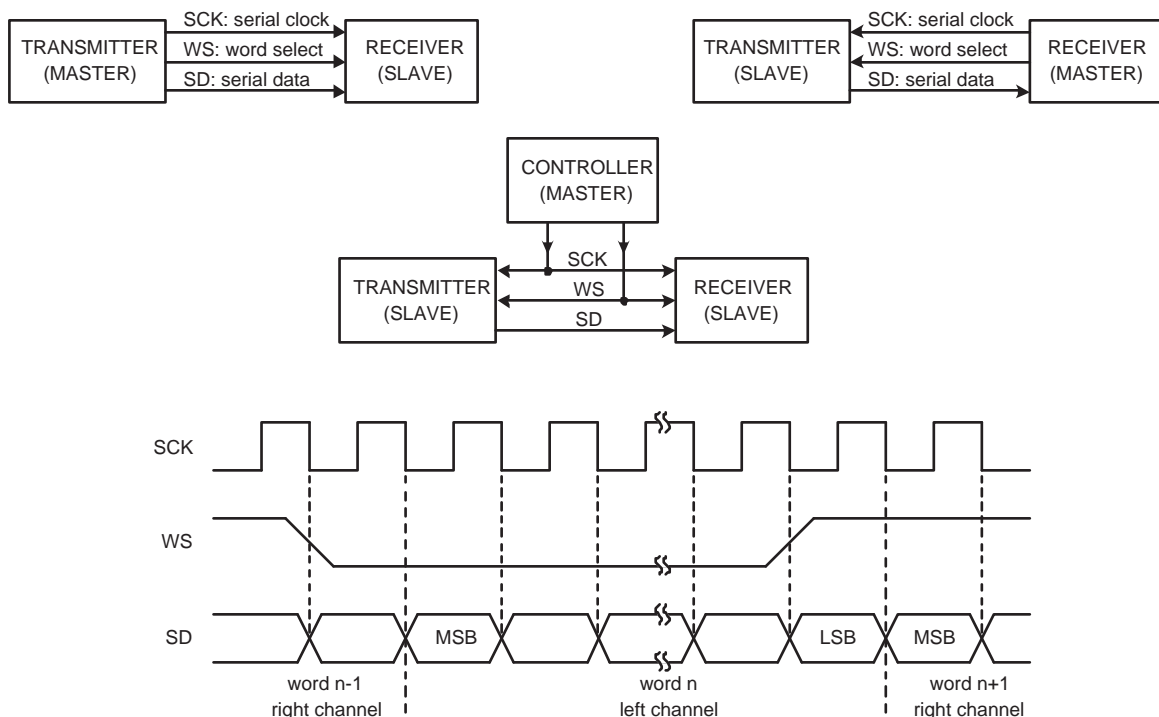


Fig 122. Simple I<sup>2</sup>S configurations and bus timing

## 23.5 Register description

[Table 515](#) shows the registers associated with the I<sup>2</sup>S interface and a summary of their functions. Following the table are details for each register.

**Table 515. I<sup>2</sup>S register map**

Name	Description	Access	Reset Value <sup>[1]</sup>	Address	Table
I2SDAO	Digital Audio Output Register. Contains control bits for the I <sup>2</sup> S transmit channel.	R/W	0x87E1	0x400A 8000	<a href="#">516</a>
I2SDAI	Digital Audio Input Register. Contains control bits for the I <sup>2</sup> S receive channel.	R/W	0x07E1	0x400A 8004	<a href="#">517</a>
I2STXFIFO	Transmit FIFO. Access register for the 8 × 32-bit transmitter FIFO.	WO	0	0x400A 8008	<a href="#">518</a>
I2SRXFIFO	Receive FIFO. Access register for the 8 × 32-bit receiver FIFO.	RO	0	0x400A 800C	<a href="#">519</a>
I2SSTATE	Status Feedback Register. Contains status information about the I <sup>2</sup> S interface.	RO	0x7	0x400A 8010	<a href="#">520</a>
I2SDMA1	DMA Configuration Register 1. Contains control information for DMA request 1.	R/W	0	0x400A 8014	<a href="#">521</a>
I2SDMA2	DMA Configuration Register 2. Contains control information for DMA request 2.	R/W	0	0x400A 8018	<a href="#">522</a>
I2SIRQ	Interrupt Request Control Register. Contains bits that control how the I <sup>2</sup> S interrupt request is generated.	R/W	0	0x400A 801C	<a href="#">523</a>
I2STXRATE	Transmit reference clock divider. This register determines the I <sup>2</sup> S TX_REF rate by specifying the value to divide CCLK by in order to produce TX_REF.	R/W	0	0x400A 8020	<a href="#">524</a>
I2SRXRATE	Receive reference clock divider. This register determines the I <sup>2</sup> S RX_REF rate by specifying the value to divide CCLK by in order to produce RX_REF.	R/W	0	0x400A 8024	<a href="#">525</a>
I2STXBITRATE	Transmit bit rate divider. This register determines the I <sup>2</sup> S transmit bit rate by specifying the value to divide TX_REF by in order to produce the transmit bit clock.	R/W	0	0x400A 8028	<a href="#">526</a>
I2SRXBITRATE	Receive bit rate divider. This register determines the I <sup>2</sup> S receive bit rate by specifying the value to divide RX_REF by in order to produce the receive bit clock.	R/W	0	0x400A 802C	<a href="#">527</a>
I2STXMODE	Transmit mode control.	R/W	0	0x400A 8030	<a href="#">528</a>
I2SRXMODE	Receive mode control.	R/W	0	0x400A 8034	<a href="#">529</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 23.5.1 Digital Audio Output register (I2SDAO - 0x400A 8000)

The I2SDAO register controls the operation of the I<sup>2</sup>S transmit channel. The function of bits in DAO are shown in [Table 516](#).

**Table 516: Digital Audio Output register (I2SDAO - address 0x400A 8000) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	wordwidth		Selects the number of bytes in data as follows:	01
		00	8-bit data	
		01	16-bit data	
		10	Reserved, do not use this setting	
		11	32-bit data	
2	mono		When 1, data is of monaural format. When 0, the data is in stereo format.	0
3	stop		When 1, disables accesses on FIFOs, places the transmit channel in mute mode.	0
4	reset		When 1, asynchronously resets the transmit channel and FIFO.	0
5	ws_sel		When 0, the interface is in master mode. When 1, the interface is in slave mode. See <a href="#">Section 23.7</a> for a summary of useful combinations for this bit with I2STXMODE.	1
14:6	ws_halfperiod		Word select half period minus 1, i.e. WS 64clk period -> ws_halfperiod = 31.	0x1F
15	mute		When 1, the transmit channel sends only zeroes.	1
31:16	-		Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.2 Digital Audio Input register (I2SDAI - 0x400A 8004)

The I2SDAI register controls the operation of the I<sup>2</sup>S receive channel. The function of bits in DAI are shown in [Table 517](#).

**Table 517: Digital Audio Input register (I2SDAI - address 0x400A 8004) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	wordwidth		Selects the number of bytes in data as follows:	01
		00	8-bit data	
		01	16-bit data	
		10	Reserved, do not use this setting	
		11	32-bit data	
2	mono		When 1, data is of monaural format. When 0, the data is in stereo format.	0
3	stop		When 1, disables accesses on FIFOs, places the transmit channel in mute mode.	0
4	reset		When 1, asynchronously reset the transmit channel and FIFO.	0
5	ws_sel		When 0, the interface is in master mode. When 1, the interface is in slave mode. See <a href="#">Section 23.7</a> for a summary of useful combinations for this bit with I2SRXMODE.	1
14:6	ws_halfperiod		Word select half period minus 1, i.e. WS 64clk period -> ws_halfperiod = 31.	0x1F
31:15	-		Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.3 Transmit FIFO register (I2STXFIFO - 0x400A 8008)

The I2STXFIFO register provides access to the transmit FIFO. The function of bits in I2STXFIFO are shown in [Table 518](#).

**Table 518: Transmit FIFO register (I2STXFIFO - address 0x400A 8008) bit description**

Bit	Symbol	Description	Reset Value
31:0	I2STXFIFO	8 × 32-bit transmit FIFO.	Level = 0

### 23.5.4 Receive FIFO register (I2SRXFIFO - 0x400A 800C)

The I2SRXFIFO register provides access to the receive FIFO. The function of bits in I2SRXFIFO are shown in [Table 519](#).

**Table 519: Receive FIFO register (I2SRXFIFO - address 0x400A 800C) bit description**

Bit	Symbol	Description	Reset Value
31:0	I2SRXFIFO	8 × 32-bit transmit FIFO.	level = 0

### 23.5.5 Status Feedback register (I2SSTATE - 0x400A 8010)

The I2SSTATE register provides status information about the I<sup>2</sup>S interface. The meaning of bits in I2SSTATE are shown in [Table 520](#).

**Table 520: Status Feedback register (I2SSTATE - address 0x400A 8010) bit description**

Bit	Symbol	Description	Reset Value
0	irq	This bit reflects the presence of Receive Interrupt or Transmit Interrupt. This is determined by comparing the current FIFO levels to the rx_depth_irq and tx_depth_irq fields in the I2SIRQ register.	1
1	dmareq1	This bit reflects the presence of Receive or Transmit DMA Request 1. This is determined by comparing the current FIFO levels to the rx_depth_dma1 and tx_depth_dma1 fields in the I2SDMA1 register.	1
2	dmareq2	This bit reflects the presence of Receive or Transmit DMA Request 2. This is determined by comparing the current FIFO levels to the rx_depth_dma2 and tx_depth_dma2 fields in the I2SDMA2 register.	1
7:3	Unused	Unused.	0
11:8	rx_level	Reflects the current level of the Receive FIFO.	0
15:12	-	Reserved. Read value is undefined, only zero should be written.	NA
19:16	tx_level	Reflects the current level of the Transmit FIFO.	0
31:20	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.6 DMA Configuration Register 1 (I2SDMA1 - 0x400A 8014)

The I2SDMA1 register controls the operation of DMA request 1. The function of bits in I2SDMA1 are shown in [Table 521](#). Refer to the General Purpose DMA Controller chapter for details of DMA operation.

**Table 521: DMA Configuration register 1 (I2SDMA1 - address 0x400A 8014) bit description**

Bit	Symbol	Description	Reset Value
0	rx_dma1_enable	When 1, enables DMA1 for I <sup>2</sup> S receive.	0
1	tx_dma1_enable	When 1, enables DMA1 for I <sup>2</sup> S transmit.	0
7:2	-	Reserved. Read value is undefined, only zero should be written.	0
11:8	rx_depth_dma1	Set the FIFO level that triggers a receive DMA request on DMA1.	0
15:12	-	Reserved. Read value is undefined, only zero should be written.	NA
19:16	tx_depth_dma1	Set the FIFO level that triggers a transmit DMA request on DMA1.	0
31:20	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.7 DMA Configuration Register 2 (I2SDMA2 - 0x400A 8018)

The I2SDMA2 register controls the operation of DMA request 2. The function of bits in I2SDMA2 are shown in [Table 516](#).

**Table 522: DMA Configuration register 2 (I2SDMA2 - address 0x400A 8018) bit description**

Bit	Symbol	Description	Reset Value
0	rx_dma2_enable	When 1, enables DMA1 for I <sup>2</sup> S receive.	0
1	tx_dma2_enable	When 1, enables DMA1 for I <sup>2</sup> S transmit.	0
7:2	Unused	Unused.	0
11:8	rx_depth_dma2	Set the FIFO level that triggers a receive DMA request on DMA2.	0
15:12	-	Reserved. Read value is undefined, only zero should be written.	NA
19:16	tx_depth_dma2	Set the FIFO level that triggers a transmit DMA request on DMA2.	0
31:20	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.8 Interrupt Request Control register (I2SIRQ - 0x400A 801C)

The I2SIRQ register controls the operation of the I<sup>2</sup>S interrupt request. The function of bits in I2SIRQ are shown in [Table 516](#).

**Table 523: Interrupt Request Control register (I2SIRQ - address 0x400A 801C) bit description**

Bit	Symbol	Description	Reset Value
0	rx_irq_enable	When 1, enables I <sup>2</sup> S receive interrupt.	0
1	tx_irq_enable	When 1, enables I <sup>2</sup> S transmit interrupt.	0
7:2	Unused	Unused.	0
11:8	rx_depth_irq	Set the FIFO level on which to create an irq request.	0
15:12	-	Reserved. Read value is undefined, only zero should be written.	NA
19:16	tx_depth_irq	Set the FIFO level on which to create an irq request.	0
31:20	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.9 Transmit Clock Rate register (I2STXRATE - 0x400A 8020)

The TX\_REF rate for the I<sup>2</sup>S transmitter is determined by the values in the I2STXRATE register. The required I2STXRATE setting depends on the desired audio sample rate desired, the format (stereo/mono) used, and the data size. When the MCLK output is enabled for the transmit function, it is TX\_REF that is sent to the I2S\_TX\_MCLK pin.

The TX\_REF rate is generated using a fractional rate generator, dividing down the frequency of CCLK. Values of the numerator (X) and the denominator (Y) must be chosen to produce a frequency twice that desired for TX\_REF, which must be an integer multiple of the transmitter bit clock rate. Fractional rate generators have some aspects that the user should be aware of when choosing settings. These are discussed in [Section 23.5.9.1](#). The equation for the fractional rate generator is:

$$\text{I2S TX\_REF} = \text{CCLK} * (\text{X/Y}) / 2$$

Note: If the value of X or Y is 0, the clock divider is bypassed. Also, the value of Y must be greater than or equal to X.

**Table 524: Transmit Clock Rate register (I2TXRATE - address 0x400A 8020) bit description**

Bit	Symbol	Description	Reset Value
7:0	Y_divider	I <sup>2</sup> S transmit TX_REF rate denominator. This value is used to divide CCLK to produce TX_REF. Eight bits of fractional divide supports a wide range of possibilities. A value of 0 causes the clock divider to be bypassed.	0
15:8	X_divider	I <sup>2</sup> S transmit TX_REF rate numerator. This value is used to multiply CCLK by to produce the TX_REF. A value of 0 causes the clock divider to be bypassed. Eight bits of fractional divide supports a wide range of possibilities. Note: the resulting ratio X/Y is divided by 2.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.9.1 Notes on fractional rate generators

The nature of a fractional rate generator is that there will be some output jitter with some divide settings. This is because the fractional rate generator is a fully digital function, so output clock transitions are synchronous with the source clock, whereas a theoretical perfect fractional rate may have edges that are not related to the source clock. So, output jitter will not be greater than plus or minus one source clock between consecutive clock edges.

For example, if X = 0x07 and Y = 0x11, the fractional rate generator will output 7 clocks for every 17 (11 hex) input clocks, distributed as evenly as it can. In this example, there is no way to distribute the output clocks in a perfectly even fashion, so some clocks will be longer than others. The output is divided by 2 in order to square it up, which also helps with the jitter. The frequency averages out to exactly  $(7/17) / 2$ , but some clocks will be a slightly different length than their neighbors. It is possible to avoid jitter entirely by choosing fractions such that X divides evenly into Y, such as 2/4, 2/6, 3/9, 1/N, etc.

### 23.5.10 Receive Clock Rate register (I2SRXRATE - 0x400A 8024)

The RX\_REF rate for the I<sup>2</sup>S receiver is determined by the values in the I2SRXRATE register. The required I2SRXRATE setting depends on the CPU clock rate (CCLK) and the desired RX\_REF rate (such as 256 fs). When the MCLK output is enabled for the receive function, it is RX\_REF that is sent to the I2S\_RX\_MCLK pin.

The RX\_REF rate is generated using a fractional rate generator, dividing down the frequency of CCLK. Values of the numerator (X) and the denominator (Y) must be chosen to produce a frequency twice that desired for the RX\_REF, which must be an integer multiple of the receiver bit clock rate. Fractional rate generators have some aspects that the user should be aware of when choosing settings. These are discussed in [Section 23.5.9.1](#). The equation for the fractional rate generator is:

$$\text{I2S RX\_REF} = \text{CCLK} * (\text{X}/\text{Y}) / 2$$

Note: If the value of X or Y is 0, the clock divider is bypassed. Also, the value of Y must be greater than or equal to X.



**Table 525: Receive Clock Rate register (I2SRXRATE - address 0x400A 8024) bit description**

Bit	Symbol	Description	Reset Value
7:0	Y_divider	I <sup>2</sup> S receive RX_REF rate denominator. This value is used to divide CCLK to produce RX_REF. Eight bits of fractional divide supports a wide range of possibilities. A value of 0 causes the clock divider to be bypassed.	0
15:8	X_divider	I <sup>2</sup> S receive RX_REF rate numerator. This value is used to multiply CCLK by to produce RX_REF. A value of 0 causes the clock divider to be bypassed. Eight bits of fractional divide supports a wide range of possibilities. Note: the resulting ratio X/Y is divided by 2.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.11 Transmit Clock Bit Rate register (I2STXBITRATE - 0x400A 8028)

The bit rate for the I<sup>2</sup>S transmitter is determined by the value of the I2STXBITRATE register. The value depends on the audio sample rate desired, and the data size and format (stereo/mono) used. For example, a 48 kHz sample rate for 16-bit stereo data requires a bit rate of  $48,000 \times 16 \times 2 = 1.536$  MHz.

**Table 526: Transmit Clock Bit Rate register (I2TXBITRATE - address 0x400A 8028) bit description**

Bit	Symbol	Description	Reset Value
5:0	tx_bitrate	I <sup>2</sup> S transmit bit rate. This value plus one is used to divide TX_REF to produce the transmit bit clock.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.12 Receive Clock Bit Rate register (I2SRXBITRATE - 0x400A 802C)

The bit rate for the I<sup>2</sup>S receiver is determined by the value of the I2SRXBITRATE register. The value depends on the audio sample rate, as well as the data size and format used. The calculation is the same as for I2SRXBITRATE.

**Table 527: Receive Clock Rate Bit register (I2SRXBITRATE - address 0x400A 802C) bit description**

Bit	Symbol	Description	Reset Value
5:0	rx_bitrate	I <sup>2</sup> S receive bit rate. This value plus one is used to divide RX_REF to produce the receive bit clock.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.13 Transmit Mode Control register (I2STXMODE - 0x400A 8030)

The Transmit Mode Control register contains additional controls for transmit clock source, enabling the 4-pin mode, how TX\_REF is used, and whether the MCLK output is enabled. See [Section 23.7](#) for a summary of useful mode combinations.

**Table 528: Transmit Mode Control register (I2STXMODE - 0x400A 8030) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	TXCLKSEL		Clock source selection for the transmit bit clock divider.	0
		00	Select the TX fractional rate divider clock output as the source	
		01	Reserved	
		10	Select the RX_REF signal as the TX_REF clock source	
		11	Reserved	
2	TX4PIN		Transmit 4-pin mode selection. When 1, enables 4-pin mode.	0
3	TXMCENA		Enable for the TX_MCLK output.	0
		0	Output of TX_MCLK to a pin is disabled.	
		1	Output of TX_MCLK to a pin is enabled.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 23.5.14 Receive Mode Control register (I2SRXMODE - 0x400A 8034)

The Receive Mode Control register contains additional controls for receive clock source, enabling the 4-pin mode, and how RX\_REF is used. See [Section 23.7](#) for a summary of useful mode combinations.

**Table 529: Receive Mode Control register (I2SRXMODE - 0x400A 8034) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	RXCLKSEL		Clock source selection for the receive bit clock divider.	0
		00	Select the RX fractional rate divider clock output as the source	
		01	Reserved	
		10	Select the TX_REF signal as the RX_REF clock source	
		11	Reserved	
2	RX4PIN		Receive 4-pin mode selection. When 1, enables 4-pin mode.	0
3	RXMCENA		Enable for the RX_MCLK output.	0
		0	Output of RX_MCLK to a pin is disabled.	
		1	Output of RX_MCLK to a pin is enabled.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

## 23.6 I<sup>2</sup>S transmit and receive interfaces

The I<sup>2</sup>S interface can transmit and receive 8-bit, 16-bit or 32-bit stereo or mono audio information. Some details of I<sup>2</sup>S implementation are:

- When the FIFO is empty, the transmit channel will repeat transmitting the same data until new data is written to the FIFO.
- When mute is true, the data value 0 is transmitted.
- When mono is false, two successive data words are respectively left and right data.
- Data word length is determined by the wordwidth value in the configuration register. There is a separate wordwidth value for the receive channel and the transmit channel.
  - 0: word is considered to contain four 8-bit data words.
  - 1: word is considered to contain two 16-bit data words.
  - 3: word is considered to contain one 32-bit data word.
- When the transmit FIFO contains insufficient data the transmit channel will repeat transmitting the last data until new data is available. This can occur when the microprocessor or the DMA at some time is unable to provide new data fast enough. Because of this delay in new data there is a need to fill the gap, which is accomplished by continuing to transmit the last sample. The data is not muted as this would produce an noticeable and undesirable effect in the sound.
- The transmit channel and the receive channel only handle 32-bit aligned words, data chunks must be clipped or extended to a multiple of 32 bits.

When switching between data width or modes the I<sup>2</sup>S must be reset via the reset bit in the control register in order to ensure correct synchronization. It is advisable to set the stop bit also until sufficient data has been written in the transmit FIFO. Note that when stopped data output is muted.

All data accesses to FIFOs are 32 bits. [Figure 135](#) shows the possible data sequences.

A data sample in the FIFO consists of:

- 1×32 bits in 8-bit or 16-bit stereo modes.
- 1×32 bits in mono modes.
- 2×32 bits, first left data, second right data, in 32-bit stereo modes.

Data is read from the transmit FIFO after the falling edge of WS, it will be transferred to the transmit clock domain after the rising edge of WS. On the next falling edge of WS the left data will be loaded in the shift register and transmitted and on the following rising edge of WS the right data is loaded and transmitted.

The receive channel will start receiving data after a change of WS. When word select becomes low it expects this data to be left data, when WS is high received data is expected to be right data. Reception will stop when the bit counter has reached the limit set by wordwidth. On the next change of WS the received data will be stored in the appropriate hold register. When complete data is available it will be written into the receive FIFO.

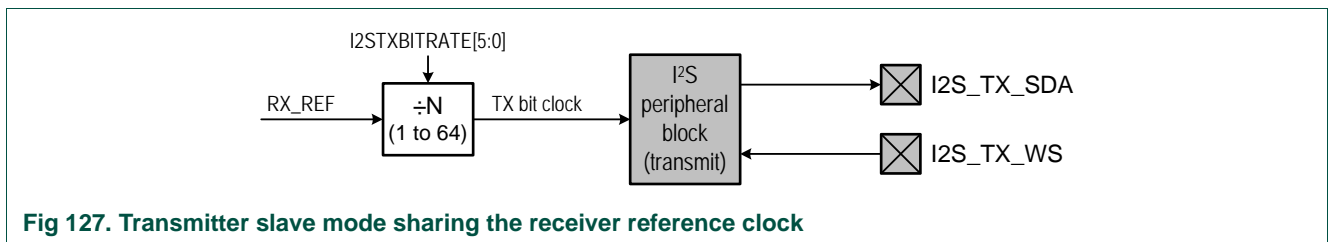
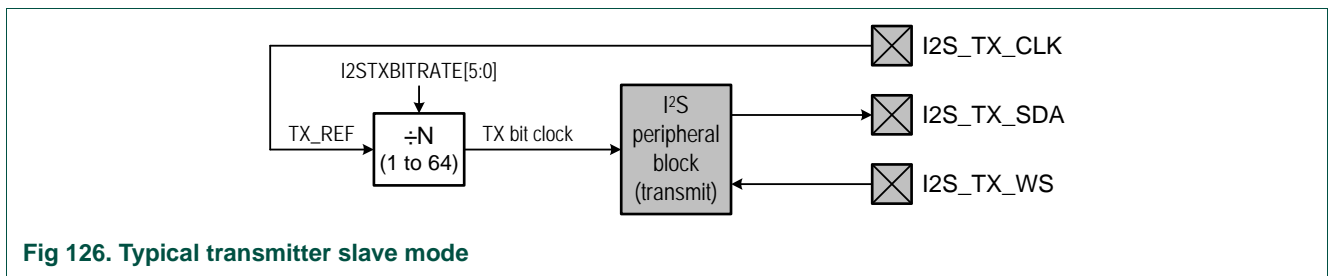
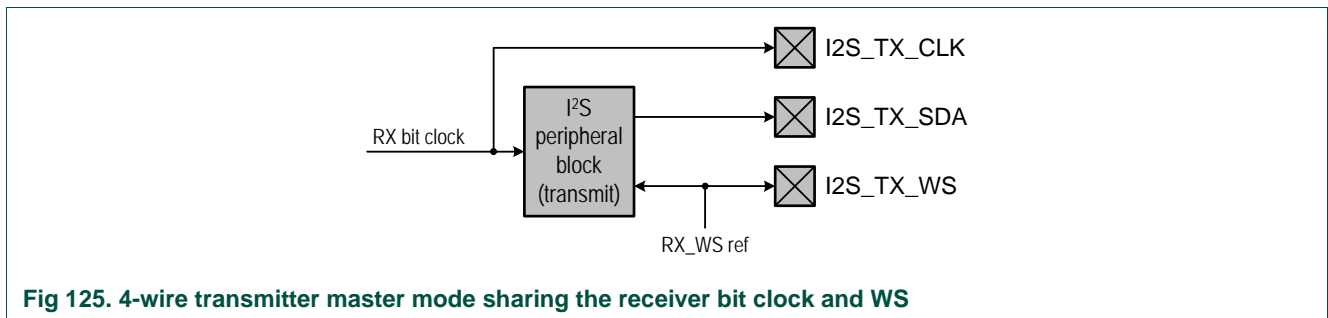
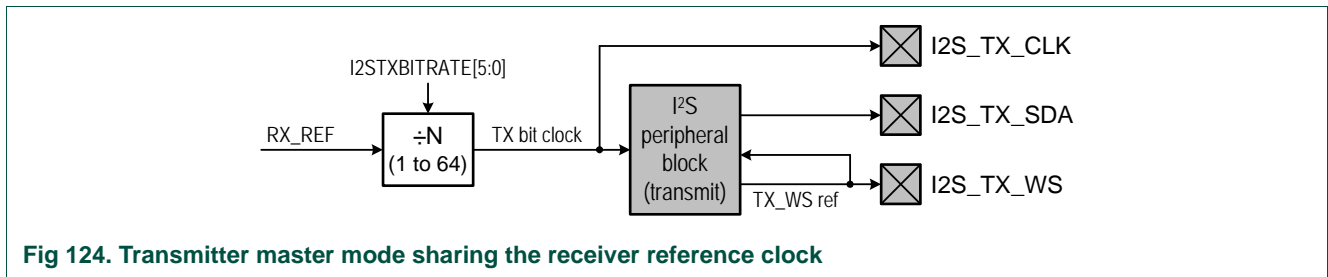
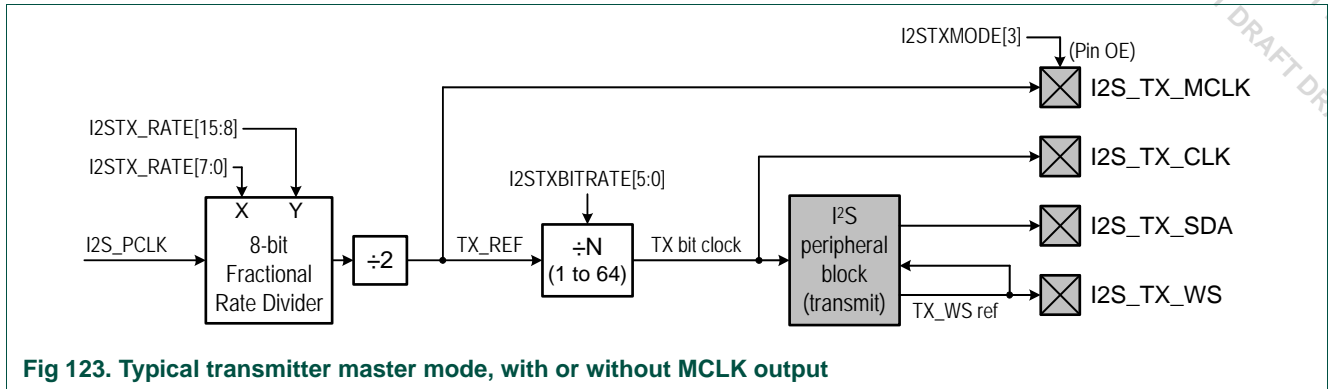
### 23.7 I<sup>2</sup>S operating modes

The clocking and WS usage of the I<sup>2</sup>S interface is configurable. In addition to master and slave modes, which are independently configurable for the transmitter and the receiver, several different clock sources are possible, including variations that share the clock and/or WS between the transmitter and receiver. This last option allows using I<sup>2</sup>S with fewer pins, typically four.

Many configurations are possible that are not considered useful, the following tables and figures give details of the configurations that are most likely to be useful.

**Table 530: I<sup>2</sup>S transmit modes**

I2SDAO [5]	I2STXMODE [3:0]	Description
0	0 0 0 0	Typical transmitter master mode. See <a href="#">Figure 123</a> . The I <sup>2</sup> S transmit function operates as a master. The transmit clock source is the fractional rate divider. The WS used is the internally generated TX_WS. TX_MCLK is not output on the I2S_TX_MCLK pin.
0	0 0 1 0	Transmitter master mode sharing the receiver reference clock. See <a href="#">Figure 124</a> . The I <sup>2</sup> S transmit function operates as a master. The transmit clock source is RX_REF. The WS used is the internally generated TX_WS. TX_MCLK is not output on the I2S_TX_MCLK pin.
0	0 1 0 0	4-wire transmitter master mode sharing the receiver bit clock and WS. See <a href="#">Figure 125</a> . The I <sup>2</sup> S transmit function operates as a master. The transmit clock source is the RX bit clock. The WS used is the internally generated RX_WS. TX_MCLK is not output on the I2S_TX_MCLK pin.
0	1 0 0 0	Transmitter master mode with TC_MCLK output. See <a href="#">Figure 123</a> . The I <sup>2</sup> S transmit function operates as a master. The transmit clock source is the fractional rate divider. The WS used is the internally generated TX_WS. TX_MCLK is output on the I2S_TX_MCLK pin.
1	0 0 0 0	Typical transmitter slave mode. See <a href="#">Figure 126</a> . The I <sup>2</sup> S transmit function operates as a slave. The transmit clock source is the TX_CLK pin. The WS used is the TX_WS pin.
1	0 0 1 0	Transmitter slave mode sharing the receiver reference clock. See <a href="#">Figure 127</a> . The I <sup>2</sup> S transmit function operates as a slave. The transmit clock source is RX_REF. The WS used is the TX_WS pin.
1	0 1 0 0	4-wire transmitter slave mode sharing the receiver bit clock and WS. See <a href="#">Figure 128</a> . The I <sup>2</sup> S transmit function operates as a slave. The transmit clock source is the RX bit clock. The WS used is RX_WS ref.



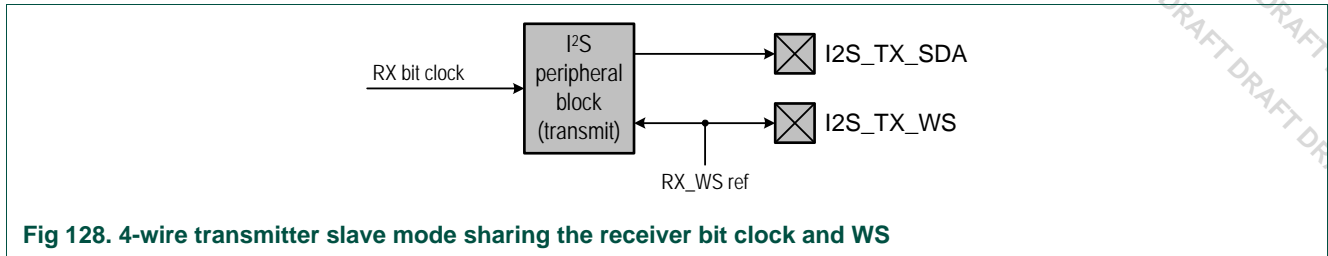


Fig 128. 4-wire transmitter slave mode sharing the receiver bit clock and WS

Table 531: I<sup>2</sup>S receive modes

I2SDAI [5]	I2SRXMODE [3:0]	Description
0	0 0 0 0	Typical receiver master mode. See <a href="#">Figure 129</a> . The I <sup>2</sup> S receive function operates as a master. The receive clock source is the fractional rate divider. The WS used is the internally generated RX_WS. RX_MCLK is not output on the I2S_RX_MCLK pin.
0	0 0 1 0	Receiver master mode sharing the transmitter reference clock. See <a href="#">Figure 130</a> . The I <sup>2</sup> S receive function operates as a master. The receive clock source is TX_REF. The WS used is the internally generated RX_WS. RX_MCLK is not output on the I2S_RX_MCLK pin.
0	0 1 0 0	4-wire receiver master mode sharing the transmitter bit clock and WS. See <a href="#">Figure 131</a> . The I <sup>2</sup> S receive function operates as a master. The receive clock source is the TX bit clock. The WS used is the internally generated TX_WS. RX_MCLK is not output on the I2S_RX_MCLK pin.
0	1 0 0 0	Receiver master mode with RX_MCLK output. See <a href="#">Figure 129</a> . The I <sup>2</sup> S receive function operates as a master. The receive clock source is the fractional rate divider. The WS used is the internally generated RX_WS. RX_MCLK is output on the I2S_RX_MCLK pin.
1	0 0 0	Typical receiver slave mode. See <a href="#">Figure 132</a> . The I <sup>2</sup> S receive function operates as a slave. The receive clock source is the RX_CLK pin. The WS used is the RX_WS pin.
1	0 1 0	Receiver slave mode sharing the transmitter reference clock. See <a href="#">Figure 133</a> . The I <sup>2</sup> S receive function operates as a slave. The receive clock source is TX_REF. The WS used is the RX_WS pin.
1	1 0 0	This is a 4-wire receiver slave mode sharing the transmitter bit clock and WS. See <a href="#">Figure 134</a> . The I <sup>2</sup> S receive function operates as a slave. The receive clock source is the TX bit clock. The WS used is TX_WS ref.

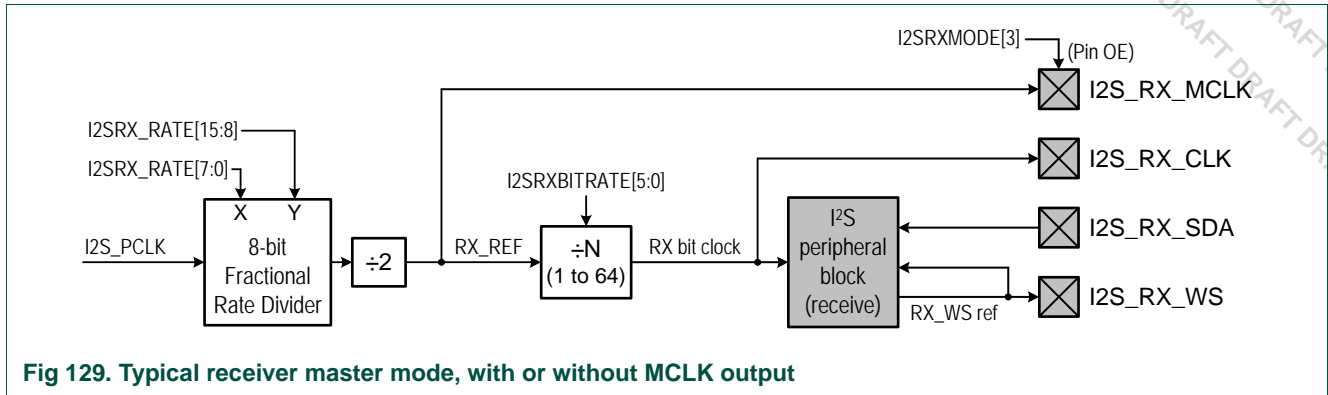


Fig 129. Typical receiver master mode, with or without MCLK output

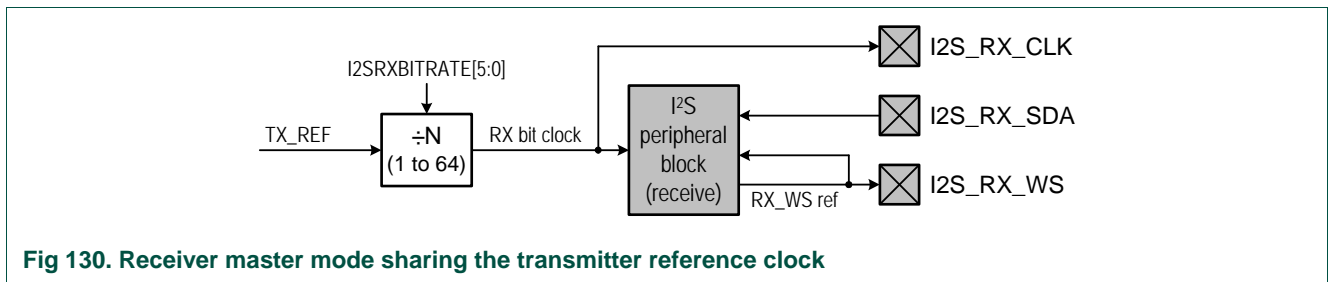


Fig 130. Receiver master mode sharing the transmitter reference clock

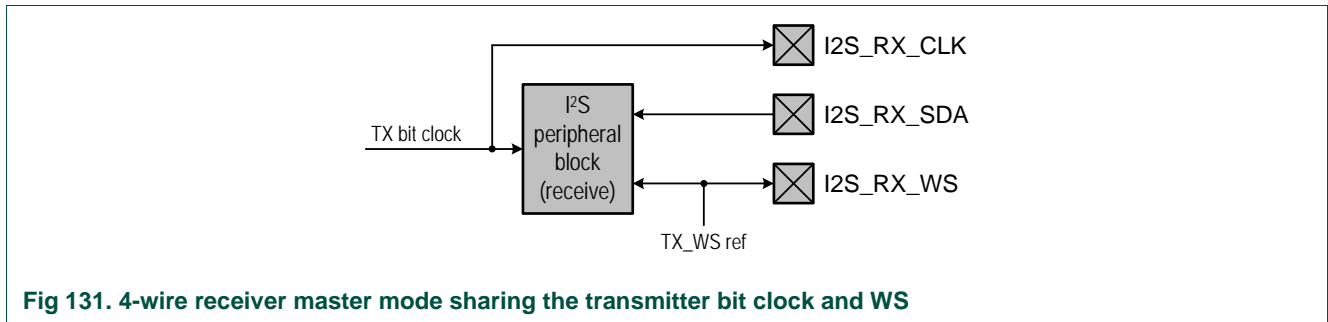


Fig 131. 4-wire receiver master mode sharing the transmitter bit clock and WS

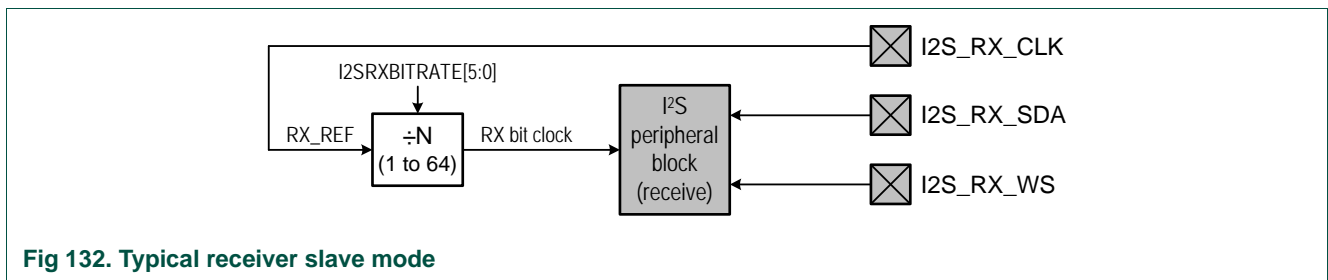


Fig 132. Typical receiver slave mode

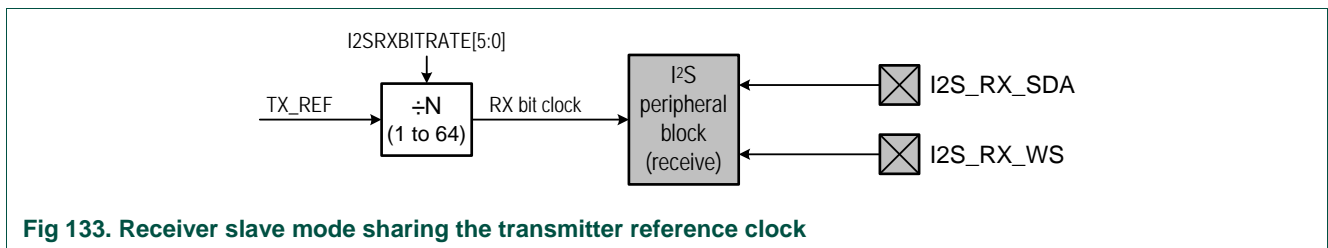


Fig 133. Receiver slave mode sharing the transmitter reference clock

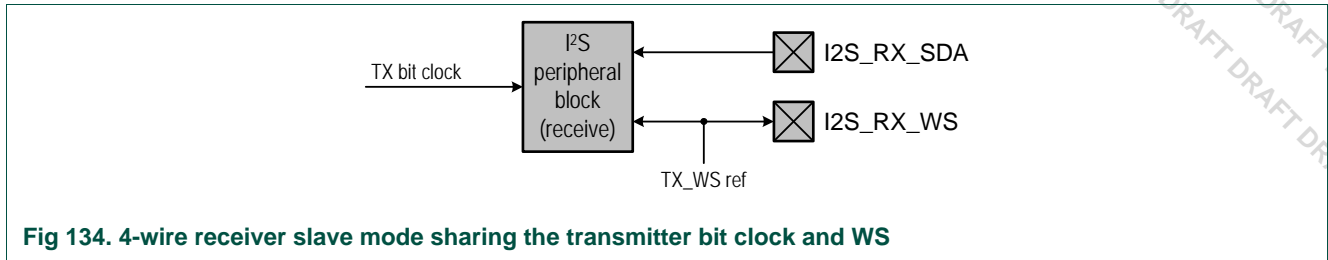


Fig 134. 4-wire receiver slave mode sharing the transmitter bit clock and WS

### 23.8 FIFO controller

Handling of data for transmission and reception is performed via the FIFO controller which can generate two DMA requests and an interrupt request. The controller consists of a set of comparators which compare FIFO levels with depth settings contained in registers. The current status of the level comparators can be seen in the APB status register.

Table 532. Conditions for FIFO level comparison

Level Comparison	Condition
dmareq_tx_1	tx_depth_dma1 >= tx_level
dmareq_rx_1	rx_depth_dma1 <= rx_level
dmareq_tx_2	tx_depth_dma2 >= tx_level
dmareq_rx_2	rx_depth_dma2 <= rx_level
irq_tx	tx_depth_irq >= tx_level
irq_rx	rx_depth_irq <= rx_level

System signaling occurs when a level detection is true and enabled.

Table 533. DMA and interrupt request generation

System Signaling	Condition
irq	(irq_rx & rx_irq_enable)   (irq_tx & tx_irq_enable)
dmareq[0]	(dmareq_tx_1 & tx_dma1_enable)   (dmareq_rx_1 & rx_dma1_enable)
dmareq[1]	(dmareq_tx_2 & tx_dma2_enable)   (dmareq_rx_2 & rx_dma2_enable)

Table 534. Status feedback in the I2SSTATE register

Status Feedback	Status
irq	irq_rx   irq_tx
dmareq1	(dmareq_tx_1   dmareq_rx_1)
dmareq2	(dmareq_rx_2   dmareq_tx_2)



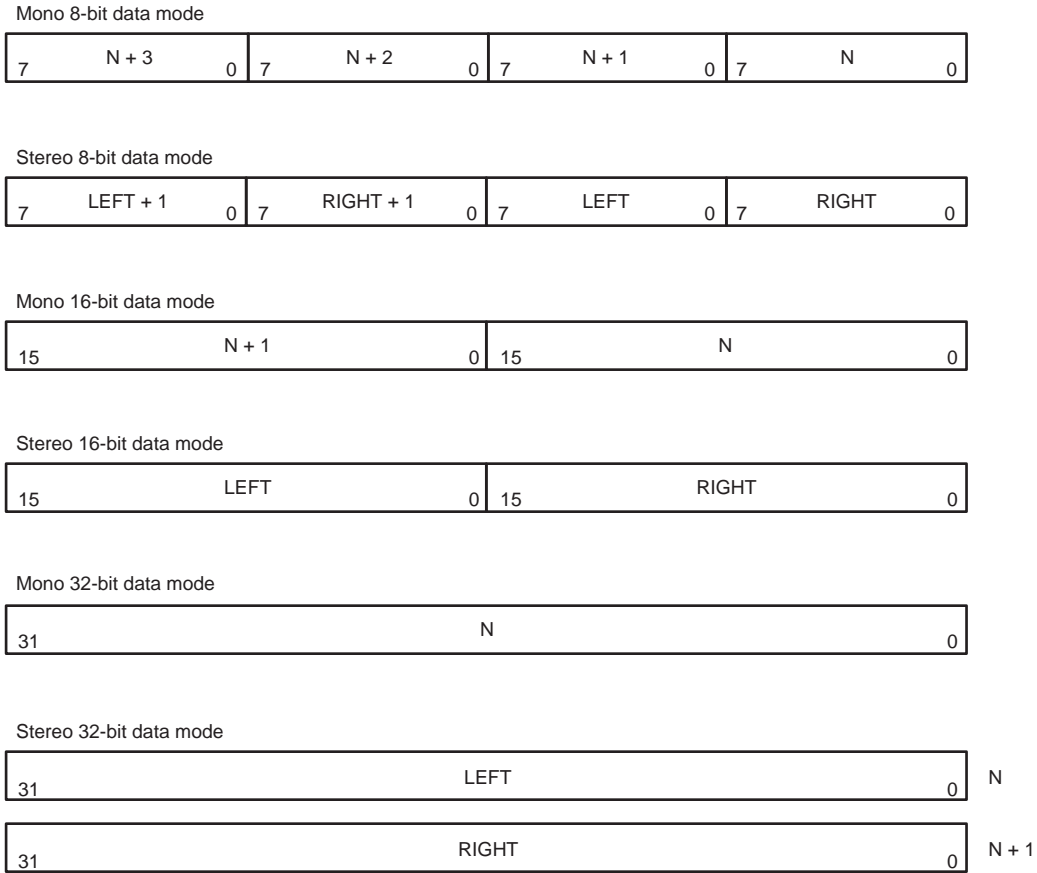


Fig 135. FIFO contents for various I<sup>2</sup>S modes

### 24.1 Basic configuration

The Timer 0, 1, 2, and 3 peripherals are configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bits PCTIM0/1/2/3.  
**Remark:** On reset, Timer0/1 are enabled (PCTIM0/1 = 1), and Timer2/3 are disabled (PCTIM2/3 = 0).
2. Peripheral clock: The timers operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select timer pins and pin modes through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: See register T0/1/2/3MCR ([Table 540](#)) and T0/1/2/3CCR ([Table 541](#)) for match and capture events. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
5. DMA: Up to two match conditions can be used to generate timed DMA requests, see [Table 655](#).

### 24.2 Features

**Remark:** The four Timer/Counters are identical except for the peripheral base address. A minimum of two Capture inputs and two Match outputs are pinned out for all four timers, with a choice of multiple pins for each. Timer 2 brings out all four Match outputs.

- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.
- Counter or Timer operation
- Up to two 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to four external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.

## 24.3 Applications

---

- Interval Timer for counting internal events.
- Pulse Width Demodulator via Capture inputs.
- Free running timer.

## 24.4 Description

---

The Timer/Counter is designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock, and can optionally generate interrupts or perform other actions at specified timer values, based on four match registers. It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

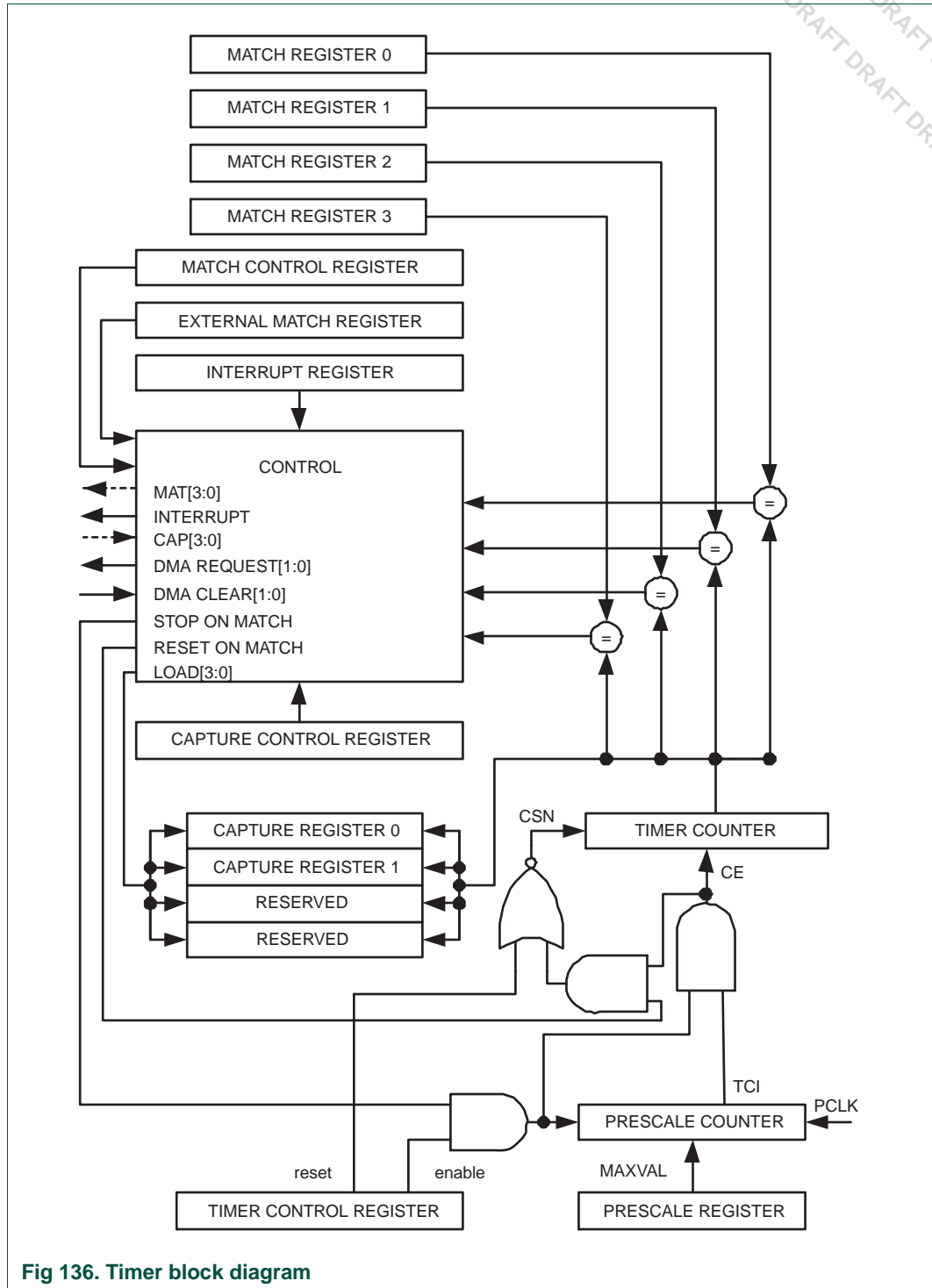


Fig 136. Timer block diagram

## 24.5 Pin description

[Table 535](#) gives a brief summary of each of the Timer/Counter related pins.

**Table 535. Timer/Counter pin description**

Pin	Type	Description
T0_CAP[1:0] T1_CAP[1:0] T2_CAP[1:0] T3_CAP[1:0]	Input	Capture Signals- A transition on a capture pin can be configured to load one of the Capture Registers with the value in the Timer Counter and optionally generate an interrupt. Capture functionality can be selected from a number of pins. When more than one pin is selected for a Capture input on a single TIMER0/1 channel, the pin with the lowest Port number is used  Timer/Counter block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see <a href="#">Section 24.6.3</a> .
T0_MAT[1:0] T1_MAT[1:0] T2_MAT[3:0] T3_MAT[1:0]	Output	External Match Output - When a match register (MR3:0) equals the timer counter (TC) this output can either toggle, go low, go high, or do nothing. The External Match Register (EMR) controls the functionality of this output. Match Output functionality can be selected on a number of pins in parallel.

### 24.5.1 Multiple CAP and MAT pins

Software can select from multiple pins for the CAP or MAT functions in the IOCON registers, which are described in [Section 8.4.1](#). When more than one pin is selected for a MAT output, all such pins are driven identically. When more than one pin is selected for a CAP input, the pin with the lowest Port number is used. Note that match conditions may be used internally without the use of a device pin.

## 24.6 Register description

Each Timer/Counter contains the registers shown in [Table 536](#) ("Reset Value" refers to the data stored in used bits only; it does not include reserved bits content). More detailed descriptions follow.

**Table 536. TIMER/COUNTER0-3 register map**

Generic Name	Description	Access	Reset Value <sup>[1]</sup>	TIMERN Register/ Name & Address	Reference
IR	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of the potential interrupt sources are pending.	R/W	0	T0IR - 0x4000 4000 T1IR - 0x4000 8000 T2IR - 0x4009 0000 T3IR - 0x4009 4000	<a href="#">24.6.1</a>
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0	T0TCR - 0x4000 4004 T1TCR - 0x4000 8004 T2TCR - 0x4009 0004 T3TCR - 0x4009 4004	<a href="#">24.6.2</a>
TC	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	R/W	0	T0TC - 0x4000 4008 T1TC - 0x4000 8008 T2TC - 0x4009 0008 T3TC - 0x4009 4008	<a href="#">24.6.4</a>
PR	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	R/W	0	T0PR - 0x4000 400C T1PR - 0x4000 800C T2PR - 0x4009 000C T3PR - 0x4009 400C	<a href="#">24.6.5</a>
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	R/W	0	T0PC - 0x4000 4010 T1PC - 0x4000 8010 T2PC - 0x4009 0010 T3PC - 0x4009 4010	<a href="#">24.6.6</a>
MCR	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0	T0MCR - 0x4000 4014 T1MCR - 0x4000 8014 T2MCR - 0x4009 0014 T3MCR - 0x4009 4014	<a href="#">24.6.8</a>
MR0	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	R/W	0	T0MR0 - 0x4000 4018 T1MR0 - 0x4000 8018 T2MR0 - 0x4009 0018 T3MR0 - 0x4009 4018	<a href="#">24.6.7</a>
MR1	Match Register 1. See MR0 description.	R/W	0	T0MR1 - 0x4000 401C T1MR1 - 0x4000 801C T2MR1 - 0x4009 001C T3MR1 - 0x4009 401C	<a href="#">24.6.7</a>
MR2	Match Register 2. See MR0 description.	R/W	0	T0MR2 - 0x4000 4020 T1MR2 - 0x4000 8020 T2MR2 - 0x4009 0020 T3MR2 - 0x4009 4020	<a href="#">24.6.7</a>
MR3	Match Register 3. See MR0 description.	R/W	0	T0MR3 - 0x4000 4024 T1MR3 - 0x4000 8024 T2MR3 - 0x4009 0024 T3MR3 - 0x4009 4024	<a href="#">24.6.7</a>

Table 536. TIMER/COUNTER0-3 register map

Generic Name	Description	Access	Reset Value <sup>[1]</sup>	TIMERN Register/ Name & Address	Reference
CCR	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.	R/W	0	T0CCR - 0x4000 4028 T1CCR - 0x4000 8028 T2CCR - 0x4009 0028 T3CCR - 0x4009 4028	<a href="#">24.6.10</a>
CR0	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the Tn_CAP0 input.	RO	0	T0CR0 - 0x4000 402C T1CR0 - 0x4000 802C T2CR0 - 0x4009 002C T3CR0 - 0x4009 402C	<a href="#">24.6.9</a>
CR1	Capture Register 1. CR1 is loaded with the value of TC when there is an event on the Tn_CAP1 input.	RO	0	T0CR1 - 0x4000 4030 T1CR1 - 0x4000 8030 T2CR1 - 0x4009 0030 T3CR1 - 0x4009 4030	<a href="#">24.6.9</a>
EMR	External Match Register. The EMR controls the external match pins Tn_MAT0-3.	R/W	0	T0EMR - 0x4000 403C T1EMR - 0x4000 803C T2EMR - 0x4009 003C T3EMR - 0x4009 403C	<a href="#">24.6.11</a>
CTCR	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	R/W	0	T0CTCR - 0x4000 4070 T1CTCR - 0x4000 8070 T2CTCR - 0x4009 0070 T3CTCR - 0x4009 4070	<a href="#">24.6.3</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 24.6.1 Interrupt Register (T[0/1/2/3]IR - 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000)

The Interrupt Register consists of 4 bits for the match interrupts and 4 bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect. The act of clearing an interrupt for a timer match also clears any corresponding DMA request.

Table 537. Interrupt Register (T[0/1/2/3]IR - addresses 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000) bit description

Bit	Symbol	Description	Reset Value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0
1	MR1 Interrupt	Interrupt flag for match channel 1.	0
2	MR2 Interrupt	Interrupt flag for match channel 2.	0
3	MR3 Interrupt	Interrupt flag for match channel 3.	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	-

### 24.6.2 Timer Control Register (T[0/1/2/3]CR - 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004)

The Timer Control Register (TCR) is used to control the operation of the Timer/Counter.

**Table 538. Timer Control Register (TCR, TIMERN: TnTCR - addresses 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004) bit description**

Bit	Symbol	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When 1, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0
31:2	-	Reserved. Read value is undefined, only zero should be written.	NA

### 24.6.3 Count Control Register (T[0/1/2/3]CTCR - 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070)

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting.

When Counter Mode is chosen as a mode of operation, the CAP input (selected by the CTCR bits 3:2) is sampled on every rising edge of the PCLK clock. After comparing two consecutive samples of this CAP input, one of the following four events is recognized: rising edge, falling edge, either of edges or no changes in the level of the selected CAP input. Only if the identified event occurs and the event corresponds to the one selected by bits 1:0 in the CTCR register, will the Timer Counter register be incremented.

Effective processing of the externally supplied clock to the counter has some limitations. Since two successive rising edges of the PCLK clock are used to identify only one edge on the CAP selected input, the frequency of the CAP input can not exceed one quarter of the PCLK clock. Consequently, duration of the high/low levels on the same CAP input in this case can not be shorter than 1/(2 PCLK).

**Table 539. Count Control Register (T[0/1/2/3]CTCR - addresses 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Counter/Timer Mode		This field selects which conditions can increment the Timer's Prescale Counter (PC), or clear the PC and increment the Timer Counter (TC). The prescaler is bypassed in counter mode.	00
		00	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register. The Prescale Counter is incremented on every rising PCLK edge.	
		01	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		10	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		11	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	



**Table 539. Count Control Register (T[0/1/2/3]CTCR - addresses 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) bit description**

Bit	Symbol	Value	Description	Reset Value
3:2	Count Input Select		When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking.	00
		00	CAPn.0 for TIMERN	
		01	CAPn.1 for TIMERN	
		10	Reserved	
		11	Reserved	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

**Note:** If Counter mode is selected for a particular CAPn input in the TnCTCR, the 3 bits for that input in the Capture Control Register (TnCCR) must be programmed as 000. However, capture and/or interrupt can be selected for the other 3 CAPn inputs in the same timer.

#### 24.6.4 Timer Counter registers (T0TC - T3TC, 0x4000 4008, 0x4000 8008, 0x4009 0008, 0x4009 4008)

The 32-bit Timer Counter register is incremented when the prescale counter reaches its terminal count. Unless it is reset before reaching its upper limit, the Timer Counter will count up through the value 0xFFFF FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a match register can be used to detect an overflow if needed.

#### 24.6.5 Prescale register (T0PR - T3PR, 0x4000 400C, 0x4000 800C, 0x4009 000C, 0x4009 400C)

The 32-bit Prescale register specifies the maximum value for the Prescale Counter.

#### 24.6.6 Prescale Counter register (T0PC - T3PC, 0x4000 4010, 0x4000 8010, 0x4009 0010, 0x4009 4010)

The 32-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale register, the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the Timer Counter to increment on every PCLK when PR = 0, every 2 pclks when PR = 1, etc.

### 24.6.7 Match Registers (MR0 - MR3)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

### 24.6.8 Match Control Register (T[0/1/2/3]MCR - 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter. The function of each of the bits is shown in [Table 540](#).

**Table 540. Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014) bit description**

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		0	This interrupt is disabled	
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.	0
		0	Feature disabled.	
5	MR1S	1	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.	0
		0	Feature disabled.	
6	MR2I	1	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.	0
		0	This interrupt is disabled	
7	MR2R	1	Reset on MR2: the TC will be reset if MR2 matches it.	0
		0	Feature disabled.	
8	MR2S	1	Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.	0
		0	Feature disabled.	
9	MR3I	1	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0
		0	This interrupt is disabled	
10	MR3R	1	Reset on MR3: the TC will be reset if MR3 matches it.	0
		0	Feature disabled.	
11	MR3S	1	Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.	0
		0	Feature disabled.	
31:12	-		Reserved. Read value is undefined, only zero should be written.	NA

### 24.6.9 Capture Registers (CR0 - CR1)

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin. The settings in the Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

### 24.6.10 Capture Control Register (T[0/1/2/3]CCR - 0x4000 4028, 0x4000 8028, 0x4009 0028, 0x4009 4028)

The Capture Control Register is used to control whether one of the four Capture Registers is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the description below, "n" represents the Timer number, 0 or 1.

Note: If Counter mode is selected for a particular CAP input in the CTCR, the 3 bits for that input in this register should be programmed as 000, but capture and/or interrupt can be selected for the other 3 CAP inputs.

**Table 541. Capture Control Register (T[0/1/2/3]CCR - addresses 0x4000 4028, 0x4000 8020, 0x4009 0028, 0x4009 4028) bit description**

Bit	Symbol	Value	Description	Reset Value
0	CAP0RE	1	Capture on CAPn.0 rising edge: a sequence of 0 then 1 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
1	CAP0FE	1	Capture on CAPn.0 falling edge: a sequence of 1 then 0 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
2	CAP0I	1	Interrupt on CAPn.0 event: a CR0 load due to a CAPn.0 event will generate an interrupt.	0
		0	This feature is disabled.	
3	CAP1RE	1	Capture on CAPn.1 rising edge: a sequence of 0 then 1 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
4	CAP1FE	1	Capture on CAPn.1 falling edge: a sequence of 1 then 0 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
5	CAP1I	1	Interrupt on CAPn.1 event: a CR1 load due to a CAPn.1 event will generate an interrupt.	0
		0	This feature is disabled.	
31:6	-		Reserved. Read value is undefined, only zero should be written.	NA

### 24.6.11 External Match Register (T[0/1/2/3]EMR - 0x4000 403C, 0x4000 803C, 0x4009 003C, 0x4009 403C)

The External Match Register provides both control and status of the external match pins. In the descriptions below, "n" represents the Timer number, 0 or 1, and "m" represent a Match number, 0 through 3.

Match events for Match 0 and Match 1 in each timer can cause a DMA request, see [Section 24.6.12](#).

**Table 542. External Match Register (T[0/1/2/3]EMR - addresses 0x4000 403C, 0x4000 803C, 0x4009 003C, 0x4009 403C) bit description**

Bit	Symbol	Description	Reset Value
0	EM0	External Match 0. When a match occurs between the TC and MR0, this bit can either toggle, go low, go high, or do nothing, depending on bits 5:4 of this register. This bit can be driven onto a MATn.0 pin, in a positive-logic manner (0 = low, 1 = high).	0
1	EM1	External Match 1. When a match occurs between the TC and MR1, this bit can either toggle, go low, go high, or do nothing, depending on bits 7:6 of this register. This bit can be driven onto a MATn.1 pin, in a positive-logic manner (0 = low, 1 = high).	0
2	EM2	External Match 2. When a match occurs between the TC and MR2, this bit can either toggle, go low, go high, or do nothing, depending on bits 9:8 of this register. This bit can be driven onto a MATn.2 pin, in a positive-logic manner (0 = low, 1 = high).	0
3	EM3	External Match 3. When a match occurs between the TC and MR3, this bit can either toggle, go low, go high, or do nothing, depending on bits 11:10 of this register. This bit can be driven onto a MATn.3 pin, in a positive-logic manner (0 = low, 1 = high).	0
5:4	EMC0	External Match Control 0. Determines the functionality of External Match 0. <a href="#">Table 543</a> shows the encoding of these bits.	00
7:6	EMC1	External Match Control 1. Determines the functionality of External Match 1. <a href="#">Table 543</a> shows the encoding of these bits.	00
9:8	EMC2	External Match Control 2. Determines the functionality of External Match 2. <a href="#">Table 543</a> shows the encoding of these bits.	00
11:10	EMC3	External Match Control 3. Determines the functionality of External Match 3. <a href="#">Table 543</a> shows the encoding of these bits.	00
15:12	-	Reserved. Read value is undefined, only zero should be written.	NA

**Table 543. External Match Control**

EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4]	Function
00	Do Nothing.
01	Clear the corresponding External Match bit/output to 0 (MATn.m pin is LOW if pinned out).
10	Set the corresponding External Match bit/output to 1 (MATn.m pin is HIGH if pinned out).
11	Toggle the corresponding External Match bit/output.

### 24.6.12 DMA operation

DMA requests are generated by a match of the Timer Counter (TC) register value to either Match Register 0 (MR0) or Match Register 1 (MR1). This is not connected to the operation of the Match outputs controlled by the EMR register. Each match sets a DMA request flag, which is connected to the DMA controller. In order to have an effect, the GPDMA must be configured and the relevant timer DMA request selected as a DMA source via the DMAREQSEL register, see [Section 33.5.15](#).

When a timer is initially set up to generate a DMA request, the request may already be asserted before a match condition occurs. An initial DMA request may be avoided by having software write a one to the interrupt flag location, as if clearing a timer interrupt. See [Section 24.6.1](#). A DMA request will be cleared automatically when it is acted upon by the GPDMA controller.

**Note:** because timer DMA requests are generated whenever the timer value is equal to the related Match Register value, DMA requests are always generated when the timer is running, unless the Match Register value is higher than the upper count limit of the timer. It is important not to select and enable timer DMA requests in the GPDMA block unless the timer is correctly configured to generate valid DMA requests.

## 24.7 Example timer operation

Figure 137 shows a timer configured to reset the count and generate an interrupt on match. The prescaler is set to 2 and the match register set to 6. At the end of the timer cycle where the match occurs, the timer count is reset. This gives a full length cycle to the match value. The interrupt indicating that a match occurred is generated in the next clock after the timer reached the match value.

Figure 138 shows a timer configured to stop and generate an interrupt on match. The prescaler is again set to 2 and the match register set to 6. In the next clock after the timer reaches the match value, the timer enable bit in TCR is cleared, and the interrupt indicating that a match occurred is generated.

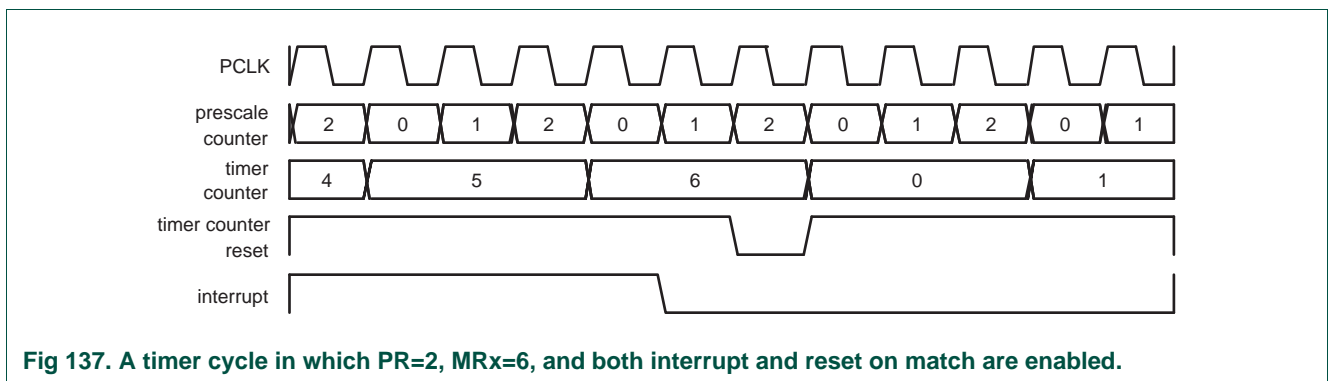


Fig 137. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled.

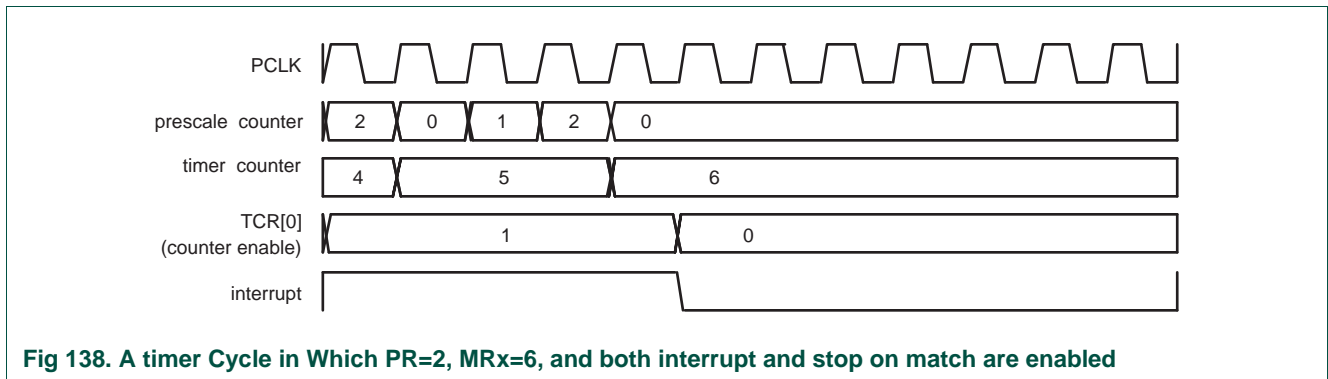


Fig 138. A timer Cycle in Which PR=2, MRx=6, and both interrupt and stop on match are enabled

### 25.1 Basic configuration

---

The System Tick Timer is configured using the following registers:

1. Clock Source: Select either the internal CCLK or external STCLK (pin P3[26]) clock as the source in the STCTRL register.
2. Pins: If STCLK (pin P3[26]) was selected as clock source enable the STCLK pin function in the relevant IOCON register ([Section 8.4.1](#)).
3. Interrupt: The System Tick Timer Interrupt is enabled in the NVIC using the appropriate Interrupt Set Enable register. The SysTick interrupt is hard-wired within the Cortex-M3 as exception 15. Refer to the NVIC chapter ([Section 6.1](#)) and the Cortex-M3 User Guide appended to this manual ([Section 38.4.2](#))

### 25.2 Features

---

- Times intervals of 10 milliseconds
- Dedicated exception vector
- Can be clocked internally by the CPU clock or by a clock input from a pin (STCLK)

### 3. Description

---

The System Tick Timer is an integral part of the Cortex-M3. The System Tick Timer is intended to generate a fixed 10 millisecond interrupt for use by an operating system or other system management software.

Since the System Tick Timer is a part of the Cortex-M3, it facilitates porting of software by providing a standard timer that is available on Cortex-M3 based devices.

Refer to the Cortex-M3 User Guide appended to this manual ([Section 38.4.4](#)) for details of System Tick Timer operation.

### 25.4 Operation

---

The System Tick Timer is a 24-bit timer that counts down to zero and generates an interrupt. The intent is to provide a fixed 10 millisecond time interval between interrupts. The System Tick Timer may be clocked either from the CPU clock or from the external pin STCLK. The STCLK function shares pin P3[26] with other functions, and must be selected for use as the System Tick Timer clock. In order to generate recurring interrupts at a specific interval, the STRELOAD register must be initialized with the correct value for the desired interval. A default value is provided in the STCALIB register and may be changed by software. The default value gives a 10 millisecond interrupt rate if the CPU clock is set to 100 MHz.

The block diagram of the System Tick Timer is shown below in the [Figure 139](#).

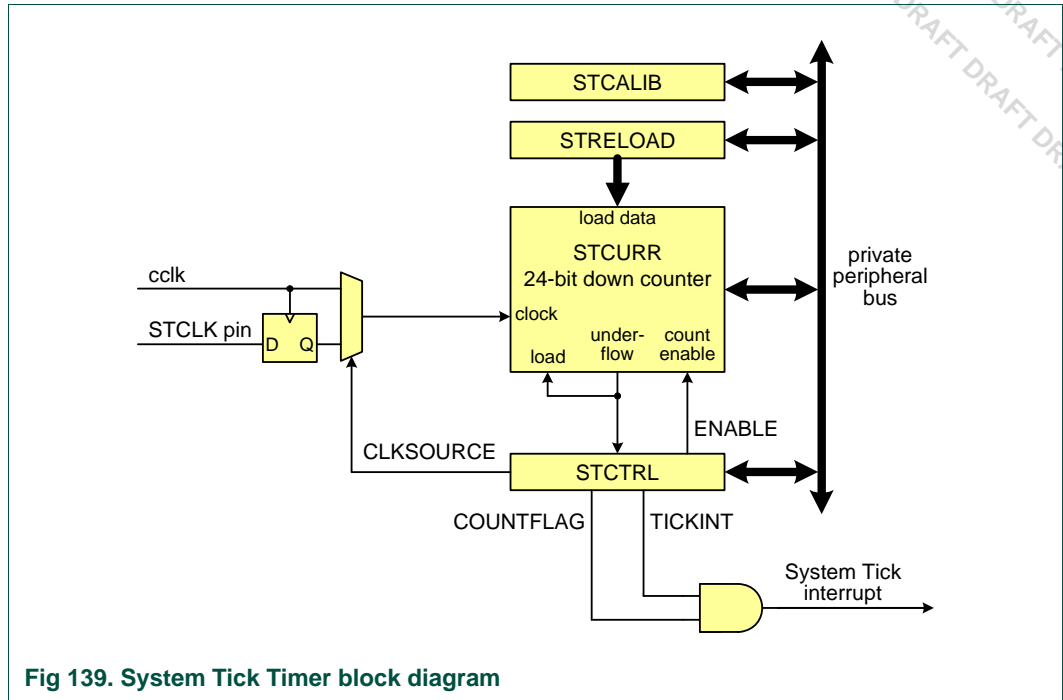


Fig 139. System Tick Timer block diagram

## 25.5 Register description

Table 544. System Tick Timer register map

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
STCTRL	System Timer Control and status register	R/W	0x4	0xE000 E010	<a href="#">545</a>
STRELOAD	System Timer Reload value register	R/W	0	0xE000 E014	<a href="#">546</a>
STCURRE	System Timer Current value register	R/W	0	0xE000 E018	<a href="#">547</a>
STCALIB	System Timer Calibration value register	R/W	0x000F 423F	0xE000 E01C	<a href="#">548</a>

[1] Reset Value reflects the data stored in used bits only. It does not include content of reserved bits.

### 25.5.1 System Timer Control and status register (STCTRL - 0xE000 E010)

The STCTRL register contains control information for the System Tick Timer, and provides a status flag.

Table 545. System Timer Control and status register (STCTRL - 0xE000 E010) bit description

Bit	Symbol	Description	Reset value
0	ENABLE	System Tick counter enable. When 1, the counter is enabled. When 0, the counter is disabled.	0
1	TICKINT	System Tick interrupt enable. When 1, the System Tick interrupt is enabled. When 0, the System Tick interrupt is disabled. When enabled, the interrupt is generated when the System Tick counter counts down to 0.	0
2	CLKSOURCE	System Tick clock source selection. When 1, the CPU clock is selected. When 0, the external clock pin (STCLK) is selected.	1



**Table 545. System Timer Control and status register (STCTRL - 0xE000 E010) bit description**

Bit	Symbol	Description	Reset value
15:3	-	Reserved. Read value is undefined, only zero should be written.	NA
16	COUNTFLAG	System Tick counter flag. This flag is set when the System Tick counter counts down to 0, and is cleared by reading this register.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

### 25.5.2 System Timer Reload value register (STRELOAD - 0xE000 E014)

The STRELOAD register is set to the value that will be loaded into the System Tick Timer whenever it counts down to zero. This register is loaded by software as part of timer initialization. The STCALIB register may be read and used as the value for STRELOAD if the CPU or external clock is running at the frequency intended for use with the STCALIB value.

**Table 546. System Timer Reload value register (STRELOAD - 0xE000 E014) bit description**

Bit	Symbol	Description	Reset value
23:0	RELOAD	This is the value that is loaded into the System Tick counter when it counts down to 0.	0
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 25.5.3 System Timer Current value register (STCURRE - 0xE000 E018)

The STCURRE register returns the current count from the System Tick counter when it is read by software.

**Table 547. System Timer Current value register (STCURRE - 0xE000 E018) bit description**

Bit	Symbol	Description	Reset value
23:0	CURRENT	Reading this register returns the current value of the System Tick counter. Writing any value clears the System Tick counter and the COUNTFLAG bit in STCTRL.	0
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 25.5.4 System Timer Calibration value register (STCALIB - 0xE000 E01C)

The STCALIB register contains a value that is initialized by the Boot Code to a factory programmed value that is appropriate for generating an interrupt every 10 milliseconds if the System Tick Timer is clocked at a frequency of 100 MHz. This is the intended use of the System Tick Timer by ARM. It can be used to generate interrupts at other frequencies by selecting the correct reload value.

**Table 548. System Timer Calibration value register (STCALIB - 0xE000 E01C) bit description**

Bit	Symbol	Description	Reset value
23:0	TENMS	Reload value to get a 10 millisecond System Tick underflow rate when running at 100 MHz. This value initialized at reset with a factory supplied value selected for the LPC178x/177x. The provided values of TENMS, SKEW, and NOREF are applicable only when using a CPU clock or external STCLK source of 100 MHz.	0x0F 423F
29:24	-	Reserved. Read value is undefined, only zero should be written.	NA
30	SKEW	Indicates whether the TENMS value will generate a precise 10 millisecond time, or an approximation. This bit is initialized at reset with a factory supplied value selected for the LPC178x/177x. See the description of TENMS above.  When 0, the value of TENMS is considered to be precise. When 1, the value of TENMS is not considered to be precise.	0
31	NOREF	Indicates whether an external reference clock is available. This bit is initialized at reset with a factory supplied value selected for the LPC178x/177x. See the description of TENMS above.  When 0, a separate reference clock is available. When 1, a separate reference clock is not available.	0

## 25.6 Example timer calculations

The following examples illustrate selecting System Tick Timer values for different system configurations. All of the examples calculate an interrupt interval of 10 milliseconds, as the System Tick Timer is intended to be used.

### Example 1)

This example is for the System Tick Timer running from the CPU clock (cclk), which is 100 MHz.

STCTRL = 7. This enables the timer and its interrupt, and selects cclk as the clock source.

$$\text{RELOAD} = (\text{cclk} / 100) - 1 = 1,000,000 - 1 = 999,999 = 0xF423F$$

In this case, there is no rounding error, so the result is as accurate as cclk.

### Example 2)

This example is for the System Tick Timer running from the CPU clock (cclk), which is 80 MHz.

STCTRL = 7. This enables the timer and its interrupt, and selects cclk as the clock source.

$$\text{RELOAD} = (\text{cclk} / 100) - 1 = 800,000 - 1 = 799,999 = 0xC34FF$$

In this case, there is no rounding error, so the result is as accurate as cclk.

### Example 3)

This example is for the CPU clock (cclk) is taken from the Internal RC Oscillator (IRC), factory trimmed to 4 MHz.

STCTRL = 7. This enables the timer and its interrupt, and selects cclk as the clock source.

$$\text{RELOAD} = (F_{\text{IRC}} / 100) - 1 = 40,000 - 1 = 39,999 = 0x9C3F$$

In this case, there is no rounding error, so the result is as accurate as the IRC.

### Example 4)

This example is for the System Tick Timer running from an external clock source (the STCLK pin), which in this case happens to be 32.768 kHz.

STCTRL = 3. This enables the timer and its interrupt, and selects the STCLK pin as the clock source. STCLK must be selected as the function of the relevant pin. See [Section 8.4.1](#).

$$\text{RELOAD} = (\text{cclk} / 100) - 1 = 327.6 - 1 = 327 \text{ (rounded up)} = 0x0147$$

In this case, there is rounding error, so the interrupt rate will drift slightly relative to the input frequency.

### 26.1 Basic configuration

The PWM is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCPWM1.  
**Remark:** On reset, PWM1 is enabled (PCPWM1 = 1) and PWM0 is disabled (PCPWM1 = 0).
2. Peripheral clock: The PWMs operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select PWM pins and pin modes for port pins with PWM functions through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: See registers PWMMCR ([Table 555](#)) and PWMCCR ([Table 556](#)) for match and capture events. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 26.2 Features

- The two PWMs have the same operational features. The PWMs may be operated in a synchronized fashion by setting them both up to run at the same rate, then enabling both simultaneously. PWM0 acts as the Master and PWM1 as the slave for this use.
- Counter or Timer operation (may use the peripheral clock or one of the capture inputs as the clock source).
- Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types. The match registers also allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Supports single edge controlled and/or double edge controlled PWM outputs. Single edge controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low. Double edge controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.
- Pulse period and width can be any number of timer counts. This allows complete flexibility in the trade-off between resolution and repetition rate. All PWM outputs will occur at the same repetition rate.
- Double edge controlled PWM outputs can be programmed to be either positive going or negative going pulses.
- Match register updates are synchronized with pulse outputs to prevent generation of erroneous pulses. Software must "release" new match values before they can become effective.
- May be used as a standard timer if the PWM mode is not enabled.
- A 32 bit Timer/Counter with a programmable 32 bit Prescaler.

- A transition on a capture input signal can trigger a snapshot of the 32-bit timer value. A capture event may also optionally generate an interrupt.

## 26.3 Description

The PWM function is based on the standard Timer block and inherits all of its features, although many timer functions are not brought out to package pins. The Timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers. It also includes capture inputs to save the timer value when an input signal transitions, and optionally generate an interrupt when those events occur. The PWM function is in addition to these features, and is based on match register events.

The ability to separately control rising and falling edge locations allows the PWM to be used for more applications. For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and positions.

Two match registers can be used to provide a single edge controlled PWM output. One match register (MR0) controls the PWM cycle rate, by resetting the count upon match. The other match register controls the PWM edge position. Additional single edge controlled PWM outputs require only one match register each, since the repetition rate is the same for all PWM outputs. Multiple single edge controlled PWM outputs will all have a rising edge at the beginning of each PWM cycle, when an MR0 match occurs.

Three match registers can be used to provide a PWM output with both edges controlled. Again, the MR0 match register controls the PWM cycle rate. The other match registers control the two PWM edge positions. Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs.

With double edge controlled PWM outputs, specific match registers control the rising and falling edge of the output. This allows both positive going PWM pulses (when the rising edge occurs prior to the falling edge), and negative going PWM pulses (when the falling edge occurs prior to the rising edge).

[Figure 140](#) shows the block diagram of the PWM. The portions that have been added to the standard timer block are on the right hand side and at the top of the diagram. At the lower left of the diagram may be found the Master Enable output from the Timer Control register that allows the Master PWM (PWM0) to enable both itself and the Slave PWM (PWM1) at the same time, if desired. The Master Enable output from PWM0 is connected to the external enable input of both PWM blocks.

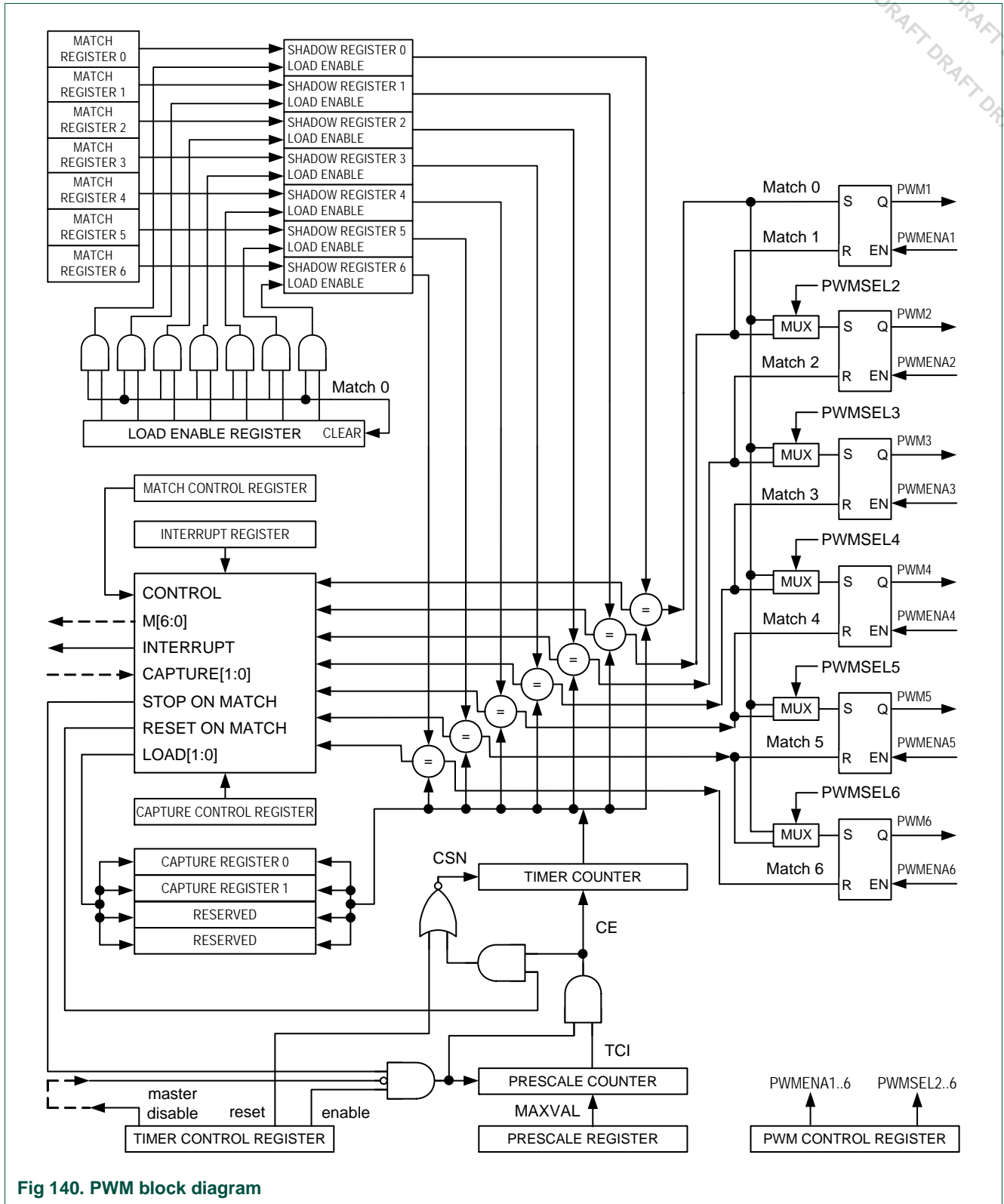
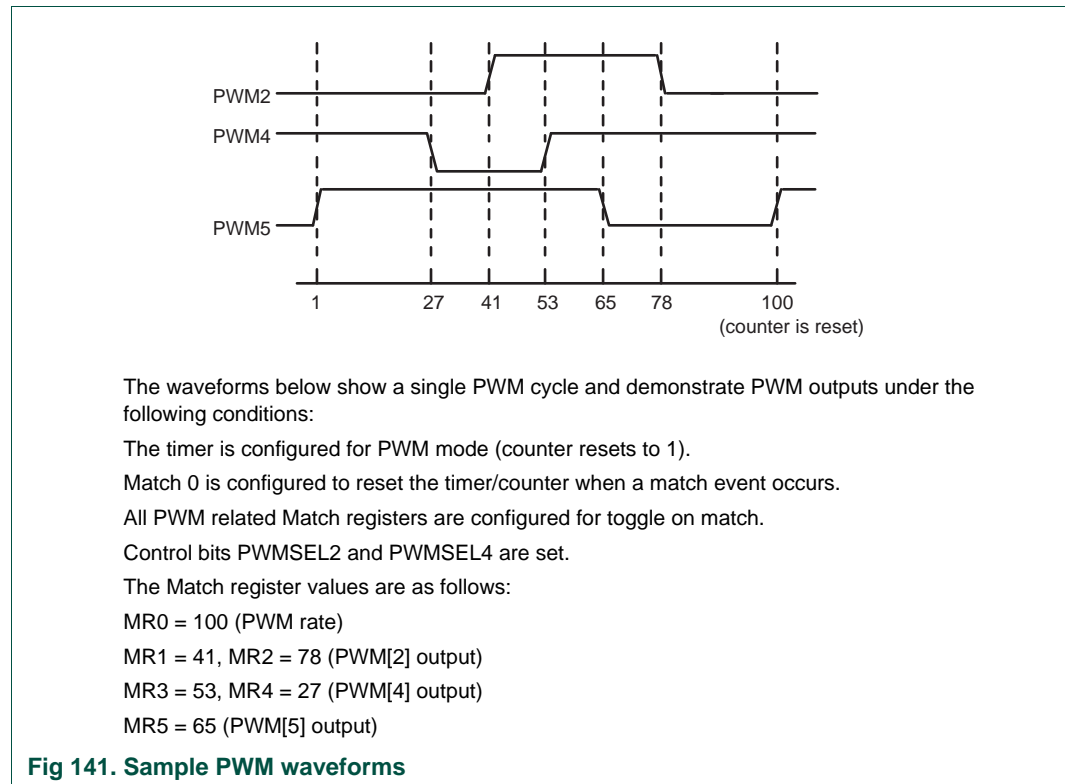


Fig 140. PWM block diagram

## 26.4 Sample waveform with rules for single and double edge control

A sample of how PWM values relate to waveform outputs is shown in [Figure 141](#). PWM output logic is shown in [Figure 140](#) that allows selection of either single or double edge controlled PWM outputs via the muxes controlled by the PWMSELn bits. The match register selections for various PWM outputs is shown in [Table 549](#). This implementation supports up to N-1 single edge PWM outputs or (N-1)/2 double edge PWM outputs, where N is the number of match registers and outputs that are implemented. PWM types can be mixed if desired.



**Table 549. Set and reset inputs for PWM Flip-Flops**

PWM Channel	Single Edge PWM (PWMSELn = 0)		Double Edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0 <sup>[1]</sup>	Match 1 <sup>[1]</sup>
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2 <sup>[2]</sup>	Match 3 <sup>[2]</sup>
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4 <sup>[2]</sup>	Match 5 <sup>[2]</sup>
6	Match 0	Match 6	Match 5	Match 6

[1] Identical to single edge mode in this case since Match 0 is the neighboring match register. Essentially, PWM1 cannot be a double edged output.

[2] It is generally not advantageous to use PWM channels 3 and 5 for double edge PWM outputs because it would reduce the number of double edge PWM outputs that are possible. Using PWM[2], PWM[4], and PWM[6] for double edge PWM outputs provides the most pairings.

### 26.4.1 Rules for Single Edge Controlled PWM Outputs

1. All single edge controlled PWM outputs go high at the beginning of a PWM cycle unless their match value is equal to 0.
2. Each PWM output will go low when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM rate), the PWM output remains continuously high.

### 26.4.2 Rules for Double Edge Controlled PWM Outputs

Five rules are used to determine the next value of a PWM output when a new cycle is about to begin:

1. The match values for the **next** PWM cycle are used at the end of a PWM cycle (a time point which is coincident with the beginning of the next PWM cycle), except as noted in rule 3.
2. A match value equal to 0 or the current PWM rate (the same as the Match channel 0 value) have the same effect, except as noted in rule 3. For example, a request for a falling edge at the beginning of the PWM cycle has the same effect as a request for a falling edge at the end of a PWM cycle.
3. When match values are changing, if one of the "old" match values is equal to the PWM rate, it is used again once if the neither of the new match values are equal to 0 or the PWM rate, and there was no old match value equal to 0.
4. If both a set and a clear of a PWM output are requested at the same time, clear takes precedence. This can occur when the set and clear match values are the same as in, or when the set or clear value equals 0 and the other value equals the PWM rate.
5. If a match value is out of range (i.e. greater than the PWM rate value), no match event occurs and that match channel has no effect on the output. This means that the PWM output will remain always in one state, allowing always low, always high, or "no change" outputs.

## 26.5 Pin description

[Table 550](#) gives a brief summary of each of PWM related pins.

**Table 550. Pin summary**

Pin	Type	Description
PWM0[6:1]	Output	Outputs from PWM0 channels 6 to 1.
PWM0_CAP0	Input	Capture input for PWM0. A transition on the capture pin can be configured to load the corresponding Capture register with the value of the Timer Counter and optionally generate an interrupt.
PWM1[6:1]	Output	Outputs from PWM1 channels 6 to 1.
PWM1_CAP1[1:0]	Input	Capture inputs for PWM1. A transition on the capture pin can be configured to load the corresponding Capture register with the value of the Timer Counter and optionally generate an interrupt.



## 26.6 Register description

Table 551. PWM0 and PWM1 register map

Generic Name	Description	Access	Reset Value <sup>[1]</sup>	PWMn Register/ Name & Address	Reference
IR	Interrupt Register. The IR can be written to clear interrupts, or read to identify which PWM interrupt sources are pending.	R/W	0	PWM0IR - 0x4001 4000 PWM1IR - 0x4001 8000	<a href="#">26.6.1</a>
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions.	R/W	0	PWM0TCR - 0x4001 4004 PWM1TCR 0x4001 8004	<a href="#">26.6.2</a>
CTCR	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	R/W	0	PWM0CTCR - 0x4001 4070 PWM1CTCR 0x4001 8070	<a href="#">26.6.3</a>
TC	Timer Counter. The 32 bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	R/W	0	PWM0TC 0x4001 4008 PWM1TC 0x4001 8008	<a href="#">26.6.4</a>
PR	Prescale Register. Determines how often the PWM counter is incremented.	R/W	0	PWM0PR 0x4001 400C PWM1PR 0x4001 800C	<a href="#">26.6.5</a>
PC	Prescale Counter. Prescaler for the main PWM counter.	R/W	0	PWM0PC 0x4001 4010 PWM1PC 0x4001 8010	<a href="#">26.6.6</a>
MCR	Match Control Register. The MCR is used to control whether an interrupt is generated and if the PWM counter is reset when a Match occurs.	R/W	0	PWM0MCR 0x4001 4014 PWM1MCR 0x4001 8014	<a href="#">26.6.7</a>
MR0	Match Register 0. Match registers are continuously compared to the PWM counter in order to control PWM output edges.	R/W	0	PWM0MR0 0x4001 4018 PWM1MR0 0x4001 8018	<a href="#">26.6.8</a>
MR1	Match Register 1. See MR0 description.	R/W	0	PWM0MR1 0x4001 401C PWM1MR1 0x4001 801C	<a href="#">26.6.8</a>
MR2	Match Register 2. See MR0 description.	R/W	0	PWM0MR2 0x4001 4020 PWM1MR2 0x4001 8020	<a href="#">26.6.8</a>
MR3	Match Register 3. See MR0 description.	R/W	0	PWM0MR3 0x4001 4024 PWM1MR3 0x4001 8024	<a href="#">26.6.8</a>
MR4	Match Register 4. See MR0 description.	R/W	0	PWM0MR 0x4001 4040 PWM1MR 0x4001 8040	<a href="#">26.6.8</a>
MR5	Match Register 5. See MR0 description.	R/W	0	PWM0MR 0x4001 4044 PWM1MR 0x4001 8044	<a href="#">26.6.8</a>
MR6	Match Register 6. See MR0 description.	R/W	0	PWM0MR 0x4001 4048 PWM1MR 0x4001 8048	<a href="#">26.6.8</a>
CCR	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated for a capture event.	R/W	0	PWM0CCR 0x4001 4028 PWM1CCR 0x4001 8028	<a href="#">26.6.9</a>
CR0	Capture Register 0. CR0 of PWMn is loaded with the value of the TC when there is an event on the PWMn_CAP[0] input.	RO	0	PWM0CR0 0x4001 402C PWM1CR0 0x4001 802C	<a href="#">26.6.10</a>

Table 551. PWM0 and PWM1 register map

Generic Name	Description	Access	Reset Value <sup>[1]</sup>	PWMn Register/ Name & Address	Reference
CR1	Capture Register 1. See CR0 description.	RO	0	PWM1CR1 0x4001 8030	<a href="#">26.6.10</a>
PCR	PWM Control Register. Enables PWM outputs and selects either single edge or double edge controlled PWM outputs.	R/W	0	PWM0PCR 0x4001 404C PWM1PCR 0x4001 804C	<a href="#">26.6.11</a>
LER	Load Enable Register. Enables use of updated PWM match values.	R/W	0	PWM0LER 0x4001 4050 PWM1LER 0x4001 8050	<a href="#">26.6.12</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 26.6.1 PWM Interrupt Register (PWM0IR - 0x4001 4000 and PWM1IR - 0x4001 8000)

The PWM Interrupt register consists of eleven bits ([Table 552](#)), seven for the match interrupts and four reserved. If an interrupt is generated then the corresponding bit in the PWMIR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 552: PWM Interrupt Register (PWM0IR - 0x4001 4000 and PWM1IR - 0x4001 8000) bit description

Bit	Symbol	Description	Reset Value
0	PWMMR0 Interrupt	Interrupt flag for PWM match channel 0.	0
1	PWMMR1 Interrupt	Interrupt flag for PWM match channel 1.	0
2	PWMMR2 Interrupt	Interrupt flag for PWM match channel 2.	0
3	PWMMR3 Interrupt	Interrupt flag for PWM match channel 3.	0
4	PWMCAP0 Interrupt	Interrupt flag for capture input 0	0
5	PWMCAP1 Interrupt	Interrupt flag for capture input 1 (available in PWM1IR only; this bit is reserved in PWM0IR).	0
7:6	-	Reserved. Read value is undefined, only zero should be written.	-
8	PWMMR4 Interrupt	Interrupt flag for PWM match channel 4.	0
9	PWMMR5 Interrupt	Interrupt flag for PWM match channel 5.	0
10	PWMMR6 Interrupt	Interrupt flag for PWM match channel 6.	0
15:11	-	Reserved. Read value is undefined, only zero should be written.	NA

### 26.6.2 PWM Timer Control Register (PWM0TCR - 0x4001 4004 and PWM1TCR - 0x4001 8004)

The PWM Timer Control Register (PWMTCR) is used to control the operation of the PWM Timer Counter. The function of each of the bits is shown in [Table 553](#).

Table 553: PWM Timer Control Register (PWM0TCR - 0x4001 4004 PWM1TCR - 0x4001 8004) bit description

Bit	Symbol	Value	Description	Reset Value
0	Counter Enable	1	The PWM Timer Counter and PWM Prescale Counter are enabled for counting.	0
		0	The counters are disabled.	

**Table 553: PWM Timer Control Register (PWM0TCR - 0x4001 4004 PWM1TCR - 0x4001 8004) bit description**

Bit	Symbol	Value	Description	Reset Value
1	Counter Reset	1	The PWM Timer Counter and the PWM Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until this bit is returned to zero.	0
		0	Clear reset.	
2	-		Reserved. Read value is undefined, only zero should be written.	NA
3	PWM Enable	1	PWM mode is enabled (counter resets to 1). PWM mode causes the shadow registers to operate in connection with the Match registers. A program write to a Match register will not have an effect on the Match result until the corresponding bit in PWMLER has been set, followed by the occurrence of a PWM Match 0 event. Note that the PWM Match register that determines the PWM rate (PWM Match Register 0 - MR0) must be set up prior to the PWM being enabled. Otherwise a Match event will not occur to cause shadow register contents to become effective.	0
		0	Timer mode is enabled (counter resets to 0).	
4	Master Disable (PWM0 only)		The two PWMs may be synchronized using the Master Disable control bit. The Master disable bit of the Master PWM (PWM0 module) controls a secondary enable input to both PWMs, as shown in <a href="#">Figure 140</a> . This bit has no function in the Slave PWM (PWM1).	0
		1	PWM0 is the master, and both PWMs are enabled for counting.	
		0	The PWMs are used independently, and the individual Counter Enable bits are used to control the PWMs.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 26.6.3 PWM Count Control Register (PWM0CTCR - 0x4001 4070 and PWM1CTCR - 0x4001 8070)

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting. The function of each of the bits is shown in [Table 554](#).

**Table 554: PWM Count control Register (PWM0CTCR - 0x4001 4070 and PWM1CTCR - 0x4001 8070) bit description**

Bit	Symbol	Description	Reset Value
1:0	Counter/Timer Mode	00: Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale register. 01: Counter Mode: the TC is incremented on rising edges of the PWM_CAP input selected by bits 3:2. 10: Counter Mode: the TC is incremented on falling edges of the PWM_CAP input selected by bits 3:2. 11: Counter Mode: the TC is incremented on both edges of the PWM_CAP input selected by bits 3:2.	00
3:2	Count Input Select	When bits 1:0 are not 00, these bits select which PWM_CAP pin carries the signal used to increment the TC. For PWM0: 00 = PWM0_CAP[0] (Other combinations are reserved) For PWM1: 00 = PWM1_CAP[0], 01 = PWM1_CAP[1] (Other combinations are reserved)	00
7:4	-	Reserved. Read value is undefined, only zero should be written.	NA

[1] PWM\_CAP input signal frequency must not exceed PCLK/4. When the PWM clock is supplied via the PWM\_CAP pin, at no time can a high or low level of the signal on this pin can last less than  $1/(2 \times PCLK)$ .

### 26.6.4 PWM Timer Counter (PWM0TC - 0x4001 4008 and PWM1TC - 0x4001 8008)

The 32-bit PWM Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the PWMTC will count up through the value 0xFFFF FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

### 26.6.5 PWM Prescale Register (PWM0PR - 0x4001 400C and PWM1PR - 0x4001 800C)

The 32-bit PWM Prescale Register specifies the maximum value for the PWM Prescale Counter.

### 26.6.6 PWM Prescale Counter Register (PWM0PC - 0x4001 4010 and PWM1PC - 0x4001 8010)

The 32-bit PWM Prescale Counter controls division of PCLK by some constant value before it is applied to the PWM Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The PWM Prescale Counter is incremented on every PCLK. When it reaches the value stored in the PWM Prescale Register, the PWM Timer Counter is incremented and the PWM Prescale Counter is reset on the next PCLK. This causes the PWM TC to increment on every PCLK when PWMPR = 0, every 2 PCLKs when PWMPR = 1, etc.

### 26.6.7 PWM Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014)

The PWM Match Control registers are used to control what operations are performed when one of the PWM Match registers matches the PWM Timer Counter. The function of each of the bits is shown in [Table 555](#).

**Table 555: Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014) bit description**

Bit	Symbol	Value	Description	Reset Value
0	PWMMR0I	1	Interrupt on PWMMR0: an interrupt is generated when PWMMR0 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
1	PWMMR0R	1	Reset on PWMMR0: the PWMTC will be reset if PWMMR0 matches it.	0
		0	This feature is disabled.	
2	PWMMR0S	1	Stop on PWMMR0: the PWMTC and PWMPC will be stopped and PWMTCR bit 0 will be set to 0 if PWMMR0 matches the PWMTC.	0
		0	This feature is disabled	
3	PWMMR1I	1	Interrupt on PWMMR1: an interrupt is generated when PWMMR1 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	

Table 555: Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014) bit description

Bit	Symbol	Value	Description	Reset Value
4	PWMMR1R	1	Reset on PWMMR1: the PWMTC will be reset if PWMMR1 matches it.	0
		0	This feature is disabled.	
5	PWMMR1S	1	Stop on PWMMR1: the PWMTC and PWMPC will be stopped and PWMTCR bit 0 will be set to 0 if PWMMR1 matches the PWMTC.	0
		0	This feature is disabled.	
6	PWMMR2I	1	Interrupt on PWMMR2: an interrupt is generated when PWMMR2 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
7	PWMMR2R	1	Reset on PWMMR2: the PWMTC will be reset if PWMMR2 matches it.	0
		0	This feature is disabled.	
8	PWMMR2S	1	Stop on PWMMR2: the PWMTC and PWMPC will be stopped and PWMTCR bit 0 will be set to 0 if PWMMR2 matches the PWMTC.	0
		0	This feature is disabled	
9	PWMMR3I	1	Interrupt on PWMMR3: an interrupt is generated when PWMMR3 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
10	PWMMR3R	1	Reset on PWMMR3: the PWMTC will be reset if PWMMR3 matches it.	0
		0	This feature is disabled	
11	PWMMR3S	1	Stop on PWMMR3: The PWMTC and PWMPC will be stopped and PWMTCR bit 0 will be set to 0 if PWMMR3 matches the PWMTC.	0
		0	This feature is disabled	
12	PWMMR4I	1	Interrupt on PWMMR4: An interrupt is generated when PWMMR4 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
13	PWMMR4R	1	Reset on PWMMR4: the PWMTC will be reset if PWMMR4 matches it.	0
		0	This feature is disabled.	
14	PWMMR4S	1	Stop on PWMMR4: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR4 matches the PWMTC.	0
		0	This feature is disabled	
15	PWMMR5I	1	Interrupt on PWMMR5: An interrupt is generated when PWMMR5 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
16	PWMMR5R	1	Reset on PWMMR5: the PWMTC will be reset if PWMMR5 matches it.	0
		0	This feature is disabled.	
17	PWMMR5S	1	Stop on PWMMR5: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR5 matches the PWMTC.	0
		0	This feature is disabled	
18	PWMMR6I	1	Interrupt on PWMMR6: an interrupt is generated when PWMMR6 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
19	PWMMR6R	1	Reset on PWMMR6: the PWMTC will be reset if PWMMR6 matches it.	0
		0	This feature is disabled.	

**Table 555: Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014) bit description**

Bit	Symbol	Value	Description	Reset Value
20	PWMMR6S	1	Stop on PWMMR6: the PWMTTC and PWMPC will be stopped and PWMTTCR[0] will be set to 0 if PWMMR6 matches the PWMTTC.	0
		0	This feature is disabled	
31:21	-		Reserved. Read value is undefined, only zero should be written.	NA

### 26.6.8 PWM Match Registers (PWM0MR0 to PWM0MR6 and PWM1MR0 to PWM1MR6)

The 32-bit PWM Match register values are continuously compared to the PWM Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the PWM Timer Counter, or stop the timer. Actions are controlled by the settings in the PWMMCR register.

### 26.6.9 PWM Capture Control Register (PWM0CCR - 0x4001 4028 and PWM1CCR - 0x4001 8028)

The Capture Control register is used to control whether any of the Capture registers is loaded with the value in the Timer Counter when a capture event occurs on PWM0\_CAP[0] or PWM1\_CAP[1:0], and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the descriptions below, “n” represents the Timer number, 0 or 1.

**Note:** If Counter mode is selected for a particular PWM\_CAP input in the CTCR, the 3 bits for that input in this register should be programmed as 000, but capture and/or interrupt can be selected for the other two PWM\_CAP inputs.

**Table 556: PWM Capture Control Register (PWM0CCR - 0x4001 4028 and PWM1CCR - 0x4001 8028) bit description**

Bit	Symbol	Value	Description	Reset Value
0	Capture on PWMn_CAP[0] rising edge	0	This feature is disabled.	0
		1	A synchronously sampled rising edge on PWMn_CAP[0] will cause CR0 to be loaded with the contents of the TC.	
1	Capture on PWMn_CAP[0] falling edge	0	This feature is disabled.	0
		1	A synchronously sampled falling edge on PWMn_CAP[0] will cause CR0 to be loaded with the contents of TC.	
2	Interrupt on PWMn_CAP[0] event	0	This feature is disabled.	0
		1	A CR0 load due to a PWMn_CAP[0] event will generate an interrupt.	
3	Capture on PWMn_CAP[1] rising edge <sup>[1]</sup>	0	This feature is disabled.	0
		1	A synchronously sampled rising edge on PWMn_CAP[1] will cause CR1 to be loaded with the contents of the TC.	
4	Capture on PWMn_CAP[1] falling edge <sup>[1]</sup>	0	This feature is disabled.	0
		1	A synchronously sampled falling edge on PWMn_CAP[1] will cause CR1 to be loaded with the contents of TC.	

**Table 556: PWM Capture Control Register (PWM0CCR - 0x4001 4028 and PWM1CCR - 0x4001 8028) bit description**

Bit	Symbol	Value	Description	Reset Value
5	Interrupt on PWMn_CAP[1] event <sup>[1]</sup>	0	This feature is disabled.	0
		1	A CR1 load due to a PWMn_CAP[1] event will generate an interrupt.	
31:6	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] Reserved for PWM0.

### 26.6.10 PWM Capture Registers (PWM0CR0 - 0x4001 402C and PWM1CR0/CR1- 0x4001 802C/30)

Each 32-bit Capture Register is associated with a device pin and may be loaded with the PWM Timer Counter value when a specified event occurs on that pin. The settings in the PWM Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

### 26.6.11 PWM Control Registers (PWM0PCR - 0x4001 404C and PWM1PCR 0x4001 804C)

The PWM Control registers are used to enable and select the type of each PWM channel. The function of each of the bits are shown in [Table 557](#).

**Table 557: PWM Control Registers (PWMPCR - 0x4001 404C and PWM1PCR - 0x4001 804C) bit description**

Bit	Symbol	Value	Description	Reset Value
1:0	Unused		Unused, always zero.	NA
2	PWMSEL2		PWM[2] output single/double edge mode control.	0
		1	Double edge controlled mode is selected.	
		0	Single edge controlled mode is selected.	
3	PWMSEL3	1	PWM[3] output edge control. See PWMSEL2 for details.	0
4	PWMSEL4	1	PWM[4] output edge control. See PWMSEL2 for details.	0
5	PWMSEL5	1	PWM[5] output edge control. See PWMSEL2 for details.	0
6	PWMSEL6	1	PWM[6] output edge control. See PWMSEL2 for details.	0
8:7	-		Reserved. Read value is undefined, only zero should be written.	NA
9	PWMENA1		PWM[1] output enable control.	0
		0	The PWM output is disabled.	
		1	The PWM output is enabled.	
10	PWMENA2		PWM[2] output enable control. See PWMENA1 for details.	0
11	PWMENA3		PWM[3] output enable control. See PWMENA1 for details.	0
12	PWMENA4		PWM[4] output enable control. See PWMENA1 for details.	0
13	PWMENA5		PWM[5] output enable control. See PWMENA1 for details.	0
14	PWMENA6		PWM[6] output enable control. See PWMENA1 for details.	0
31:15	Unused		Unused, always zero.	NA

**26.6.12 PWM Latch Enable Register (PWM0LER - 0x4001 4050 and PWM1LER 0x4001 8050)**

The PWM Latch Enable registers are used to control the update of the PWM Match registers when they are used for PWM generation. When software writes to the location of a PWM Match register while the Timer is in PWM mode, the value is actually held in a shadow register and not used immediately.

When a PWM Match 0 event occurs (normally also resetting the timer in PWM mode), the contents of shadow registers will be transferred to the actual Match registers if the corresponding bit in the Latch Enable register has been set. At that point, the new values will take effect and determine the course of the next PWM cycle. Once the transfer of new values has taken place, all bits of the LER are automatically cleared. Until the corresponding bit in the PWMLER is set and a PWM Match 0 event occurs, any value written to the PWM Match registers has no effect on PWM operation.

For example, if PWM is configured for double edge operation and is currently running, a typical sequence of events for changing the timing would be:

- Write a new value to the PWM Match1 register.
- Write a new value to the PWM Match2 register.
- Write to the PWMLER, setting bits 1 and 2 at the same time.
- The altered values will become effective at the next reset of the timer (when a PWM Match 0 event occurs).

The order of writing the two PWM Match registers is not important, since neither value will be used until after the write to PWMLER. This insures that both values go into effect at the same time, if that is required. A single value may be altered in the same way if needed.

The function of each of the bits in the PWMLER is shown in [Table 558](#).

**Table 558: PWM Latch Enable Register (PWM0LER - 0x4001 4050 and PWM1LER - 0x4001 8050) bit description**

Bit	Symbol	Description	Reset Value
0	Enable PWM Match 0 Latch	PWM MR0 register update control. Writing a one to this bit allows the last value written to the PWM Match Register 0 to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 26.6.7</a> .	0
1	Enable PWM Match 1 Latch	PWM MR1 register update control. See bit 0 for details.	0
2	Enable PWM Match 2 Latch	PWM MR2 register update control. See bit 0 for details.	0
3	Enable PWM Match 3 Latch	PWM MR3 register update control. See bit 0 for details.	0
4	Enable PWM Match 4 Latch	PWM MR4 register update control. See bit 0 for details.	0
5	Enable PWM Match 5 Latch	PWM MR5 register update control. See bit 0 for details.	0
6	Enable PWM Match 6 Latch	PWM MR6 register update control. See bit 0 for details.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA



### 27.1 Basic configuration

---

The Motor Control PWM is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCMCPWM.  
**Remark:** On reset, the MCPWM is disabled (PCMCPWM = 0).
2. Peripheral clock: The MCPWM operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select MCPWM pins through and pin modes for port pins with MCPWM functions through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: See [Section 27.9.8](#). The MCPWM interrupt is enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 27.2 Introduction

---

The Motor Control PWM (MCPWM) is optimized for three-phase AC and DC motor control applications, but can be used in many other applications that need timing, counting, capture, and comparison.

### 27.3 Description

---

The MCPWM contains three independent channels, each including:

- a 32-bit Timer/Counter (TC)
- a 32-bit Limit register (LIM)
- a 32-bit Match register (MAT)
- a 10-bit dead-time register (DT) and an associated 10-bit dead-time counter
- a 32-bit capture register (CAP)
- two modulated outputs (MC\_A and MC\_B) with opposite polarities
- a period interrupt, a pulse-width interrupt, and a capture interrupt

Input pins MC\_FB0-2 can trigger TC capture or increment a channel's TC. A global Abort input can force all of the channels into "A passive" state and cause an interrupt.

## 27.4 Pin description

[Table 559](#) lists the MCPWM pins.

**Table 559. Pin summary**

Pin	Type	Description
MC_0A, MC_0B	O	Outputs A and B for channel 0
MC_1A, MC_1B	O	Outputs A and B for channel 1
MC_2A, MC_2B	O	Outputs A and B for channel 2
MC_ABORT	I	Low-active Fast Abort
MC_FB0, MC_FB1, MC_FB2	I	Inputs for channels 0, 1, 2

27.5 Block diagram

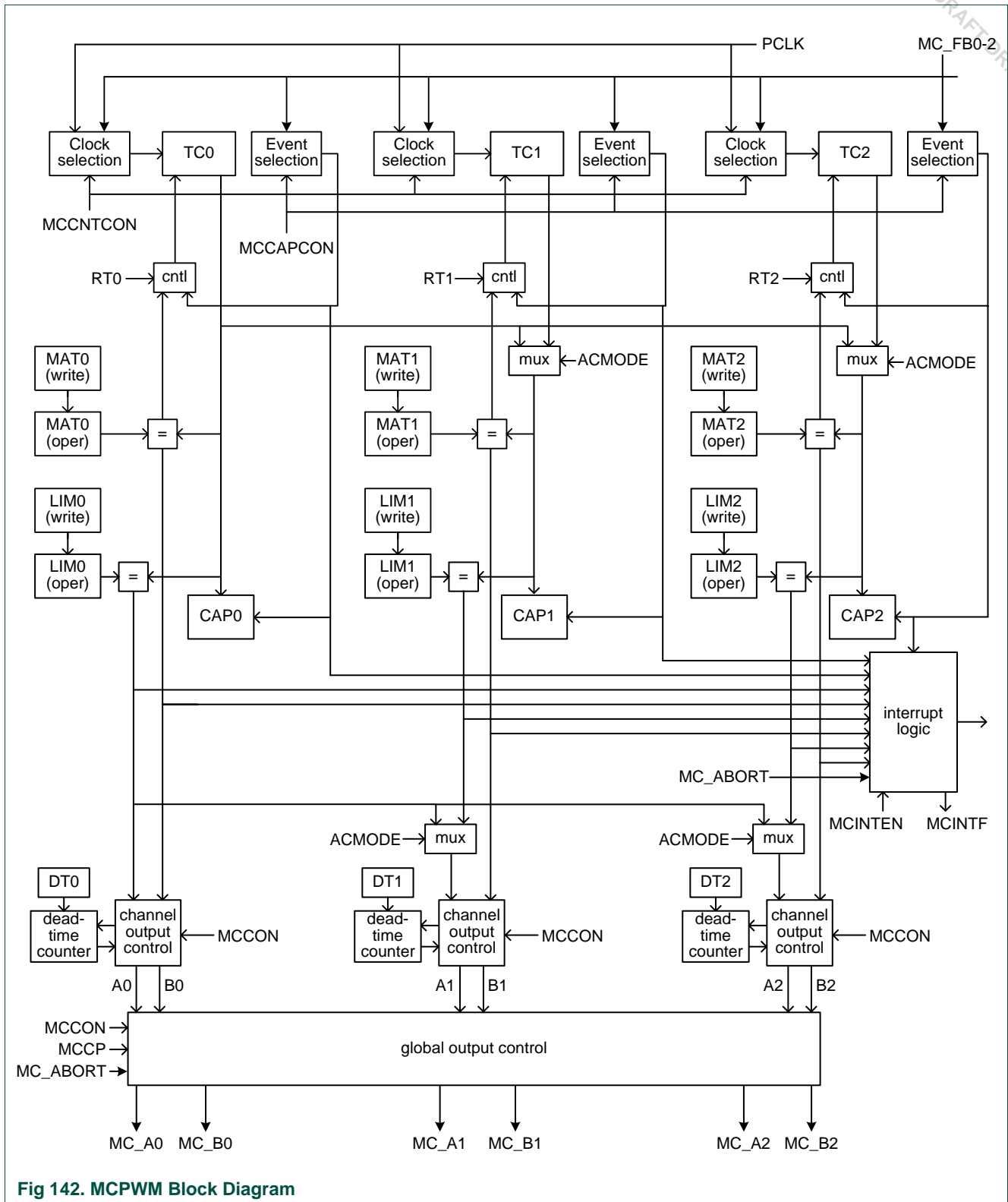


Fig 142. MCPWM Block Diagram

## 27.6 Configuring other modules for MCPWM use

Configure the following registers in other modules before using the Motor Control PWM:

1. Power: in the PCONP register ([Table 37](#)), set bit PCMCPWM.  
**Remark:** On reset the MCPWM is disabled (PCMCPWM = 0).
2. Peripheral clock: the MCPWM operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: select MCPWM functions and pin modes for these pins through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: See [Section 27.8.3](#) for motor control PWM related interrupts. Interrupts can be enabled in the NVIC using the appropriate Interrupt Set Enable register.

## 27.7 General operation

[Section 27.9](#) includes detailed descriptions of the various modes of MCPWM operation, but a quick preview here will provide background for the register descriptions below.

The MCPWM includes 3 channels, each of which controls a pair of outputs that in turn can control something off-chip, like one set of coils in a motor. Each channel includes a Timer/Counter (TC) register that is incremented by a processor clock (timer mode) or by an input pin (counter mode).

Each channel has a Limit register that is compared to the TC value, and when a match occurs the TC is “recycled” in one of two ways. In “edge-aligned mode” the TC is reset to 0, while in “centered mode” a match switches the TC into a state in which it decrements on each processor clock or input pin transition until it reaches 0, at which time it starts counting up again.

Each channel also includes a Match register that holds a smaller value than the Limit register. In edge-aligned mode the channel’s outputs are switched whenever the TC matches either the Match or Limit register, while in center-aligned mode they are switched only when it matches the Match register.

So the Limit register controls the period of the outputs, while the Match register controls how much of each period the outputs spend in each state. Having a small value in the Limit register minimizes “ripple” if the output is integrated into a voltage, and allows the MCPWM to control devices that operate at high speed.

The “downside” of small values in the Limit register is that they reduce the resolution of the duty cycle controlled by the Match register. If you have 8 in the Limit register, the Match register can only select the duty cycle among 0%, 12.5%, 25%, ..., 87.5%, or 100%. In general, the resolution of each step in the Match value is 1 divided by the Limit value.

This trade-off between resolution and period/frequency is inherent in the design of pulse width modulators.

## 27.8 Register description

Many registers in the MCPWM have separate read, set, and clear addresses. Reading such a register's read address (e.g. MCCON) yields the state of the register bits. Writing ones to the set address (e.g. MCCON\_SET) sets register bit(s), and writing ones to the clear address (e.g. MCCON\_CLR) clears register bit(s). Other registers are standard read-write registers.

**Table 560. Motor Control Pulse Width Modulator (MCPWM) register map**

Name	Description	Access	Reset value	Address	Table
MCCON	PWM Control read address	RO	0	0x400B 8000	<a href="#">561</a>
MCCON_SET	PWM Control set address	WO	-	0x400B 8004	<a href="#">562</a>
MCCON_CLR	PWM Control clear address	WO	-	0x400B 8008	<a href="#">563</a>
MCCAPCON	Capture Control read address	RO	0	0x400B 800C	<a href="#">564</a>
MCCAPCON_SET	Capture Control set address	WO	-	0x400B 8010	<a href="#">565</a>
MCCAPCON_CLR	Event Control clear address	WO	-	0x400B 8014	<a href="#">566</a>
MCTC0	Timer Counter register, channel 0	R/W	0	0x400B 8018	<a href="#">578</a>
MCTC1	Timer Counter register, channel 1	R/W	0	0x400B 801C	<a href="#">578</a>
MCTC2	Timer Counter register, channel 2	R/W	0	0x400B 8020	<a href="#">578</a>
MCLIM0	Limit register, channel 0	R/W	0xFFFF FFFF	0x400B 8024	<a href="#">579</a>
MCLIM1	Limit register, channel 1	R/W	0xFFFF FFFF	0x400B 8028	<a href="#">579</a>
MCLIM2	Limit register, channel 2	R/W	0xFFFF FFFF	0x400B 802C	<a href="#">579</a>
MCMAT0	Match register, channel 0	R/W	0xFFFF FFFF	0x400B 8030	<a href="#">580</a>
MCMAT1	Match register, channel 1	R/W	0xFFFF FFFF	0x400B 8034	<a href="#">580</a>
MCMAT2	Match register, channel 2	R/W	0xFFFF FFFF	0x400B 8038	<a href="#">580</a>
MCDT	Dead time register	R/W	0x3FFF FFFF	0x400B 803C	<a href="#">581</a>
MCCP	Commutation Pattern register	R/W	0	0x400B 8040	<a href="#">582</a>
MCCAP0	Capture register, channel 0	RO	0	0x400B 8044	<a href="#">583</a>
MCCAP1	Capture register, channel 1	RO	0	0x400B 8048	<a href="#">583</a>
MCCAP2	Capture register, channel 2	RO	0	0x400B 804C	<a href="#">583</a>
MCINTEN	Interrupt Enable read address	RO	0	0x400B 8050	<a href="#">569</a>
MCINTEN_SET	Interrupt Enable set address	WO	-	0x400B 8054	<a href="#">570</a>
MCINTEN_CLR	Interrupt Enable clear address	WO	-	0x400B 8058	<a href="#">571</a>
MCCNTCON	Count Control read address	RO	0	0x400B 805C	<a href="#">575</a>
MCCNTCON_SET	Count Control set address	WO	-	0x400B 8060	<a href="#">576</a>
MCCNTCON_CLR	Count Control clear address	WO	-	0x400B 8064	<a href="#">577</a>
MCINTF	Interrupt flags read address	RO	0	0x400B 8068	<a href="#">572</a>
MCINTF_SET	Interrupt flags set address	WO	-	0x400B 806C	<a href="#">573</a>
MCINTF_CLR	Interrupt flags clear address	WO	-	0x400B 8070	<a href="#">574</a>
MCCAP_CLR	Capture clear address	WO	-	0x400B 8074	<a href="#">584</a>

27.8.1 MCPWM Control register

27.8.1.1 MCPWM Control read address (MCCON - 0x400B 8000)

The MCCON register controls the operation of all channels of the PWM. This address is read-only, but the underlying register can be modified by writing to addresses MCCON\_SET and MCCON\_CLR.

Table 561. MCPWM Control read address (MCCON - 0x400B 8000) bit description

Bit	Symbol	Value	Description	Reset value
0	RUN0		Stops/starts timer channel 0.	0
		0	Stop.	
		1	Run.	
1	CENTER0		Edge/center aligned operation for channel 0.	0
		0	Edge-aligned.	
		1	Center-aligned.	
2	POLA0		Selects polarity of the MC_0A and MC_0B pins.	0
		0	Passive state is LOW, active state is HIGH.	
		1	Passive state is HIGH, active state is LOW.	
3	DTE0		Controls the dead-time feature for channel 0.	0
		0	Dead-time disabled.	
		1	Dead-time enabled.	
4	DISUP0		Enable/disable updates of functional registers for channel 0 (see <a href="#">Section 27.9.2</a> ).	0
		0	Functional registers are updated from the write registers at the end of each PWM cycle.	
		1	Functional registers remain the same as long as the timer is running.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	
8	RUN1		Stops/starts timer channel 1.	0
		0	Stop.	
		1	Run.	
9	CENTER1		Edge/center aligned operation for channel 1.	0
		0	Edge-aligned.	
		1	Center-aligned.	
10	POLA1		Selects polarity of the MC_1A and MC_1B pins.	0
		0	Passive state is LOW, active state is HIGH.	
		1	Passive state is HIGH, active state is LOW.	
11	DTE1		Controls the dead-time feature for channel 1.	0
		0	Dead-time disabled.	
		1	Dead-time enabled.	
12	DISUP1		Enable/disable updates of functional registers for channel 1 (see <a href="#">Section 27.9.2</a> ).	0
		0	Functional registers are updated from the write registers at the end of each PWM cycle.	
		1	Functional registers remain the same as long as the timer is running.	
15:13	-		Reserved. The value read from a reserved bit is not defined.	0

Table 561. MCPWM Control read address (MCCON - 0x400B 8000) bit description

Bit	Symbol	Value	Description	Reset value
16	RUN2		Stops/starts timer channel 2.	0
		0	Stop.	
		1	Run.	
17	CENTER2		Edge/center aligned operation for channel 2.	0
		0	Edge-aligned.	
		1	Center-aligned.	
18	POLA2		Selects polarity of the MC_2A and MC_2B pins.	0
		0	Passive state is LOW, active state is HIGH.	
		1	Passive state is HIGH, active state is LOW.	
19	DTE2		Controls the dead-time feature for channel 1.	0
		0	Dead-time disabled.	
		1	Dead-time enabled.	
20	DISUP2		Enable/disable updates of functional registers for channel 2 (see <a href="#">Section 27.9.2</a> ).	0
		0	Functional registers are updated from the write registers at the end of each PWM cycle.	
		1	Functional registers remain the same as long as the timer is running.	
28:21	-		Reserved. Read value is undefined, only zero should be written.	
29	INVBDC		Controls the polarity of the MC_B outputs for all 3 channels. This bit is typically set to 1 only in 3-phase DC mode.	
		0	The MC_B outputs have opposite polarity from the MC_A outputs (aside from dead time).	
		1	The MC_B outputs have the same basic polarity as the MC_A outputs. (see <a href="#">Section 27.9.6</a> )	
30	ACMODE		3-phase AC mode select (see <a href="#">Section 27.9.7</a> ).	0
		0	3-phase AC-mode off: Each PWM channel uses its own timer-counter and period register.	
		1	3-phase AC-mode on: All PWM channels use the timer-counter and period register of channel 0.	
31	DCMODE		3-phase DC mode select (see <a href="#">Section 27.9.6</a> ).	0
		0	3-phase DC mode off: PWM channels are independent (unless bit ACMODE = 1)	
		1	3-phase DC mode on: The internal MC_0A output is routed through the MCCP (i.e. a mask) register to all six PWM outputs.	

### 27.8.1.2 MCPWM Control set address (MCCON\_SET - 0x400B 8004)

Writing ones to this write-only address sets the corresponding bits in MCCON.

Table 562. MCPWM Control set address (MCCON\_SET - 0x400B 8004) bit description

Bit	Description
31:0	Writing ones to this address sets the corresponding bits in the MCCON register. See <a href="#">Table 561</a> .

### 27.8.1.3 MCPWM Control clear address (MCCON\_CLR - 0x400B 8008)

Writing ones to this write-only address clears the corresponding bits in MCCON.

**Table 563. MCPWM Control clear address (MCCON\_CLR - 0x400B 8008) bit description**

Bit	Description
31:0	Writing ones to this address clears the corresponding bits in the MCCON register. See <a href="#">Table 561</a> .



## 27.8.2 MCPWM Capture Control register

### 27.8.2.1 MCPWM Capture Control read address (MCCAPCON - 0x400B 800C)

The MCCAPCON register controls detection of events on the MC\_FB0-2 inputs for all MCPWM channels. Any of the three MC\_FB inputs can be used to trigger a capture event on any or all of the three channels. This address is read-only, but the underlying register can be modified by writing to addresses MCCAPCON\_SET and MCCAPCON\_CLR.

**Table 564. MCPWM Capture Control read address (MCCAPCON - 0x400B 800C) bit description**

Bit	Symbol	Description	Reset Value
0	CAP0MCFB0_RE	A 1 in this bit enables a channel 0 capture event on a rising edge on MC_FB0.	0
1	CAP0MCFB0_FE	A 1 in this bit enables a channel 0 capture event on a falling edge on MC_FB0.	0
2	CAP0MCFB1_RE	A 1 in this bit enables a channel 0 capture event on a rising edge on MC_FB1.	0
3	CAP0MCFB1_FE	A 1 in this bit enables a channel 0 capture event on a falling edge on MC_FB1.	0
4	CAP0MCFB2_RE	A 1 in this bit enables a channel 0 capture event on a rising edge on MC_FB2.	0
5	CAP0MCFB2_FE	A 1 in this bit enables a channel 0 capture event on a falling edge on MC_FB2.	0
6	CAP1MCFB0_RE	A 1 in this bit enables a channel 1 capture event on a rising edge on MC_FB0.	0
7	CAP1MCFB0_FE	A 1 in this bit enables a channel 1 capture event on a falling edge on MC_FB0.	0
8	CAP1MCFB1_RE	A 1 in this bit enables a channel 1 capture event on a rising edge on MC_FB1.	0
9	CAP1MCFB1_FE	A 1 in this bit enables a channel 1 capture event on a falling edge on MC_FB1.	0
10	CAP1MCFB2_RE	A 1 in this bit enables a channel 1 capture event on a rising edge on MC_FB2.	0
11	CAP1MCFB2_FE	A 1 in this bit enables a channel 1 capture event on a falling edge on MC_FB2.	0
12	CAP2MCFB0_RE	A 1 in this bit enables a channel 2 capture event on a rising edge on MC_FB0.	0
13	CAP2MCFB0_FE	A 1 in this bit enables a channel 2 capture event on a falling edge on MC_FB0.	0
14	CAP2MCFB1_RE	A 1 in this bit enables a channel 2 capture event on a rising edge on MC_FB1.	0
15	CAP2MCFB1_FE	A 1 in this bit enables a channel 2 capture event on a falling edge on MC_FB1.	0
16	CAP2MCFB2_RE	A 1 in this bit enables a channel 2 capture event on a rising edge on MC_FB2.	0
17	CAP2MCFB2_FE	A 1 in this bit enables a channel 2 capture event on a falling edge on MC_FB2.	0
18	RT0	If this bit is 1, TC0 is reset by a channel 0 capture event.	0
19	RT1	If this bit is 1, TC1 is reset by a channel 1 capture event.	0
20	RT2	If this bit is 1, TC2 is reset by a channel 2 capture event.	0
21	HNFCAP0	Hardware noise filter: if this bit is 1, channel 0 capture events are delayed as described in <a href="#">Section 27.9.4</a> .	0
22	HNFCAP1	Hardware noise filter: if this bit is 1, channel 1 capture events are delayed as described in <a href="#">Section 27.9.4</a> .	0
23	HNFCAP2	Hardware noise filter: if this bit is 1, channel 2 capture events are delayed as described in <a href="#">Section 27.9.4</a> .	0
31:24	-	Reserved. The value read from a reserved bit is not defined.	-

### 27.8.2.2 MCPWM Capture Control set address (MCCAPCON\_SET - 0x400B 8010)

Writing ones to this write-only address sets the corresponding bits in MCCAPCON.

**Table 565. MCPWM Capture Control set address (MCCAPCON\_SET - 0x400B 8010) bit description**

Bit	Description
31:0	Writing ones to this address sets the corresponding bits in the MCCAPCON register. See <a href="#">Table 564</a> .

**27.8.2.3 MCPWM Capture control clear address (MCCAPCON\_CLR - 0x400B 8014)**

Writing ones to this write-only address clears the corresponding bits in MCCAPCON.

**Table 566. MCPWM Capture control clear register (MCCAPCON\_CLR - address 0x400B 8014) bit description**

Bit	Description
31:0	Writing ones to this address clears the corresponding bits in the MCCAPCON register. See <a href="#">Table 564</a> .

### 27.8.3 MCPWM Interrupt registers

The Motor Control PWM module includes the following interrupt sources:

**Table 567. Motor Control PWM interrupts**

Symbol	Description
ILIM0/1/2	Limit interrupts for channels 0, 1, 2.
IMAT0/1/2	Match interrupts for channels 0, 1, 2.
ICAP0/1/2	Capture interrupts for channels 0, 1, 2.
ABORT	Fast abort interrupt

All MCPWM interrupt registers contain one bit for each source as shown in [Table 568](#).

**Table 568. Interrupt sources bit allocation table**

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	-	-	-	-	-	-
Bit	15	14	13	12	11	10	9	8
Symbol	ABORT	-	-	-	-	ICAP2	IMAT2	ILIM2
Bit	7	6	5	4	3	2	1	0
Symbol	-	ICAP1	IMAT1	ILIM1	-	ICAP0	IMAT0	ILIM0

#### 27.8.3.1 MCPWM Interrupt Enable read address (MCINTEN - 0x400B 8050)

The MCINTEN register controls which of the MCPWM interrupts are enabled. This address is read-only, but the underlying register can be modified by writing to addresses MCINTEN\_SET and MCINTEN\_CLR.

**Table 569. MCPWM Interrupt Enable read address (MCINTEN - 0x400B 8050) bit description**

Bit	Value	Description	Reset value
31:0		See <a href="#">Table 568</a> for the bit allocation.	0
	0	Interrupt disabled.	
	1	Interrupt enabled.	

#### 27.8.3.2 MCPWM Interrupt Enable set address (MCINTEN\_SET - 0x400B 8054)

Writing ones to this write-only address sets the corresponding bits in MCINTEN, thus enabling interrupts.

**Table 570. PWM interrupt enable set register (MCINTEN\_SET - address 0x400B 8054) bit description**

Bit	Description
31:0	Writing ones to this address sets the corresponding bits in MCINTEN, thus enabling interrupts. See <a href="#">Table 568</a> .

### 27.8.3.3 MCPWM Interrupt Enable clear address (MCINTEN\_CLR - 0x400B 8058)

Writing ones to this write-only address clears the corresponding bits in MCINTEN, thus disabling interrupts.

**Table 571. PWM interrupt enable clear register (MCINTEN\_CLR - address 0x400B 8058) bit description**

Bit	Description
31:0	Writing ones to this address clears the corresponding bits in MCINTEN, thus disabling interrupts. See <a href="#">Table 568</a> .

### 27.8.3.4 MCPWM Interrupt Flags read address (MCINTF - 0x400B 8068)

The MCINTF register includes all MCPWM interrupt flags, which are set when the corresponding hardware event occurs, or when ones are written to the MCINTF\_SET address. When corresponding bits in this register and MCINTEN are both 1, the MCPWM asserts its interrupt request to the Interrupt Controller module. This address is read-only, but the bits in the underlying register can be modified by writing ones to addresses MCINTF\_SET and MCINTF\_CLR.

**Table 572. MCPWM Interrupt Flags read address (MCINTF - 0x400B 8068) bit description**

Bit	Value	Description	Reset value
31:0		See <a href="#">Table 568</a> for the bit allocation.	0
	0	This interrupt source is not contributing to the MCPWM interrupt request.	
	1	If the corresponding bit in MCINTEN is 1, the MCPWM module is asserting its interrupt request to the Interrupt Controller.	

### 27.8.3.5 MCPWM Interrupt Flags set address (MCINTF\_SET - 0x400B 806C)

Writing one(s) to this write-only address sets the corresponding bit(s) in MCINTF, thus possibly simulating hardware interrupt(s).

**Table 573. MCPWM Interrupt Flags set address (MCINTF\_SET - 0x400B 806C) bit description**

Bit	Description
31:0	Writing one(s) to this write-only address sets the corresponding bit(s) in the MCINTF register, thus possibly simulating hardware interrupt(s). See <a href="#">Table 568</a> .

### 27.8.3.6 MCPWM Interrupt Flags clear address (MCINTF\_CLR - 0x400B 8070)

Writing one(s) to this write-only address sets the corresponding bit(s) in MCINTF, thus clearing the corresponding interrupt request(s). This is typically done in interrupt service routines.

**Table 574. MCPWM Interrupt Flags clear address (MCINTF\_CLR - 0x400B 8070) bit description**

Bit	Description
31:0	Writing one(s) to this write-only address sets the corresponding bit(s) in the MCINTF register, thus clearing the corresponding interrupt request(s). See <a href="#">Table 568</a> .

### 27.8.4 MCPWM Count Control register

#### 27.8.4.1 MCPWM Count Control read address (MCCNTCON - 0x400B 805C)

The MCCNTCON register controls whether the MCPWM channels are in timer or counter mode, and in counter mode whether the counter advances on rising and/or falling edges on any or all of the three MC\_FB inputs. If timer mode is selected, the counter advances based on the PCLK clock.

This address is read-only. To set or clear the register bits, write ones to the MCCNTCON\_SET or MCCNTCON\_CLR address.

**Table 575. MCPWM Count Control read address (MCCNTCON - 0x400B 805C) bit description**

Bit	Symbol	Value	Description	Reset Value
0	TC0MCFB0_RE	0	A rising edge on MC_FB0 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a rising edge on MC_FB0.	
1	TC0MCFB0_FE	0	A falling edge on MC_FB0 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a falling edge on MC_FB0.	
2	TC0MCFB1_RE	0	A rising edge on MC_FB1 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a rising edge on MC_FB1.	
3	TC0MCFB1_FE	0	A falling edge on MC_FB1 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a falling edge on MC_FB1.	
4	TC0MCFB2_RE	0	A rising edge on MC_FB0 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a rising edge on MC_FB2.	
5	TC0MCFB2_FE	0	A falling edge on MC_FB0 does not affect counter 0.	0
		1	If MODE0 is 1, counter 0 advances on a falling edge on MC_FB2.	
6	TC1MCFB0_RE	0	A rising edge on MC_FB0 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a rising edge on MC_FB0.	
7	TC1MCFB0_FE	0	A falling edge on MC_FB0 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a falling edge on MC_FB0.	
8	TC1MCFB1_RE	0	A rising edge on MC_FB1 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a rising edge on MC_FB1.	
9	TC1MCFB1_FE	0	A falling edge on MC_FB0 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a falling edge on MC_FB1.	
10	TC1MCFB2_RE	0	A rising edge on MC_FB2 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a rising edge on MC_FB2.	
11	TC1MCFB2_FE	0	A falling edge on MC_FB2 does not affect counter 1.	0
		1	If MODE1 is 1, counter 1 advances on a falling edge on MC_FB2.	
12	TC2MCFB0_RE	0	A rising edge on MC_FB0 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a rising edge on MC_FB0.	
13	TC2MCFB0_FE	0	A falling edge on MC_FB0 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a falling edge on MC_FB0.	
14	TC2MCFB1_RE	0	A rising edge on MC_FB1 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a rising edge on MC_FB1.	

**Table 575. MCPWM Count Control read address (MCCNTCON - 0x400B 805C) bit description**

Bit	Symbol	Value	Description	Reset Value
15	TC2MCFB1_FE	0	A falling edge on MC_FB1 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a falling edge on MC_FB1.	
16	TC2MCFB2_RE	0	A rising edge on MC_FB2 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a rising edge on MC_FB2.	
17	TC2MCFB2_FE	0	A falling edge on MC_FB2 does not affect counter 2.	0
		1	If MODE2 is 1, counter 2 advances on a falling edge on MC_FB2.	
28:18	-		Reserved. Read value is undefined, only zero should be written.	-
29	CNTR0	0	Channel 0 is in timer mode.	0
		1	Channel 0 is in counter mode.	
30	CNTR1	0	Channel 1 is in timer mode.	0
		1	Channel 1 is in counter mode.	
31	CNTR2	0	Channel 2 is in timer mode.	0
		1	Channel 2 is in counter mode.	

**27.8.4.2 MCPWM Count Control set address (MCCNTCON\_SET - 0x400B 8060)**

Writing one(s) to this write-only address sets the corresponding bit(s) in MCCNTCON.

**Table 576. MCPWM Count Control set address (MCCNTCON\_SET - 0x400B 8060) bit description**

Bit	Description
31:0	Writing one(s) to this write-only address sets the corresponding bit(s) in the MCCNTCON register. See <a href="#">Table 575</a> .

**27.8.4.3 MCPWM Count Control clear address (MCCNTCON\_CLR - 0x400B 8064)**

Writing one(s) to this write-only address clears the corresponding bit(s) in MCCNTCON.

**Table 577. MCPWM Count Control clear address (MCCAPCON\_CLR - 0x400B 8064) bit description**

Bit	Description
31:0	Writing one(s) to this write-only address clears the corresponding bit(s) in the MCCNTCON register. See <a href="#">Table 575</a> .

**27.8.5 MCPWM Timer/Counter 0-2 registers (MCTC0-2 - 0x400B 8018, 0x400B 801C, 0x400B 8020)**

These registers hold the current values of the 32-bit counter/timers for channels 0-2. Each value is incremented on every PCLK, or by edges on the MC\_FB0-2 pins, as selected by MCCNTCON. The timer/counter counts up from 0 until it reaches the value in its corresponding MCLIM register (or is stopped by writing to MCON\_CLR).

A TC register can be read at any time. In order to write to the TC register, its channel must be stopped. If not, the write will not take place, no exception is generated.

**Table 578. MCPWM Timer/Counter 0-2 registers (MCTC0-2 - 0x400B 8018, 0x400B 801C, 0x400B 8020) bit description**

Bit	Symbol	Description	Reset value
31:0	MCTC0/1/2	Timer/Counter values for channels 0, 1, 2.	0

**27.8.6 MCPWM Limit 0-2 registers (MCLIM0-2 - 0x400B 8024, 0x400B 8028, 0x400B 802C)**

These registers hold the limiting values for timer/counters 0-2. When a timer/counter reaches its corresponding limiting value: 1) in edge-aligned mode, it is reset and starts over at 0; 2) in center-aligned mode, it begins counting down until it reaches 0, at which time it begins counting up again.

If the channel's CENTER bit in MCCON is 0 selecting edge-aligned mode, the match between TC and LIM switches the channel's A output from "active" to "passive" state. If the channel's CENTER and DTE bits in MCCON are both 0, the match simultaneously switches the channel's B output from "passive" to "active" state.

If the channel's CENTER bit is 0 but the DTE bit is 1, the match triggers the channel's deadtime counter to begin counting -- when the deadtime counter expires, the channel's B output switches from "passive" to "active" state.

In center-aligned mode, matches between a channel's TC and LIM registers have no effect on its A and B outputs.

Writing to either a Limit or a Match (27.8.7) register loads a "write" register, and if the channel is stopped it also loads an "operating" register that is compared to the TC. If the channel is running and its "disable update" bit in MCCON is 0, the operating registers are loaded from the write registers: 1) in edge-aligned mode, when the TC matches the operating Limit register; 2) in center-aligned mode, when the TC counts back down to 0. If the channel is running and the "disable update" bit is 1, the operating registers are not loaded from the write registers until software stops the channel.

Reading an MCLIM address always returns the operating value.

**Table 579. MCPWM Limit 0-2 registers (MCLIM0-2 - 0x400B 8024, 0x400B 8028, 0x400B 802C) bit description**

Bit	Symbol	Description	Reset value
31:0	MCLIM0/1/2	Limit values for TC0, 1, 2.	0xFFFF FFFF

**Note:** In timer mode, the period of a channel's modulated MC\_A and MC\_B outputs is determined by its Limit register, and the pulse width at the start of the period is determined by its Match register. If it suits your way of thinking, consider the Limit register to be the "Period register" and the Match register to be the "Pulse Width register".



### 27.8.7 MCPWM Match 0-2 registers (MCMAT0-2 - 0x400B 8030, 0x400B 8034, 0x400B 8038)

These registers also have “write” and “operating” versions as described above for the Limit registers, and the operating registers are also compared to the channels’ TCs. See [27.8.6](#) above for details of reading and writing both Limit and Match registers.

The Match and Limit registers control the MC\_A and MC\_B outputs. If a Match register is to have any effect on its channel’s operation, it must contain a smaller value than the corresponding Limit register.

**Table 580. MCPWM Match 0-2 registers (MCMAT0-2 - addresses 0x400B 8030, 0x400B 8034, 0x400B 8038) bit description**

Bit	Symbol	Description	Reset value
31:0	MCMAT0/1/2	Match values for TC0, 1, 2.	0xFFFF FFFF

#### 27.8.7.1 Match register in Edge-Aligned mode

If the channel’s CENTER bit in MCCON is 0 selecting edge-aligned mode, a match between TC and MAT switches the channel’s B output from “active” to “passive” state. If the channel’s CENTER and DTE bits in MCCON are both 0, the match simultaneously switches the channel’s A output from “passive” to “active” state.

If the channel’s CENTER bit is 0 but the DTE bit is 1, the match triggers the channel’s deadtime counter to begin counting -- when the deadtime counter expires, the channel’s A output switches from “passive” to “active” state.

#### 27.8.7.2 Match register in Center-Aligned mode

If the channel’s CENTER bit in MCCON is 1 selecting center-aligned mode, a match between TC and MAT while the TC is incrementing switches the channel’s B output from “active” to “passive” state, and a match while the TC is decrementing switches the A output from “active” to “passive”. If the channel’s CENTER bit in MCCON is 1 but the DTE bit is 0, a match simultaneously switches the channel’s other output in the opposite direction.

If the channel’s CENTER and DTE bits are both 1, a match between TC and MAT triggers the channel’s deadtime counter to begin counting -- when the deadtime counter expires, the channel’s B output switches from “passive” to “active” if the TC was counting up at the time of the match, and the channel’s A output switches from “passive” to “active” if the TC was counting down at the time of the match.

#### 27.8.7.3 0 and 100% duty cycle

To lock a channel’s MC\_A and MC\_B outputs at the state “B active, A passive”, write its Match register with a higher value than you write to its Limit register. The match never occurs.

To lock a channel’s MC\_A and MC\_B outputs at the opposite state, “A active, B passive”, simply write 0 to its Match register.

### 27.8.8 MCPWM Dead-time register (MCDT - 0x400B 803C)

This register holds the dead-time values for the three channels. If a channel's DTE bit in MCON is 1 to enable its dead-time counter, the counter counts down from this value whenever one its channel's outputs changes from "active" to "passive" state. When the dead-time counter reaches 0, the channel changes its other output from "passive" to "active" state.

The motivation for the dead-time feature is that power transistors, like those driven by the A and B outputs in a motor-control application, take longer to fully turn off than they take to start to turn on. If the A and B transistors are ever turned on at the same time, a wasteful and damaging current will flow between the power rails through the transistors. In such applications, the dead-time register should be programmed with the number of PCLK periods that is greater than or equal to the transistors' maximum turn-off time minus their minimum turn-on time.

**Table 581. MCPWM Dead-time register (MCDT - address 0x400B 803C) bit description**

Bit	Symbol	Description	Reset value
9:0	DT0	Dead time for channel 0. <sup>[1]</sup>	0x3FF
19:10	DT1	Dead time for channel 1. <sup>[2]</sup>	0x3FF
29:20	DT2	Dead time for channel 2. <sup>[2]</sup>	0x3FF
31:30	-	Reserved. Read value is undefined, only zero should be written.	

[1] If ACMODE is 1 selecting AC-mode, this field controls the dead time for all three channels.

[2] If ACMODE is 0.

### 27.8.9 MCPWM Commutation Pattern register (MCCP - 0x400B 8040)

This register is used in DC mode only. The internal MC\_0A signal is routed to any or all of the six output pins under the control of the bits in this register. Like the Match and Limit registers, this register has "write" and "operational" versions. See [27.8.6](#) and [27.9.2](#) for more about this subject.

**Table 582. MCPWM Commutation Pattern register (MCCP - address 0x400B 8040) bit description**

Bit	Symbol	Description	Reset value
0	CCPA0	0 = MC_0A passive, 1 = internal MC_0A.	0
1	CCPB0	0 = MC_0B passive, 1 = MC_0B tracks internal MC_0A.	0
2	CCPA1	0 = MC_1A passive, 1 = MC_1A tracks internal MC_0A.	0
3	CCPB1	0 = MC_1B passive, 1 = MC_1B tracks internal MC_0A.	0
4	CCPA2	0 = MC_2A passive, 1 = MC_2A tracks internal MC_0A.	0
5	CCPB2	0 = MC_2B passive, 1 = MC_2B tracks internal MC_0A.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	

**27.8.10 MCPWM Capture Registers**

**27.8.10.1 MCPWM Capture read addresses (MCCAP0-2 - 0x400B 8044, 0x400B 8048, 0x400B 804C)**

The MCCAPCON register ([Table 564](#)) allows software to select any edge(s) on any of the MC\_FB0-2 inputs as a capture event for each channel. When a channel's capture event occurs, the current TC value for that channel is stored in its read-only Capture register. These addresses are read-only, but the underlying registers can be cleared by writing to the CAP\_CLR address

**Table 583. MCPWM Capture read addresses (MCCAP0/1/2 - 0x400B 8044, 0x400B 8048, 0x400B 804C) bit description**

Bit	Symbol	Description	Reset value
31:0	CAP0/1/2	TC value at a capture event for channels 0, 1, 2.	0x0000 0000

**27.8.10.2 MCPWM Capture clear address (MCCAP\_CLR - 0x400B 8074)**

Writing ones to this write-only address clears the selected CAP register(s).

**Table 584. MCPWM Capture clear address (CAP\_CLR - 0x400B 8074) bit description**

Bit	Symbol	Description
0	CAP_CLR0	Writing a 1 to this bit clears the MCCAP0 register.
1	CAP_CLR1	Writing a 1 to this bit clears the MCCAP1 register.
2	CAP_CLR2	Writing a 1 to this bit clears the MCCAP2 register.
31:3	-	Reserved. Read value is undefined, only zero should be written.

## 27.9 PWM operation

### 27.9.1 Pulse-width modulation

Each channel of the MCPWM has two outputs, A and B, that can drive a pair of transistors to switch a controlled point between two power rails. Most of the time the two outputs have opposite polarity, but a dead-time feature can be enabled (on a per-channel basis) to delay both signals' transitions from "passive" to "active" state so that the transistors are never both turned on simultaneously. In a more general view, the states of each output pair can be thought of "high", "low", and "floating" or "up", "down", and "center-off".

Each channel's mapping from "active" and "passive" to "high" and "low" is programmable. After Reset, the three A outputs are passive/low, and the B outputs are active/high.

The MCPWM can perform edge-aligned and center-aligned pulse-width modulation.

**Note:** In timer mode, the period of a channel's modulated MC\_A and MC\_B outputs is determined by its Limit register, and the pulse width at the start of the period is determined by its Match register. If it suits your way of thinking, consider the Limit register to be the "Period register" and the Match register to be the "Pulse Width register".

#### Edge-aligned PWM without dead-time

In this mode the timer TC counts up from 0 to the value in the LIM register. As shown in [Figure 143](#), the output pin state is "A passive" until the TC matches the Match register, at which point it changes to "A active". When the TC matches the Limit register, the output pin state changes back to "A passive", and the TC is reset and starts counting up again.

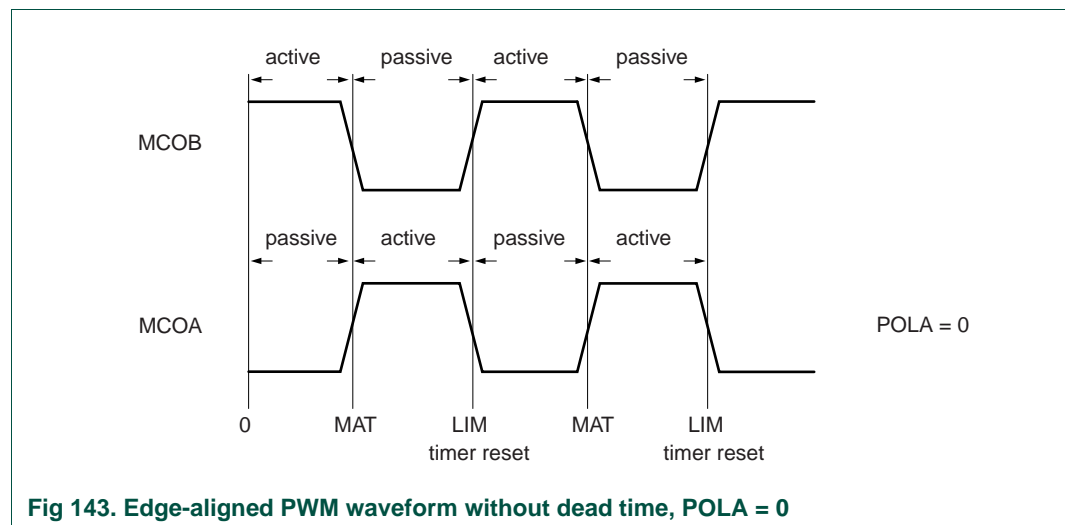


Fig 143. Edge-aligned PWM waveform without dead time, POLA = 0

#### Center-aligned PWM without dead-time

In this mode the timer TC counts up from 0 to the value in the LIM register, then counts back down to 0 and repeats. As shown in [Figure 144](#), while the timer counts up, the output pin state is "A passive" until the TC matches the Match register, at which point it changes to "A active". When the TC matches the Limit register it starts counting down. When the TC matches the Match register on the way down, the output pin state changes back to "A passive".

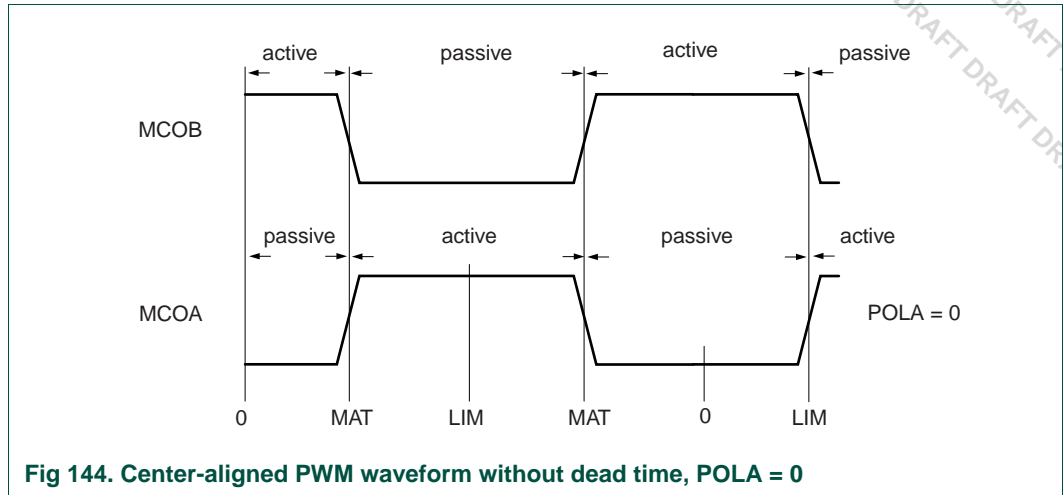


Fig 144. Center-aligned PWM waveform without dead time, POLA = 0

**Dead-time counter**

When the a channel's DTE bit is set in MCON, the dead-time counter delays the passive-to-active transitions of both output pins. The dead-time counter starts counting down, from the channel's DT value (in the MCDT register) to 0, whenever the channel's A or B output changes from active to passive. The transition of the other output from passive to active is delayed until the dead-time counter reaches 0. During the dead time, the MC\_A and MC\_B output levels are both passive. [Figure 145](#) shows operation in edge aligned mode with dead time, and [Figure 146](#) shows center-aligned operation with dead time.

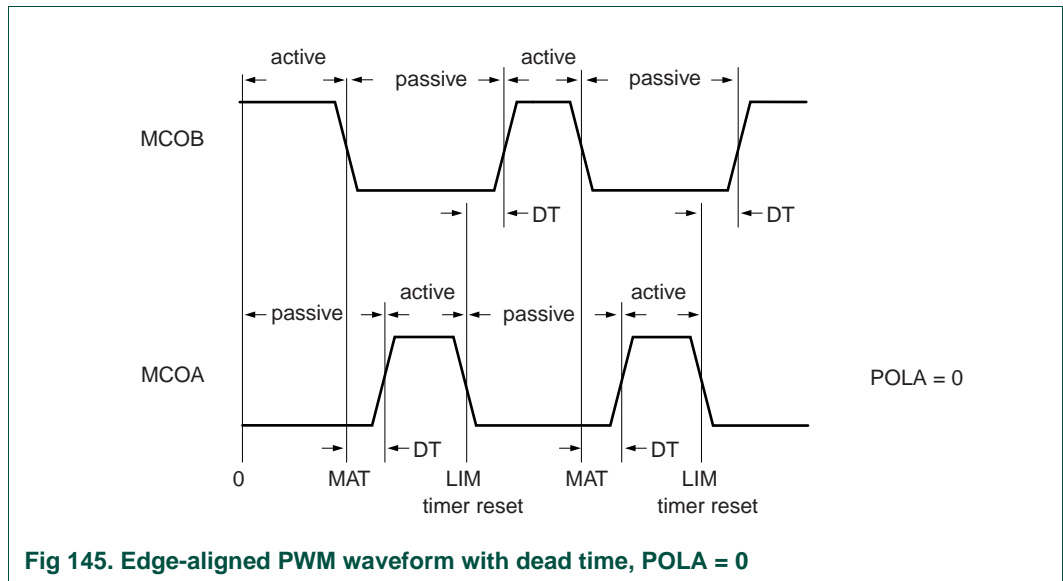


Fig 145. Edge-aligned PWM waveform with dead time, POLA = 0

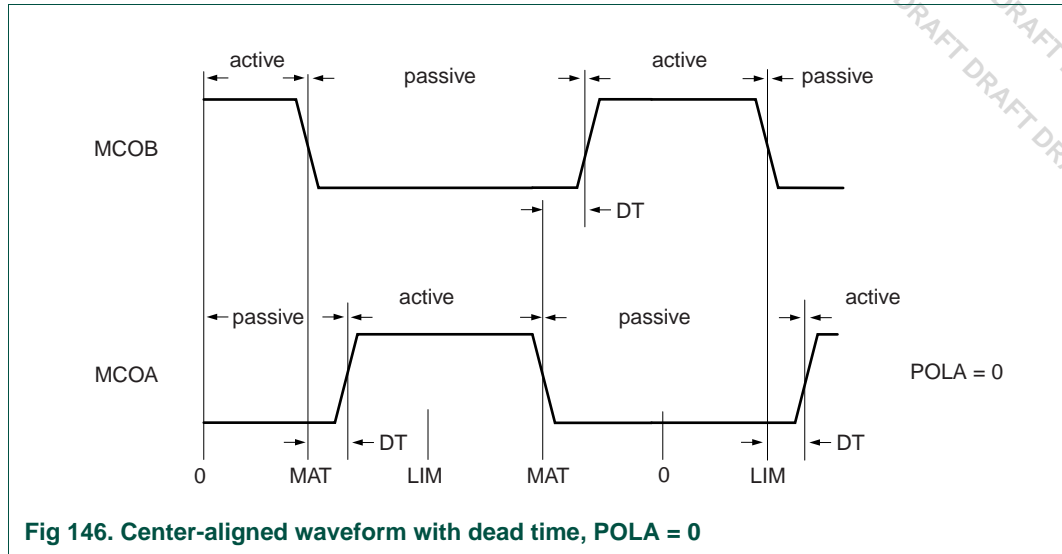


Fig 146. Center-aligned waveform with dead time, POLA = 0

### 27.9.2 Shadow registers and simultaneous updates

The Limit, Match, and Commutation Pattern registers (MCLIM, MCMAT, and MCCP) are implemented as register pairs, each consisting of a write register and an operational register. Software writes into the write registers. The operational registers control the actual operation of each channel and are loaded with the current value in the write registers when the TC starts counting up from 0.

Updating of the functional registers can be disabled by setting a channel's DISUP bit in the MCCON register. If the DISUP bits are set, the functional registers are not updated until software stops the channel.

If a channel is not running when software writes to its LIM or MAT register, the functional register is updated immediately.

Software can write to a TC register only when its channel is stopped.

### 27.9.3 Fast Abort (ABORT)

The MCPWM has an external input  $\overline{\text{MC\_ABORT}}$ . When this input goes low, all six output pins assume their "A passive" states, and the Abort interrupt is generated if enabled. The outputs remain locked in "A passive" state until the ABORT interrupt flag is cleared or the Abort interrupt is disabled. The ABORT flag may not be cleared before the  $\overline{\text{MC\_ABORT}}$  input goes high.

In order to clear an ABORT flag, a 1 must be written to bit 15 of the MCINTF\_CLR register. This will remove the interrupt request. The interrupt can also be disabled by writing a 1 to bit 15 of the MCINTEN\_CLR register.

### 27.9.4 Capture events

Each PWM channel can take a snapshot of its TC when an input signal transitions. Any channel may use any combination of rising and/or falling edges on any or all of the MC\_FB0-2 inputs as a capture event, under control of the MCCAPCON register. Rising or falling edges on the inputs are detected synchronously with respect to PCLK.

If a channel's HNF bit in the MCCAPCON register is set to enable “noise filtering”, a selected edge on an MC\_FB pin starts the dead-time counter for that channel, and the capture event actions described below are delayed until the dead-time counter reaches 0. This function is targeted specifically for performing three-phase brushless DC motor control with Hall sensors.

A capture event on a channel (possibly delayed by HNF) causes the following:

- The current value of the TC is stored in the Capture register (CAP).
- If the channel's capture event interrupt is enabled (see [Table 569](#)), the capture event interrupt flag is set.
- If the channel's RT bit is set in the MCCAPCON register, enabling reset on a capture event, the input event has the same effect as matching the channel's TC to its LIM register. This includes resetting the TC and switching the output pin(s) in edge-aligned mode as described in [27.8.6](#) and [27.9.1](#).

### 27.9.5 External event counting (Counter mode)

If a channel's MODE bit is 1 in MCCNTCON, its TC is incremented by rising and/or falling edge(s) (synchronously detected) on the MC\_FB0-2 input(s), rather than by PCLK. The PWM functions and capture functions are unaffected.

### 27.9.6 Three-phase DC mode

The three-phase DC mode is selected by setting the DCMODE bit in the MCCON register.

In this mode, the internal MC\_0A signal can be routed to any or all of the output pins. Each output pin is masked by a bit in the current Commutation Pattern register MCCP. If a bit in the MCCP register is 0, its output pin has the logic level for the passive state of output MC\_0A. The polarity of the off state is determined by the POLA0 bit.

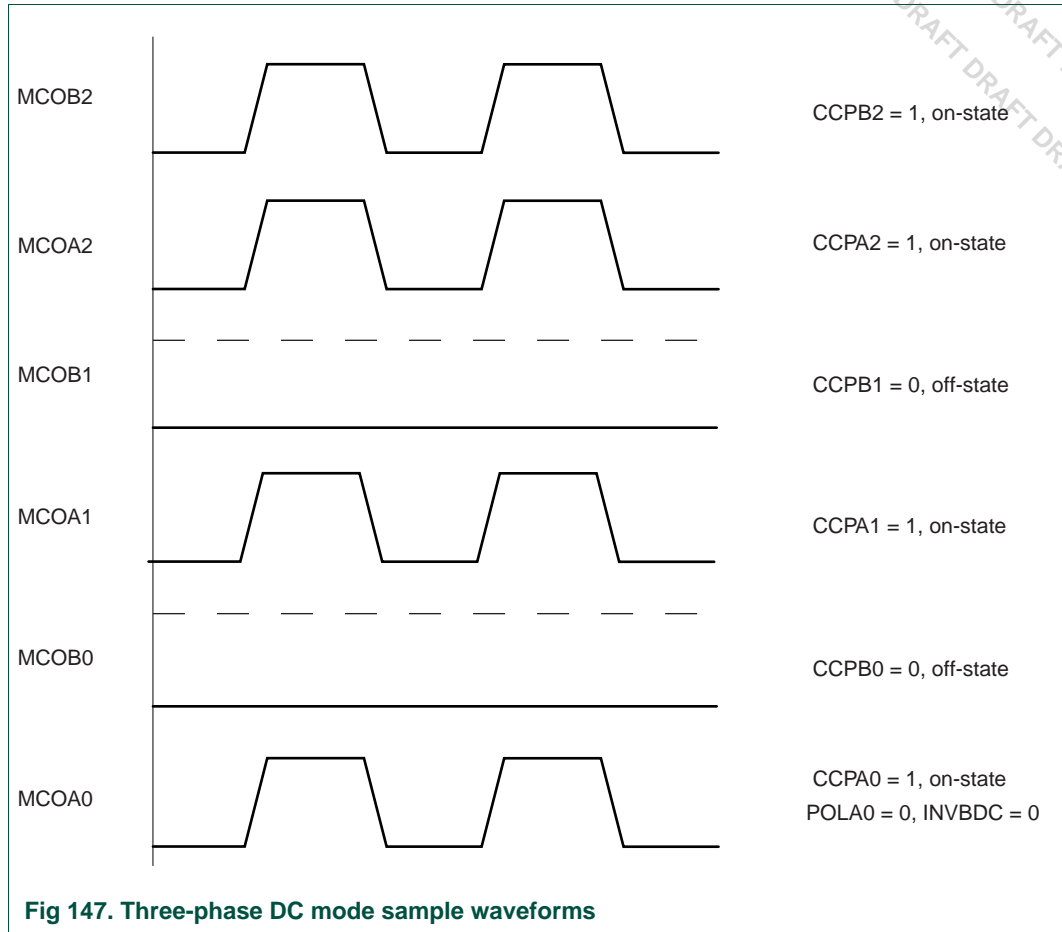
All output pins that have 1 bits in the MCCP register are controlled by the internal MC\_0A signal.

The three MC\_B output pins are inverted when the INVBDC bit is 1 in the MCCON register. This feature accommodates bridge-drivers that have active-low inputs for the low-side switches.

The MCCP register is implemented as a shadow register pair, so that changes to the active commutation pattern occur at the beginning of a new PWM cycle. See [27.8.6](#) and [27.9.2](#) for more about writing and reading such registers.

[Figure 147](#) shows sample waveforms of the output pins in three-phase DC mode. Bits 1 and 3 in the MCCP register (corresponding to outputs MC\_1B and MC\_0B) are set to 0 so that these outputs are masked and in the off state. Their logic level is determined by the POLA0 bit (here, POLA0 = 0 so the passive state is logic LOW). The INVBDC bit is set to 0 (logic level not inverted) so that the B output have the same polarity as the A outputs. Note that this mode differs from other modes in that the MC\_B outputs are **not** the opposite of the MC\_A outputs.

In the situation shown in [Figure 147](#), bits 0, 2, 4, and 5 in the MCCP register are set to 1. That means that MC\_1A and both output pins for channel 2 follow the MC\_0A signal.



### 27.9.7 Three phase AC mode

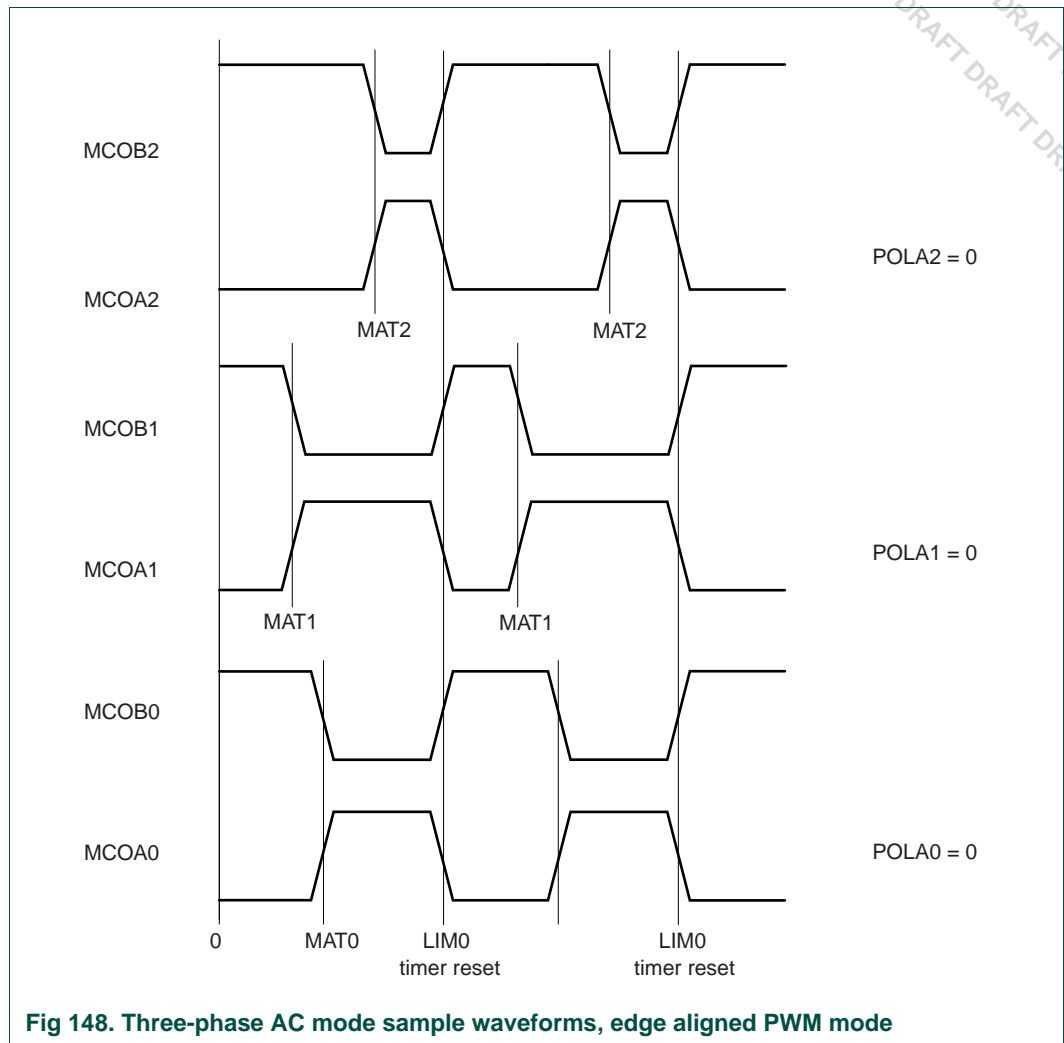
The three-phase AC-mode is selected by setting the ACMODE bit in the MCON register.

In this mode, the value of channel 0's TC is routed to all channels for comparison with their MAT registers. (The LIM1-2 registers are not used.)

Each channel controls its output pins by comparing its MAT value to TC0.

[Figure 148](#) shows sample waveforms for the six output pins in three-phase AC mode. The POLA bits are set to 0 for all three channels, so that for all output pins the active levels are high and the passive levels are low. Each channel has a different MAT value which is compared to the MCTC0 value. In this mode the period value is identical for all three channels and is determined by MCLIM0. The dead-time mode is disabled.





### 27.9.8 Interrupts

The MCPWM includes 10 possible interrupt sources:

- When any channel’s TC matches its Match register.
- When any channel’s TC matches its Limit register.
- When any channel captures the value of its TC into its Capture register, because a selected edge occurs on any of MC\_FB0-2.
- When all three channels’ outputs are forced to “A passive” state because the MC\_ABORT pin goes low.

[Section 27.8.3 “MCPWM Interrupt registers”](#) explains how to enable these interrupts, and [Section 27.8.2 “MCPWM Capture Control register”](#) describes how to map edges on the MC\_FB0-2 inputs to “capture events” on the three channels.

### 28.1 Basic configuration

---

The QEI is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCQEI.  
**Remark:** On reset, the QEI is disabled (PCQEI = 0).
2. Peripheral clock: The QEI operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Select QEI pins through and pin modes for port pins with QEI functions through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: See [Section 28.6.4](#). The QEI interrupt is enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 28.2 Features

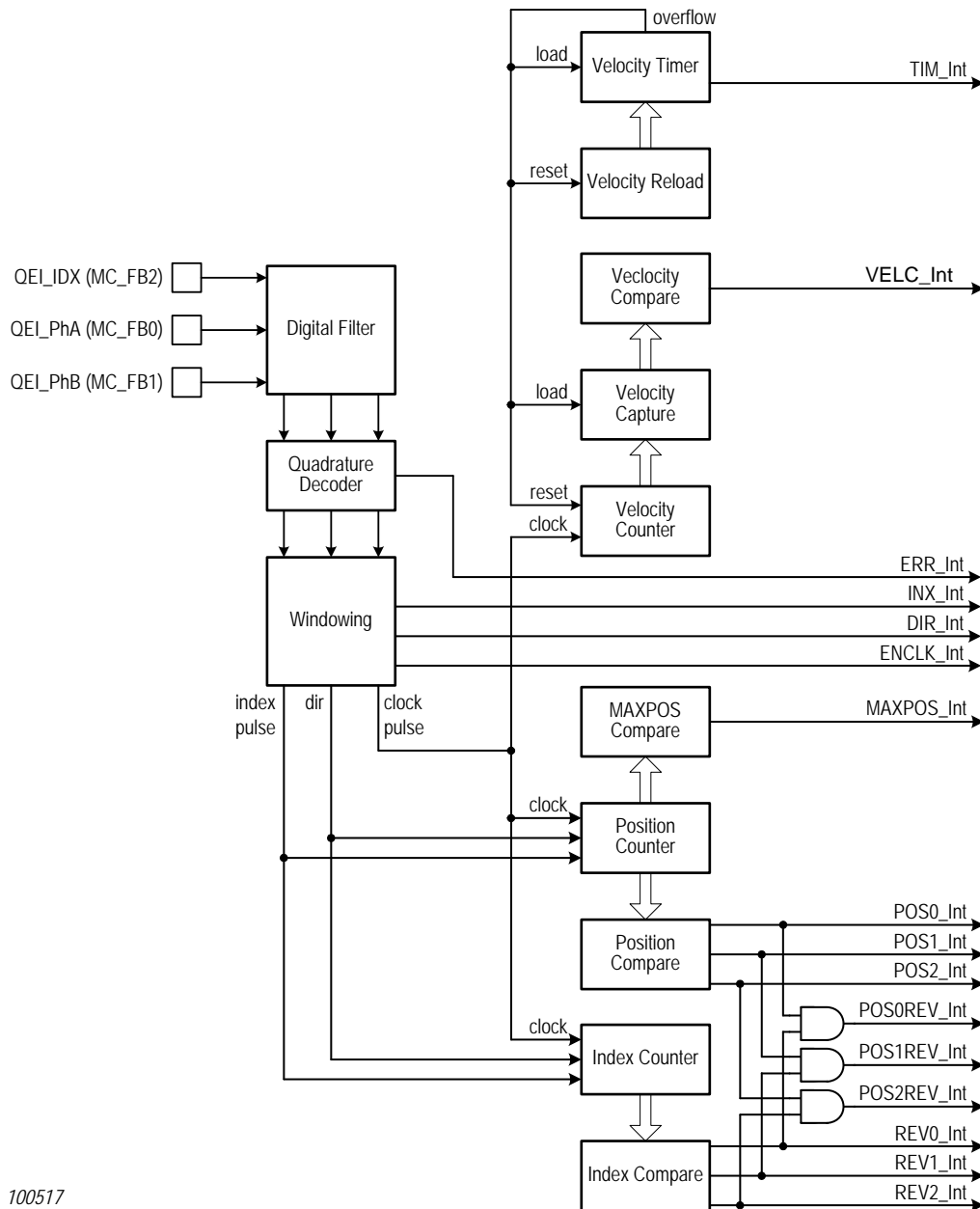
---

This Quadrature Encoder Interface (QEI) has the following features:

- tracks encoder position.
- increments/ decrements depending on direction.
- programmable for 2X or 4X position counting.
- velocity capture using built-in timer.
- velocity compare function with less than interrupt.
- uses 32-bit registers for position and velocity.
- three position compare registers with interrupts.
- index counter for revolution counting.
- index compare register with interrupts.
- can combine index and position interrupts to produce an interrupt for whole and partial revolution displacement.
- digital filter with programmable delays for encoder input signals.
- can accept decoded signal inputs (clock and direction).

28.3 Introduction

A quadrature encoder, also known as a 2-channel incremental encoder, converts angular displacement into two pulse signals. By monitoring both the number of pulses and the relative phase of the two signals, position, direction of rotation, and velocity can be tracked. In addition, a third channel, or index signal, can be used to reset the position counter. This quadrature encoder interface module decodes the digital pulses from a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.



100517

Fig 149. Encoder interface block diagram

## 28.4 Functional description

The QEI module interprets the two-bit gray code produced by a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.

### 28.4.1 Input signals

The QEI module supports two modes of signal operation: quadrature phase mode and clock/direction mode. In quadrature phase mode, the encoder produces two clocks that are 90 degrees out of phase; the edge relationship is used to determine the direction of rotation. In clock/direction mode, the encoder produces a clock signal to indicate steps and a direction signal to indicate the direction of rotation.).

This mode is determined by the SigMode bit of the QEI Configuration register (QEICONF) register (See [Table 591](#)). When the SigMode bit = 1, the quadrature decoder is bypassed and the PhA pin functions as the direction signal and PhB pin functions as the clock signal for the counters, etc. When the SigMode bit = 0, the PhA pin and PhB pins are decoded by the quadrature decoder. In this mode the quadrature decoder produces the direction and clock signals for the counters, etc. In both modes the direction signal is subject to the effects of the direction invert (DIRINV) bit.

#### 28.4.1.1 Quadrature input signals

When edges on PhA lead edges on PhB, the position counter is incremented. When edges on PhB lead edges on PhA, the position counter is decremented. When a rising and falling edge pair is seen on one of the phases without any edges on the other, the direction of rotation has changed.

**Table 585. Encoder states**

Phase A	Phase B	state
1	0	1
1	1	2
0	1	3
0	0	4

**Table 586. Encoder state transitions<sup>[1]</sup>**

from state	to state	Direction
1	2	positive
2	3	
3	4	
4	1	
4	3	negative
3	2	
2	1	
1	4	

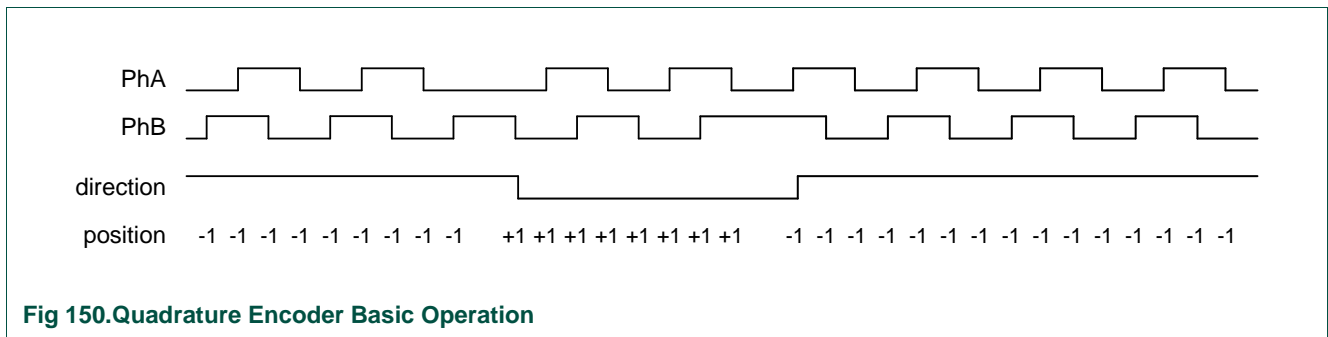
[1] All other state transitions are illegal and should set the ERR bit.

Interchanging of the PhA and PhB input signals are compensated by complementing the DIR bit. When set = 1, the direction inversion bit (DIRINV) complements the DIR bit.

**Table 587. Encoder direction**

DIR bit	DIRINV bit	direction
0	0	forward
1	0	reverse
0	1	reverse
1	1	forward

Figure 150 shows how quadrature encoder signals equate to direction and count.



**Fig 150. Quadrature Encoder Basic Operation**

**28.4.1.2 Digital input filtering**

All three encoder inputs (PhA, PhB, and index) require digital filtering. The number of sample clocks is user programmable from 1 to 4,294,967,295 (0xFFFF FFFF). In order for a transition to be accepted, the input signal must remain in new state for the programmed number of sample clocks.

**28.4.2 Position capture**

The capture mode for the position integrator can be set to update the position counter on every edge of the PhA signal or to update on every edge of both PhA and PhB. Updating the position counter on every PhA and PhB provides more positional resolution at the cost of less range in the positional counter.

The position integrator and velocity capture can be independently enabled. Alternatively, the phase signals can be interpreted as a clock and direction signal as output by some encoders.

The position counter is automatically reset on one of three conditions. Incrementing past the maximum position value (QEIMAXPOS) will reset the position counter to zero. If the reset on index bit (RESPI) is set, sensing the index pulse for the first time will once reset the position counter to zero after the next positional increase (calibrate). If the continuously reset on index bit (CRESPI) is set, sensing the index pulse will continuously reset the position counter to zero after the next positional increase (recalibrate).

### 28.4.3 Velocity capture

The velocity capture has a programmable timer and a capture register. It counts the number of phase edges (using the same configuration as for the position integrator) in a given time period. When the velocity timer (QEITIME) overflows the contents of the velocity counter (QEIVEL) are transferred to the capture (QEICAP) register. The velocity counter is then cleared. The velocity timer is loaded with the contents of the velocity reload register (QEILOAD). Finally, the velocity interrupt (TIM\_Int) is asserted. The number of edges counted in a given time period is directly proportional to the velocity of the encoder. Note that the velocity counter counts up regardless of the direction of rotation, and whether the direction changes.

Setting the reset velocity bit (RESV) will clear the velocity counter, reset the velocity capture register to 0xFFFF FFFF, load the velocity timer with the contents of the velocity reload register (QEILOAD).

The following equation converts the velocity counter value into an RPM value:

$$\text{RPM} = (\text{PCLK} * \text{QEICAP} * 60) \div (\text{QEILOAD} * \text{PPR} * \text{Edges})$$

where:

- **PCLK** is the peripheral clock rate for the QEI block. See [Section 4.6.4](#) for more on the possibilities for PCLK).
- **QEICAP** is the captured velocity counter value for the last velocity timer period.
- **QEILOAD** is the velocity timer reload value.
- **PPR** is the number of pulses per revolution of the physical encoder used in the application
- **Edges** is 2 or 4, based on the capture mode set in the QEICON register (2 for CapMode set to 0 and 4 for CapMode set to 1)

For example, consider a motor running at 600 RPM. A 2048 pulse per revolution quadrature encoder is attached to the motor, producing 8192 phase edges per revolution (PPR \* Edges). This results in 81,920 pulses per second (the motor turns 10 times per second at 600 RPM and there are 8192 edges per revolution). If the timer were clocked at 10,000 Hz, and the QEILOAD was 2,500 (corresponding to ¼ of a second), it would count 20,480 pulses per update. Using the above equation:

$$\text{RPM} = (10000 * 1 * 20480 * 60) \div (2500 * 2048 * 4) = 600 \text{ RPM}$$

Now, consider that the motor is sped up to 3000 RPM. This results in 409,600 pulses per second, or 102,400 every ¼ of a second. Again, the above equation gives:

$$\text{RPM} = (10000 * 1 * 102400 * 60) \div (2500 * 2048 * 4) = 3000 \text{ RPM}$$

These are simple examples, real-world values will have a higher rate for PCLK, and probably a larger value for QEILOAD as well.

### 28.4.4 Velocity compare

In addition to velocity capture, the velocity measurement system includes a programmable velocity compare register. After every velocity capture event the contents of the velocity capture register (QEICAP) is compared with the contents of the velocity

compare register (VELCOMP). If the captured velocity is less than the compare value, an interrupt is asserted provided that the velocity compare interrupt enable bit is set. This can be used to determine if a motor shaft is either stalled or moving too slow.

## 28.5 Pin description

**Table 588. QEI pin description**

Pin name	I/O	Description
MC_FB0 <a href="#">[1]</a>	I	Used as the Phase A (PhA) input to the Quadrature Encoder Interface.
MC_FB1 <a href="#">[1]</a>	I	Used as the Phase B (PhB) input to the Quadrature Encoder Interface.
MC_FB2 <a href="#">[1]</a>	I	Used as the Index (IDX) input to the Quadrature Encoder Interface.

[1] The Quadrature Encoder Interface uses the same pin functions as the Motor Control PWM feedback inputs. If used as part of motor control, the QEI can be used as an alternative to feedback directly to the MCPWM.

## 28.6 Register description

### 28.6.1 Register summary

Table 589. Register summary

Symbol	Description	R/W	Reset Value	Address	Table
<b>Control registers</b>					
QEICON	Control register	WO	NA	0x400B C000	<a href="#">590</a>
QEICONF	Configuration register	R/W	0	0x400B C008	<a href="#">591</a>
QEISTAT	Status register	RO	0	0x400B C004	<a href="#">592</a>
<b>Position, index, and timer registers</b>					
QEIPOS	Position register	RO	0	0x400B C00C	<a href="#">593</a>
QEIMAXPOS	Maximum position register	R/W	0	0x400B C010	<a href="#">594</a>
CMPOS0	Position compare register 0	R/W	0xFFFF FFFF	0x400B C014	<a href="#">595</a>
CMPOS1	Position compare register 1	R/W	0xFFFF FFFF	0x400B C018	<a href="#">596</a>
CMPOS2	Position compare register 2	R/W	0xFFFF FFFF	0x400B C01C	<a href="#">597</a>
INXCNT	Index count register 0	RO	0	0x400B C020	<a href="#">598</a>
INXCMP0	Index compare register 0	R/W	0xFFFF FFFF	0x400B C024	<a href="#">599</a>
INXCMP1	Index compare register 1	R/W	0xFFFF FFFF	0x400B C04C	<a href="#">600</a>
INXCMP2	Index compare register 2	R/W	0xFFFF FFFF	0x400B C050	<a href="#">601</a>
QEILOAD	Velocity timer reload register	R/W	0	0x400B C028	<a href="#">602</a>
QEITIME	Velocity timer register	RO	0	0x400B C02C	<a href="#">603</a>
QEIVEL	Velocity counter register	RO	0	0x400B C030	<a href="#">604</a>
QEICAP	Velocity capture register	RO	0xFFFF FFFF	0x400B C034	<a href="#">605</a>
VELCOMP	Velocity compare register	R/W	0	0x400B C038	<a href="#">606</a>
FILTERPHA	Digital filter register on PHA	R/W	0	0x400B C03C	<a href="#">607</a>
FILTERPHB	Digital filter register on PHB	R/W	0	0x400B C040	<a href="#">608</a>
FILTERINX	Digital filter register on IDX	R/W	0	0x400B C044	<a href="#">609</a>
WINDOW	Index acceptance window register	R/W	0xF	0x400B C048	<a href="#">610</a>
<b>Interrupt registers</b>					
QEIINTSTAT	Interrupt status register	RO	0	0x400B CFE0	<a href="#">611</a>
QEISET	Interrupt status set register	WO	NA	0x400B CFEC	<a href="#">612</a>
QEICLR	Interrupt status clear register	WO	NA	0x400B CFE8	<a href="#">613</a>
QEIIE	Interrupt enable register	RO	0	0x400B CFE4	<a href="#">614</a>
QEIIES	Interrupt enable set register	WO	NA	0x400B CFDC	<a href="#">615</a>
QEIEEC	Interrupt enable clear register	WO	NA	0x400B CFD8	<a href="#">616</a>



28.6.2 Control registers

28.6.2.1 QEI Control register (QEICON - 0x400B C000)

This register contains bits which control the operation of the position and velocity counters of the QEI module.

Table 590: QEI Control register (QEICON - address 0x400B C000) bit description

Bit	Symbol	Description	Reset value
0	RESP	Reset position counter. When set = 1, resets the position counter to all zeros. Autoclears when the position counter is cleared.	0
1	RESPI	Reset position counter on index. When set = 1, resets the position counter to all zeros once only the first time an index pulse occurs. Autoclears when the position counter is cleared.	0
2	RESV	Reset velocity. When set = 1, resets the velocity counter to all zeros, reloads the velocity timer, and presets the velocity compare register. Autoclears when the velocity counter is cleared.	0
3	RESI	Reset index counter. When set = 1, resets the index counter to all zeros. Autoclears when the index counter is cleared.	0
31:4	-	Reserved. Read value is undefined, only zero should be written.	NA

28.6.2.2 QEI Configuration register (QEICONF - 0x400B C008)

This register contains the configuration of the QEI module.

Table 591: QEI Configuration register (QEICONF - address 0x400B C008) bit description

Bit	Symbol	Description	Reset value
0	DIRINV	Direction invert. When 1, complements the DIR bit.	0
1	SIGMODE	Signal Mode. When 0, PhA and PhB function as quadrature encoder inputs. When 1, PhA functions as the direction signal and PhB functions as the clock signal.	0
2	CAPMODE	Capture Mode. When 0, only PhA edges are counted (2X). When 1, BOTH PhA and PhB edges are counted (4X), increasing resolution but decreasing range.	0
3	INVINX	Invert Index. When 1, inverts the sense of the index input.	0
4	CRESPI	Continuously reset the position counter on index. When 1, resets the position counter to all zeros whenever an index pulse occurs after the next position increase (recalibration).	0
15:5	-	Reserved. Read value is undefined, only zero should be written.	NA
19:16	INXGATE	Index gating configuration: When INXGATE[16] = 1, pass the index when PHA = 1 and PHB = 0, otherwise block index. When INXGATE[17] = 1, pass the index when PHA = 1 and PHB = 1, otherwise block index. When INXGATE[18] = 1, pass the index when PHA = 0 and PHB = 1, otherwise block index. When INXGATE[19] = 1, pass the index when PHA = 0 and PHB = 0, otherwise block index.	0xF
31:20	-	Reserved. Read value is undefined, only zero should be written.	NA

**28.6.2.3 QEI Status register (QEISTAT - 0x400B C004)**

This register provides the status of the encoder interface.

**Table 592: QEI Status register (QEISTAT - address 0x400B C004) bit description**

Bit	Symbol	Description	Reset value
0	DIR	Direction bit. In combination with DIRINV bit indicates forward or reverse direction. See <a href="#">Table 587</a> .	
31:1	-	Reserved. Read value is undefined, only zero should be written.	NA

**28.6.3 Position, index and timer registers**

**28.6.3.1 QEI Position register (QEIPOS - 0x400B C00C)**

This register contains the current value of the encoder position. Increments or decrements when encoder counts occur, depending on the direction of rotation.

**Table 593: QEI Position register (QEIPOS - address 0x400B C00C) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current position value.	0

**28.6.3.2 QEI Maximum Position register (QEIMAXPOS - 0x400B C010)**

This register contains the maximum value of the encoder position. In forward rotation the position register resets to zero when the position register exceeds this value. In reverse rotation the position register resets to this value when the position register decrements from zero.

**Table 594: QEI Maximum Position register (QEIMAXPOS - address 0x400B C010) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current maximum position value.	0

**28.6.3.3 QEI Position Compare register 0 (CMPOS0 - 0x400B C014)**

This register contains a position compare value. This value is compared against the current value of the position register. An interrupt can be generated when the compare value is equal to the current value of the position register. The compare value must take into account the fact that the starting position is zero.

**Table 595: QEI Position Compare register 0 (CMPOS0 - address 0x400B C014) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Position compare value 0.	0

**28.6.3.4 QEI Position Compare register 1 (CMPOS1 - 0x400B C018)**

This register contains a position compare value. This value is compared against the current value of the position register. An interrupt can be generated when the compare value is equal to the current value of the position register. The compare value must take into account the fact that the starting position is zero.

**Table 596: QEI Position Compare register 1 (CMPOS1 - address 0x400B C018) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Position compare value 1.	0

**28.6.3.5 QEI Position Compare register 2 (CMPOS2 - 0x400B C01C)**

This register contains a position compare value. This value is compared against the current value of the position register. An interrupt can be generated when the compare value is equal to the current value of the position register. The compare value must take into account the fact that the starting position is zero.

**Table 597: QEI Position Compare register 2 (CMPOS2 - address 0x400B C01C) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Position compare value 2.	0

**28.6.3.6 QEI Index Count register (INXCNT - 0x400B C020)**

This register contains the current value of the index counter. It is updated when an index count occurs. This can be an increment when the position counter overflows the MAXPOS value or a decrement when the position counter underflows zero, depending on the direction of rotation. In case (re)calibration occurs due to an index pulse, the over/underflow is forced internally.

**Table 598: QEI Index Count register (INXCNT - address 0x400B C020) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current index counter value.	0

**28.6.3.7 QEI Index Compare register 0 (INXCMP0 - 0x400B C024)**

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

**Table 599: QEI Index Compare register 0 (INXCMP0 - address 0x400B C024) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Index compare value 0.	0

**28.6.3.8 QEI Index Compare register 1 (INXCMP1 - 0x400B C04C)**

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

**Table 600: QEI Index Compare register 1 (INXCMP1 - address 0x400B C04C) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Index compare value 1.	0

**28.6.3.9 QEI Index Compare register 2 (INXCMP2 - 0x400B C050)**

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

**Table 601: QEI Index Compare register 2 (INXCMP2 - address 0x400B C050) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Index compare value 2.	0

**28.6.3.10 QEI Velocity Timer Reload register (QEILOAD - 0x400B C028)**

This register contains the reload value of the velocity timer. When the timer (QEITIME) overflows or the RESV bit is asserted, this value is loaded into the timer (QEITIME).

**Table 602: QEI Timer Load register (QEILOAD - address 0x400B C028) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current velocity timer load value.	0

**28.6.3.11 QEI Velocity Timer register (QEITIME - 0x400B C02C)**

This register contains the current value of the velocity timer. When this timer overflows the value of velocity counter (QEIVEL) is stored in the velocity capture register (QEICAP), the velocity counter is reset to zero, the timer is reloaded with the value stored in the velocity reload register (QEILOAD), and the velocity interrupt (TIM\_Int) is asserted.

**Table 603: QEI Timer register (QEITIME - address 0x400B C02C) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current velocity timer value.	0

**28.6.3.12 QEI Velocity register (QEIVEL - 0x400B C030)**

This register contains the running count of velocity pulses for the current time period. When the velocity timer (QEITIME) overflows the contents of this register is captured in the velocity capture register (QEICAP). After capture, this register is set to zero. This register is also reset when the velocity reset bit (RESV) is asserted.

**Table 604: QEI Velocity register (QEIVEL - address 0x400B C030) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Current velocity pulse count.	0

**28.6.3.13 QEI Velocity Capture register (QEICAP - 0x400B C034)**

This register contains the most recently measured velocity of the encoder. This corresponds to the number of velocity pulses counted in the previous velocity timer period. The current velocity count is latched into this register when the velocity timer overflows.

**Table 605: QEI Velocity Capture register (QEICAP - address 0x400B C034) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Last velocity capture.	0

**28.6.3.14 QEI Velocity Compare register (VELCOMP - 0x400B C038)**

This register contains a velocity compare value. This value is compared against the captured velocity in the velocity capture register. If the capture velocity is less than the value in this compare register, a velocity compare interrupt (VELC\_Int) will be asserted, if enabled.

**Table 606: QEI Velocity Compare register (VELCOMP - address 0x400B C038) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Compare velocity pulse count.	0

**28.6.3.15 QEI Digital Filter on PHA (FILTERPHA - 0x400B C03C)**

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

**Table 607: QEI Digital Filter ON PHA (FILTERPHA - address 0x400B C03C) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Digital filter sampling delay for PhA.	0

**28.6.3.16 QEI Digital Filter on PHB (FILTERPHB - 0x400B C040)**

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

**Table 608: QEI Digital Filter on PHB (FILTERPHB - address 0x400B C040) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Digital filter sampling delay for PhB.	0

**28.6.3.17 QEI Digital Filter on INX (FILTERINX - 0x400B C044)**

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

**Table 609: QEI Digital Filter on INX (FILTERINX - address 0x400B C044) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Digital filter sampling delay for the index.	0

**28.6.3.18 QEI index acceptance Window (WINDOW - 0x400B C048)**

This register contains the width of the index acceptance window, when the index and the phase/clock edges fall nearly together. If the activating phase/clock edge falls before the index, but within the window, the (re)calibration will be activated on that phase/clock edge.

**Table 610: QEI index acceptance Window (WINDOW - address 0x400B C048) bit description**

Bit	Symbol	Description	Reset value
31:0	-	Index acceptance window width.	0xF

28.6.4 Interrupt registers

28.6.4.1 QEI Interrupt Status register (QEINTSTAT - 0x400B CFE0)

This register provides the status of the encoder interface and the current set of interrupt sources that are asserted to the controller. Bits set to 1 indicate the latched events that have occurred; a zero bit indicates that the event in question has not occurred.

Table 611: QEI Interrupt Status register (QEINTSTAT - address 0x400B CFE0) bit description

Bit	Symbol	Description	Reset value
0	INX_Int	Indicates that an index pulse was detected.	0
1	TIM_Int	Indicates that a velocity timer overflow occurred	0
2	VELC_Int	Indicates that captured velocity is less than compare velocity.	0
3	DIR_Int	Indicates that a change of direction was detected.	0
4	ERR_Int	Indicates that an encoder phase error was detected.	0
5	ENCLK_Int	Indicates that an encoder clock pulse was detected.	
6	POS0_Int	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_Int	Indicates that the position 1 compare value is equal to the current position.	0
8	POS2_Int	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_Int	Indicates that the index compare 0 value is equal to the current index count.	0
10	POS0REV_Int	Combined position 0 and revolution count interrupt. Set when both the POS0_Int bit is set and the REV0_Int is set.	0
11	POS1REV_Int	Combined position 1 and revolution count interrupt. Set when both the POS1_Int bit is set and the REV1_Int is set.	0
12	POS2REV_Int	Combined position 2 and revolution count interrupt. Set when both the POS2_Int bit is set and the REV2_Int is set.	0
13	REV1_Int	Indicates that the index compare 1 value is equal to the current index count.	0
14	REV2_Int	Indicates that the index compare 2 value is equal to the current index count.	0
15	MAXPOS_Int	Indicates that the current position count goes through the MAXPOS value to zero in the forward direction, or through zero to MAXPOS in the reverse direction.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

**28.6.4.2 QEI Interrupt Set register (QEISET - 0x400B CFEC)**

Writing a one to a bit in this register sets the corresponding bit in the QEI Interrupt Status register (QEISTAT).

**Table 612: QEI Interrupt Set register (QEISET - address 0x400B CFEC) bit description**

Bit	Symbol	Description
0	INX_Int	Writing a 1 sets the INX_Int bit in QEIINTSTAT.
1	TIM_Int	Writing a 1 sets the TIN_Int bit in QEIINTSTAT.
2	VELC_Int	Writing a 1 sets the VELC_Int bit in QEIINTSTAT.
3	DIR_Int	Writing a 1 sets the DIR_Int bit in QEIINTSTAT.
4	ERR_Int	Writing a 1 sets the ERR_Int bit in QEIINTSTAT.
5	ENCLK_Int	Writing a 1 sets the ENCLK_Int bit in QEIINTSTAT.
6	POS0_Int	Writing a 1 sets the POS0_Int bit in QEIINTSTAT.
7	POS1_Int	Writing a 1 sets the POS1_Int bit in QEIINTSTAT.
8	POS2_Int	Writing a 1 sets the POS2_Int bit in QEIINTSTAT.
9	REV0_Int	Writing a 1 sets the REV0_Int bit in QEIINTSTAT.
10	POS0REV_Int	Writing a 1 sets the POS0REV_Int bit in QEIINTSTAT.
11	POS1REV_Int	Writing a 1 sets the POS1REV_Int bit in QEIINTSTAT.
12	POS2REV_Int	Writing a 1 sets the POS2REV_Int bit in QEIINTSTAT.
13	REV1_Int	Writing a 1 sets the REV1_Int bit in QEIINTSTAT.
14	REV2_Int	Writing a 1 sets the REV2_Int bit in QEIINTSTAT.
15	MAXPOS_Int	Writing a 1 sets the MAXPOS_Int bit in QEIINTSTAT.
31:16	-	Reserved. Read value is undefined, only zero should be written.

**28.6.4.3 QEI Interrupt Clear register (QEICLR - 0x400B CFE8)**

Writing a one to a bit in this register clears the corresponding bit in the QEI Interrupt Status register (QEIINTSTAT).

**Table 613: QEI Interrupt Clear register (QEICLR - 0x400B CFE8) bit description**

Bit	Symbol	Description
0	INX_Int	Writing a 1 clears the INX_Int bit in QEIINTSTAT.
1	TIM_Int	Writing a 1 clears the TIN_Int bit in QEIINTSTAT.
2	VELC_Int	Writing a 1 clears the VELC_Int bit in QEIINTSTAT.
3	DIR_Int	Writing a 1 clears the DIR_Int bit in QEIINTSTAT.
4	ERR_Int	Writing a 1 clears the ERR_Int bit in QEIINTSTAT.
5	ENCLK_Int	Writing a 1 clears the ENCLK_Int bit in QEIINTSTAT.
6	POS0_Int	Writing a 1 clears the POS0_Int bit in QEIINTSTAT.
7	POS1_Int	Writing a 1 clears the POS1_Int bit in QEIINTSTAT.
8	POS2_Int	Writing a 1 clears the POS2_Int bit in QEIINTSTAT.
9	REV0_Int	Writing a 1 clears the REV0_Int bit in QEIINTSTAT.
10	POS0REV_Int	Writing a 1 clears the POS0REV_Int bit in QEIINTSTAT.
11	POS1REV_Int	Writing a 1 clears the POS1REV_Int bit in QEIINTSTAT.
12	POS2REV_Int	Writing a 1 clears the POS2REV_Int bit in QEIINTSTAT.
13	REV1_Int	Writing a 1 clears the REV1_Int bit in QEIINTSTAT.
14	REV2_Int	Writing a 1 clears the REV2_Int bit in QEIINTSTAT.
15	MAXPOS_Int	Writing a 1 clears the MAXPOS_Int bit in QEIINTSTAT.
31:16	-	Reserved. Read value is undefined, only zero should be written.



**28.6.4.4 QEI Interrupt Enable register (QEIE - 0x400B CFE4)**

This register enables interrupt sources. Bits set to 1 enable the corresponding interrupt; a zero bit disables the corresponding interrupt.

**Table 614: QEI Interrupt Enable register (QEIE - address 0x400B CFE4) bit description**

Bit	Symbol	Description	Reset value
0	INX_Int	When 1, the INX_Int interrupt is enabled.	0
1	TIM_Int	When 1, the TIN_Int interrupt is enabled.	0
2	VELC_Int	When 1, the VELC_Int interrupt is enabled.	0
3	DIR_Int	When 1, the DIR_Int interrupt is enabled.	0
4	ERR_Int	When 1, the ERR_Int interrupt is enabled.	0
5	ENCLK_Int	When 1, the ENCLK_Int interrupt is enabled.	0
6	POS0_Int	When 1, the POS0_Int interrupt is enabled.	0
7	POS1_Int	When 1, the POS1_Int interrupt is enabled.	0
8	POS2_Int	When 1, the POS2_Int interrupt is enabled.	0
9	REV0_Int	When 1, the REV0_Int interrupt is enabled.	0
10	POS0REV_Int	When 1, the POS0REV_Int interrupt is enabled.	0
11	POS1REV_Int	When 1, the POS1REV_Int interrupt is enabled.	0
12	POS2REV_Int	When 1, the POS2REV_Int interrupt is enabled.	0
13	REV1_Int	When 1, the REV1_Int interrupt is enabled.	0
14	REV2_Int	When 1, the REV2_Int interrupt is enabled.	0
15	MAXPOS_Int	When 1, the MAXPOS_Int interrupt is enabled.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

**28.6.4.5 QEI Interrupt Enable Set register (QEIIES - 0x400B CFDC)**

Writing a one to a bit in this register sets the corresponding bit in the QEI Interrupt Enable register (QEIIE).

**Table 615: QEI Interrupt Enable Set register (QEIIES - address 0x400B CFDC) bit description**

Bit	Symbol	Description
0	INX_Int	Writing a 1 enables the INX_Int interrupt in the QEIIE register.
1	TIM_Int	Writing a 1 enables the TIN_Int interrupt in the QEIIE register.
2	VELC_Int	Writing a 1 enables the VELC_Int interrupt in the QEIIE register.
3	DIR_Int	Writing a 1 enables the DIR_Int interrupt in the QEIIE register.
4	ERR_Int	Writing a 1 enables the ERR_Int interrupt in the QEIIE register.
5	ENCLK_Int	Writing a 1 enables the ENCLK_Int interrupt in the QEIIE register.
6	POS0_Int	Writing a 1 enables the POS0_Int interrupt in the QEIIE register.
7	POS1_Int	Writing a 1 enables the POS1_Int interrupt in the QEIIE register.
8	POS2_Int	Writing a 1 enables the POS2_Int interrupt in the QEIIE register.
9	REV0_Int	Writing a 1 enables the REV0_Int interrupt in the QEIIE register.
10	POS0REV_Int	Writing a 1 enables the POS0REV_Int interrupt in the QEIIE register.
11	POS1REV_Int	Writing a 1 enables the POS1REV_Int interrupt in the QEIIE register.
12	POS2REV_Int	Writing a 1 enables the POS2REV_Int interrupt in the QEIIE register.
13	REV1_Int	Writing a 1 enables the REV1_Int interrupt in the QEIIE register.
14	REV2_Int	Writing a 1 enables the REV2_Int interrupt in the QEIIE register.
15	MAXPOS_Int	Writing a 1 enables the MAXPOS_Int interrupt in the QEIIE register.
31:16	-	Reserved. Read value is undefined, only zero should be written.

#### 28.6.4.6 QEI Interrupt Enable Clear register (QEIEC - 0x400B CFD8)

Writing a one to a bit in this register clears the corresponding bit in the QEI Interrupt Enable register (QEIE).

**Table 616: QEI Interrupt Enable Clear register (QEIEC - address 0x400B CFD8) bit description**

Bit	Symbol	Description
0	INX_Int	Writing a 1 disables the INX_Int interrupt in the QEIE register.
1	TIM_Int	Writing a 1 disables the TIN_Int interrupt in the QEIE register.
2	VELC_Int	Writing a 1 disables the VELC_Int interrupt in the QEIE register.
3	DIR_Int	Writing a 1 disables the DIR_Int interrupt in the QEIE register.
4	ERR_Int	Writing a 1 disables the ERR_Int interrupt in the QEIE register.
5	ENCLK_Int	Writing a 1 disables the ENCLK_Int interrupt in the QEIE register.
6	POS0_Int	Writing a 1 disables the POS0_Int interrupt in the QEIE register.
7	POS1_Int	Writing a 1 disables the POS1_Int interrupt in the QEIE register.
8	POS2_Int	Writing a 1 disables the POS2_Int interrupt in the QEIE register.
9	REV0_Int	Writing a 1 disables the REV0_Int interrupt in the QEIE register.
10	POS0REV_Int	Writing a 1 disables the POS0REV_Int interrupt in the QEIE register.
11	POS1REV_Int	Writing a 1 disables the POS1REV_Int interrupt in the QEIE register.
12	POS2REV_Int	Writing a 1 disables the POS2REV_Int interrupt in the QEIE register.
13	REV1_Int	Writing a 1 disables the REV1_Int interrupt in the QEIE register.
14	REV2_Int	Writing a 1 disables the REV2_Int interrupt in the QEIE register.
15	MAXPOS_Int	Writing a 1 disables the MAXPOS_Int interrupt in the QEIE register.
31:16	-	Reserved. Read value is undefined, only zero should be written.

### 29.1 Basic configuration

---

The RTC is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCRTC.

**Remark:** On reset, the RTC is enabled. See [Section 29.7](#) for power saving options.

2. Clock: The RTC uses the 1 Hz clock output from the RTC oscillator as the internal function clock. The peripheral clock is used for accessing RTC registers.
3. Interrupts: See [Section 29.6.1](#) for RTC interrupt handling. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

### 29.2 Features

---

- Measures the passage of time to maintain a calendar and clock. Provides seconds, minutes, hours, day of month, month, year, day of week, and day of year.
- Ultra-low power design to support battery powered systems. Less than 1 microamp required for battery operation. Uses power from the CPU power supply when it is present.
- 20 bytes of Battery-backed storage and RTC operation when power is removed from the CPU.
- Dedicated 32 kHz ultra low power oscillator.
- Dedicated battery power supply pin.
- RTC power supply is isolated from the rest of the chip.
- Calibration counter allows adjustment to better than  $\pm 1$  sec/day with 1 sec resolution.
- Periodic interrupts can be generated from increments of any field of the time registers.
- Alarm interrupt can be generated for a specific date/time.

### 29.3 Description

---

The Real Time Clock (RTC) is a set of counters for measuring time when system power is on, and optionally when it is off. It uses very little power when its registers are not being accessed by the CPU, especially reduced power modes. On the LPC178x/177x, the RTC is clocked by a separate 32 kHz oscillator that produces a 1 Hz internal time reference. The RTC is powered by its own power supply pin, VBAT, which can be connected to a battery, externally tied to a 3V supply, or left floating.

The RTC power domain is shown in conceptual form in [Figure 151](#). A detailed view of the time keeping portion of the RTC is shown in [Figure 152](#).

29.4 Architecture

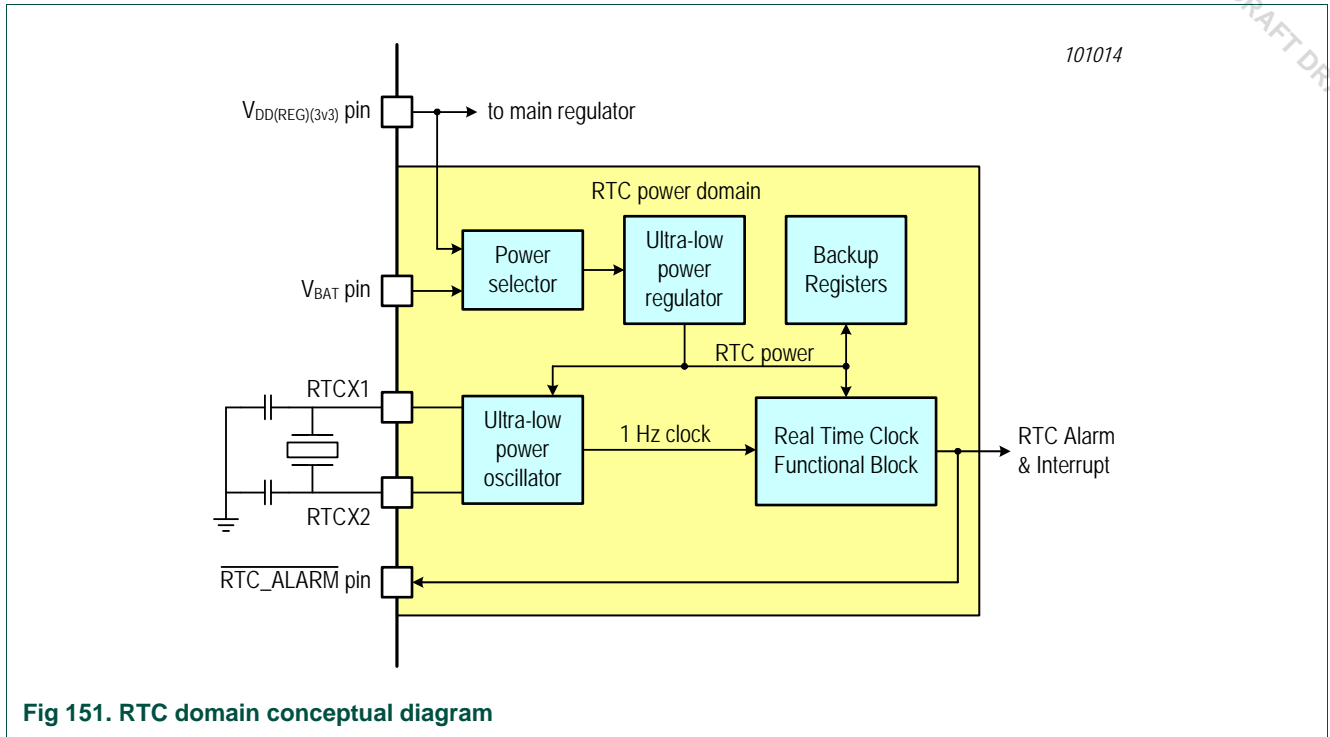


Fig 151. RTC domain conceptual diagram

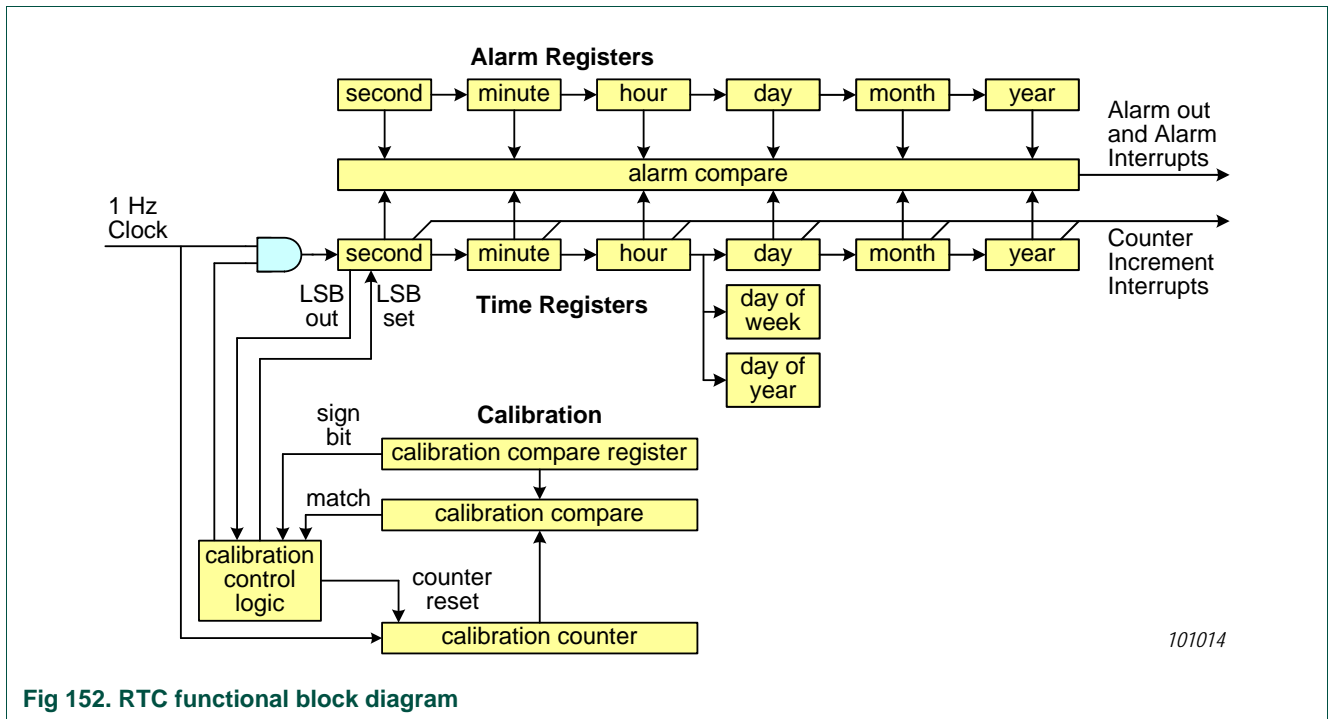


Fig 152. RTC functional block diagram

## 29.5 Pin description

Table 617. RTC pin description

Name	Type	Description
RTC_ALARM	Output	Alarm output from the RTC. This is an active low, open drain pin that goes LOW when the internal RTC alarm is asserted. The output is cleared by writing a 1 to the RTCALF bit in the Interrupt Location Register. This pin is powered by V <sub>BAT</sub> .
RTCX1	Input	Input to the RTC oscillator circuit.
RTCX2	Output	Output from the RTC oscillator circuit. <b>Remark:</b> If the RTC is not used, the RTCX1/2 pins can be left floating.
V <sub>BAT</sub>	Input	<b>RTC power supply:</b> Typically connected to an external 3V battery. If this pin is not powered, the RTC is still powered internally if V <sub>DD(REG)(3V3)</sub> is present.

## 29.6 Register description

The RTC includes a number of registers, shown in [Table 618](#). Detailed descriptions of the registers follow. In these descriptions, for most of the registers the Reset Value column shows "NC", meaning that these registers are Not Changed by a Reset. Software must initialize these registers between power-on and setting the RTC into operation. The registers are split into five sections by functionality.

Table 618. Real-Time Clock register map

Name	Description	Access	Reset Value <sup>[1]</sup>	Address	Table
<b>Miscellaneous registers (see Section 29.6.2)</b>					
ILR	Interrupt Location Register	R/W	0	0x4002 4000	<a href="#">619</a>
CCR	Clock Control Register	R/W	NC	0x4002 4008	<a href="#">619</a>
CIIR	Counter Increment Interrupt Register	R/W	0	0x4002 400C	<a href="#">621</a>
AMR	Alarm Mask Register	R/W	0	0x4002 4010	<a href="#">622</a>
RTC_AUX	RTC Auxiliary control register	R/W	0x8	0x4002 405C	<a href="#">623</a>
RTC_AUXEN	RTC Auxiliary Enable register	R/W	0	0x4002 4058	<a href="#">624</a>
<b>Consolidated time registers (see Section 29.6.3)</b>					
CTIME0	Consolidated Time Register 0	RO	NC	0x4002 4014	<a href="#">625</a>
CTIME1	Consolidated Time Register 1	RO	NC	0x4002 4018	<a href="#">626</a>
CTIME2	Consolidated Time Register 2	RO	NC	0x4002 401C	<a href="#">627</a>
<b>Time counter registers (see Section 29.6.4)</b>					
SEC	Seconds Counter	R/W	NC	0x4002 4020	<a href="#">629</a>
MIN	Minutes Register	R/W	NC	0x4002 4024	<a href="#">629</a>
HOUR	Hours Register	R/W	NC	0x4002 4028	<a href="#">629</a>
DOM	Day of Month Register	R/W	NC	0x4002 402C	<a href="#">629</a>
DOW	Day of Week Register	R/W	NC	0x4002 4030	<a href="#">629</a>
DOY	Day of Year Register	R/W	NC	0x4002 4034	<a href="#">629</a>
MONTH	Months Register	R/W	NC	0x4002 4038	<a href="#">629</a>
YEAR	Years Register	R/W	NC	0x4002 403C	<a href="#">629</a>
CALIBRATION	Calibration Value Register	R/W	NC	0x4002 4040	<a href="#">630</a>
<b>General purpose registers (see Section 29.6.6)</b>					
GPREG0	General Purpose Register 0	R/W	NC	0x4002 4044	<a href="#">631</a>
GPREG1	General Purpose Register 1	R/W	NC	0x4002 4048	<a href="#">631</a>
GPREG2	General Purpose Register 2	R/W	NC	0x4002 404C	<a href="#">631</a>
GPREG3	General Purpose Register 3	R/W	NC	0x4002 4050	<a href="#">631</a>
GPREG4	General Purpose Register 4	R/W	NC	0x4002 4054	<a href="#">631</a>
<b>Alarm register group (see Section 29.6.7)</b>					
ALSEC	Alarm value for Seconds	R/W	NC	0x4002 4060	<a href="#">632</a>
ALMIN	Alarm value for Minutes	R/W	NC	0x4002 4064	<a href="#">632</a>
ALHOUR	Alarm value for Hours	R/W	NC	0x4002 4068	<a href="#">632</a>
ALDOM	Alarm value for Day of Month	R/W	NC	0x4002 406C	<a href="#">632</a>
ALDOW	Alarm value for Day of Week	R/W	NC	0x4002 4070	<a href="#">632</a>
ALDOY	Alarm value for Day of Year	R/W	NC	0x4002 4074	<a href="#">632</a>
ALMON	Alarm value for Months	R/W	NC	0x4002 4078	<a href="#">632</a>
ALYEAR	Alarm value for Year	R/W	NC	0x4002 407C	<a href="#">632</a>

[1] Reset values apply only to a power-up of the RTC block, other types of reset have no effect on this block. Since the RTC is powered whenever either of the  $V_{DD(REG)(3V3)}$ , or  $V_{BAT}$  supplies are present, power-up reset occurs only when both supplies were absent and then one is turned on. Most registers are not affected by power-up of the RTC and must be initialized by software if the RTC is enabled. The Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 29.6.1 RTC interrupts

Interrupt generation is controlled through the Interrupt Location Register (ILR), Counter Increment Interrupt Register (CIIR), the alarm registers, and the Alarm Mask Register (AMR). Interrupts are generated only by the transition into the interrupt state. The ILR separately enables CIIR and AMR interrupts. Each bit in CIIR corresponds to one of the time counters. If CIIR is enabled for a particular counter, then every time the counter is incremented an interrupt is generated. The alarm registers allow the user to specify a date and time for an interrupt to be generated. The AMR provides a mechanism to mask alarm compares. If all non-masked alarm registers match the value in their corresponding time counter, then an interrupt is generated.

The RTC interrupt can bring the microcontroller out of Sleep, Deep Sleep, and Power-down modes when the RTC is running and the RTC interrupt is enabled in the NVIC. For details on the RTC based wake-up process see [Section 4.7.8 “Wake-up from Reduced Power Modes” on page 57](#) and [Section 4.8 “Wake-up timer” on page 60](#).

### 29.6.2 Miscellaneous register group

#### 29.6.2.1 Interrupt Location Register (ILR - 0x4002 4000)

The Interrupt Location Register is a 2-bit register that specifies which blocks are generating an interrupt (see [Table 619](#)). Writing a one to the appropriate bit clears the corresponding interrupt. Writing a zero has no effect. This allows the programmer to read this register and write back the same value to clear only the interrupt that is detected by the read.

**Table 619. Interrupt Location Register (ILR - address 0x4002 4000) bit description**

Bit	Symbol	Description	Reset value
0	RTCCIF	When one, the Counter Increment Interrupt block generated an interrupt. Writing a one to this bit location clears the counter increment interrupt.	0
1	RTCALF	When one, the alarm registers generated an interrupt. Writing a one to this bit location clears the alarm interrupt.	0
31:21	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 29.6.2.2 Clock Control Register (CCR - 0x4002 4008)

The clock register is a 4-bit register that controls the operation of the clock divide circuit. Each bit of the clock register is described in [Table 620](#). All NC bits in this register should be initialized when the RTC is first turned on.

**Table 620. Clock Control Register (CCR - address 0x4002 4008) bit description**

Bit	Symbol	Value	Description	Reset value
0	CLKEN		Clock Enable.	NC
		1	The time counters are enabled.	
		0	The time counters are disabled so that they may be initialized.	



Table 620. Clock Control Register (CCR - address 0x4002 4008) bit description

Bit	Symbol	Value	Description	Reset value
1	CTCRST		CTC Reset.	0
		1	When one, the elements in the internal oscillator divider are reset, and remain reset until CCR[1] is changed to zero. This is the divider that generates the 1 Hz clock from the 32.768 kHz crystal. The state of the divider is not visible to software.	
		0	No effect.	
3:2	-		Internal test mode controls. These bits must be 0 for normal RTC operation.	NC
4	CCALEN		Calibration counter enable.	NC
		1	The calibration counter is disabled and reset to zero.	
		0	The calibration counter is enabled and counting, using the 1 Hz clock. When the calibration counter is equal to the value of the CALIBRATION register, the counter resets and repeats counting up to the value of the CALIBRATION register. See <a href="#">Section 29.6.4.2</a> and <a href="#">Section 29.6.5</a> .	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.2.3 Counter Increment Interrupt Register (CIIR - 0x4002 400C)

The Counter Increment Interrupt Register (CIIR) gives the ability to generate an interrupt every time a counter is incremented. This interrupt remains valid until cleared by writing a 1 to bit 0 of the Interrupt Location Register (ILR[0]).

Table 621. Counter Increment Interrupt Register (CIIR - address 0x4002 400C) bit description

Bit	Symbol	Description	Reset value
0	IMSEC	When 1, an increment of the Second value generates an interrupt.	0
1	IMMIN	When 1, an increment of the Minute value generates an interrupt.	0
2	IMHOUR	When 1, an increment of the Hour value generates an interrupt.	0
3	IMDOM	When 1, an increment of the Day of Month value generates an interrupt.	0
4	IMDOW	When 1, an increment of the Day of Week value generates an interrupt.	0
5	IMDOY	When 1, an increment of the Day of Year value generates an interrupt.	0
6	IMMON	When 1, an increment of the Month value generates an interrupt.	0
7	IMYEAR	When 1, an increment of the Year value generates an interrupt.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.2.4 Alarm Mask Register (AMR - 0x4002 4010)

The Alarm Mask Register (AMR) allows the user to mask any of the alarm registers. [Table 622](#) shows the relationship between the bits in the AMR and the alarms. For the alarm function, every non-masked alarm register must match the corresponding time counter for an interrupt to be generated. The interrupt is generated only when the counter comparison first changes from no match to match. The interrupt is removed when a one is written to the appropriate bit of the Interrupt Location Register (ILR). If all mask bits are set, then the alarm is disabled.

**Table 622. Alarm Mask Register (AMR - address 0x4002 4010) bit description**

Bit	Symbol	Description	Reset value
0	AMRSEC	When 1, the Second value is not compared for the alarm.	0
1	AMRMIN	When 1, the Minutes value is not compared for the alarm.	0
2	AMRHOUR	When 1, the Hour value is not compared for the alarm.	0
3	AMRDOM	When 1, the Day of Month value is not compared for the alarm.	0
4	AMRDOW	When 1, the Day of Week value is not compared for the alarm.	0
5	AMRDOY	When 1, the Day of Year value is not compared for the alarm.	0
6	AMRMON	When 1, the Month value is not compared for the alarm.	0
7	AMRYEAR	When 1, the Year value is not compared for the alarm.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.2.5 RTC Auxiliary control register (RTC\_AUX - 0x4002 405C)

The RTC Auxiliary Control register contains added interrupt flags for functions that are not part of the Real Time Clock itself (the part recording the passage of time and generating other time related functions). On the LPC178x/177x, the only added interrupt flag is for failure of the RTC oscillator.

**Table 623. RTC Auxiliary control register (RTC\_AUX - address 0x4002 405C) bit description**

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Read value is undefined, only zero should be written.	NA
4	RTC_OSCF	RTC Oscillator Fail detect flag. Read: this bit is set if the RTC oscillator stops, and when RTC power is first turned on. An interrupt will occur when this bit is set, the RTC_OSCFEN bit in RTC_AUXEN is a 1, and the RTC interrupt is enabled in the NVIC. Write: writing a 1 to this bit clears the flag.	1
5	-	Reserved. Read value is undefined, only zero should be written.	NA
6	RTC_PDOUT	When 0: the RTC_ALARM pin reflects the RTC alarm status. When 1: the RTC_ALARM pin indicates Deep Power-down mode.	0
31:7	-	Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.2.6 RTC Auxiliary Enable register (RTC\_AUXEN - 0x4002 4058)

The RTC Auxiliary Enable Register controls whether additional interrupt sources represented in the RTC Auxiliary control register are enabled.

**Table 624. RTC Auxiliary Enable register (RTC\_AUXEN - address 0x4002 4058) bit description**

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Read value is undefined, only zero should be written.	NA
4	RTC_OSCFEN	Oscillator Fail Detect interrupt enable. When 0: the RTC Oscillator Fail detect interrupt is disabled. When 1: the RTC Oscillator Fail detect interrupt is enabled. See <a href="#">Section 29.6.2.5</a> .	0
31:5	-	Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.3 Consolidated time registers

The values of the Time Counters can optionally be read in a consolidated format which allows the programmer to read all time counters with only three read operations. The various registers are packed into 32-bit values as shown in [Table 625](#), [Table 626](#), and [Table 627](#). The least significant bit of each register is read back at bit 0, 8, 16, or 24.

The Consolidated Time Registers are read-only. To write new values to the Time Counters, the Time Counter addresses should be used.

#### 29.6.3.1 Consolidated Time Register 0 (CTIME0 - 0x4002 4014)

The Consolidated Time Register 0 contains the low order time values: Seconds, Minutes, Hours, and Day of Week.

**Table 625. Consolidated Time register 0 (CTIME0 - address 0x4002 4014) bit description**

Bit	Symbol	Description	Reset value
5:0	Seconds	Seconds value in the range of 0 to 59	NC
7:6	-	Reserved. The value read from a reserved bit is not defined.	NA
13:8	Minutes	Minutes value in the range of 0 to 59	NC
15:14	-	Reserved. The value read from a reserved bit is not defined.	NA
20:16	Hours	Hours value in the range of 0 to 23	NC
23:21	-	Reserved. The value read from a reserved bit is not defined.	NC
26:24	Day Of Week	Day of week value in the range of 0 to 6	NA
31:27	-	Reserved. The value read from a reserved bit is not defined.	NC

#### 29.6.3.2 Consolidated Time Register 1 (CTIME1 - 0x4002 4018)

The Consolidate Time Register 1 contains the Day of Month, Month, and Year values.

**Table 626. Consolidated Time register 1 (CTIME1 - address 0x4002 4018) bit description**

Bit	Symbol	Description	Reset value
4:0	Day of Month	Day of month value in the range of 1 to 28, 29, 30, or 31 (depending on the month and whether it is a leap year).	NC
7:5	-	Reserved. The value read from a reserved bit is not defined.	NA
11:8	Month	Month value in the range of 1 to 12.	NC
15:12	-	Reserved. The value read from a reserved bit is not defined.	NA
27:16	Year	Year value in the range of 0 to 4095.	NC
31:28	-	Reserved. The value read from a reserved bit is not defined.	NA

#### 29.6.3.3 Consolidated Time Register 2 (CTIME2 - 0x4002 401C)

The Consolidate Time Register 2 contains just the Day of Year value.

**Table 627. Consolidated Time register 2 (CTIME2 - address 0x4002 401C) bit description**

Bit	Symbol	Description	Reset value
11:0	Day of Year	Day of year value in the range of 1 to 365 (366 for leap years).	NC
31:12	-	Reserved. The value read from a reserved bit is not defined.	NA

### 29.6.4 Time Counter Group

The time value consists of the eight counters shown in [Table 628](#) and [Table 629](#). These counters can be read or written at the locations shown in [Table 629](#).

**Table 628. Time Counter relationships and values**

Counter	Size	Enabled by	Minimum value	Maximum value
Second	6	1 Hz Clock	0	59
Minute	6	Second	0	59
Hour	5	Minute	0	23
Day of Month	5	Hour	1	28, 29, 30 or 31
Day of Week	3	Hour	0	6
Day of Year	9	Hour	1	365 or 366 (for leap year)
Month	4	Day of Month	1	12
Year	12	Month or day of Year	0	4095

**Table 629. Time Counter registers**

Name	Size	Description	Access	Address
SEC	6	Seconds value in the range of 0 to 59	R/W	0x4002 4020
MIN	6	Minutes value in the range of 0 to 59	R/W	0x4002 4024
HOUR	5	Hours value in the range of 0 to 23	R/W	0x4002 4028
DOM	5	Day of month value in the range of 1 to 28, 29, 30, or 31 (depending on the month and whether it is a leap year). <a href="#">[1]</a>	R/W	0x4002 402C
DOW	3	Day of week value in the range of 0 to 6. <a href="#">[1]</a>	R/W	0x4002 4030
DOY	9	Day of year value in the range of 1 to 365 (366 for leap years). <a href="#">[1]</a>	R/W	0x4002 4034
MONTH	4	Month value in the range of 1 to 12	R/W	0x4002 4038
YEAR	12	Year value in the range of 0 to 4095	R/W	0x4002 403C

[1] These values are simply incremented at the appropriate intervals and reset at the defined overflow point. They are not calculated and must be correctly initialized in order to be meaningful.

#### 29.6.4.1 Leap year calculation

The RTC does a simple bit comparison to see if the two lowest order bits of the year counter are zero. If true, then the RTC considers that year a leap year. The RTC considers all years evenly divisible by 4 as leap years. This algorithm is accurate from the year 1901 through the year 2099, but fails for the year 2100, which is not a leap year. The only effect of leap year on the RTC is to alter the length of the month of February for the month, day of month, and year counters.

#### 29.6.4.2 Calibration register (CALIBRATION - address 0x4002 4040)

The following register is used to calibrate the time counter.

Table 630. Calibration register (CALIBRATION - address 0x4002 4040) bit description

Bit	Symbol	Value	Description	Reset value
16:0	CALVAL	-	If enabled, the calibration counter counts up to this value. The maximum value is 131, 072 corresponding to about 36.4 hours. Calibration is disabled if CALVAL = 0.	NC
17	CALDIR		Calibration direction	NC
		1	Backward calibration. When CALVAL is equal to the calibration counter, the RTC timers will stop incrementing for 1 second.	
		0	Forward calibration. When CALVAL is equal to the calibration counter, the RTC timers will jump by 2 seconds.	
31:12			Reserved. Read value is undefined, only zero should be written.	NA

### 29.6.5 Calibration procedure

The calibration logic can periodically adjust the time counter either by not incrementing the counter, or by incrementing the counter by 2 instead of 1. This allows calibrating the RTC oscillator under some typical voltage and temperature conditions without the need to externally trim the RTC oscillator.

A recommended method for determining the calibration value is to use the CLKOUT feature to unintrusively observe the RTC oscillator frequency under the conditions it is to be trimmed for, and calculating the number of clocks that will be seen before the time is off by one second. That value is used to determine CALVAL.

If the RTC oscillator is trimmed externally, the same method of unintrusively observing the RTC oscillator frequency may be helpful in that process.

#### Backward calibration

Enable the RTC timer and calibration in the CCR register (set bits CLKEN = 1 and CCALEN = 0). In the CALIBRATION register, set the calibration value CALVAL ≥ 1 and select CALDIR = 1.

- The SEC timer and the calibration counter count up for every 1 Hz clock cycle.
- When the calibration counter reaches CALVAL, a calibration match occurs and all RTC timers will be stopped for one clock cycle so that the timers will not increment in the next cycle.
- If an alarm match event occurs in the same cycle as the calibration match, the alarm interrupt will be delayed by one cycle to avoid a double alarm interrupt.

#### Forward calibration

Enable the RTC timer and calibration in the CCR register (set bits CLKEN = 1 and CCALEN = 0). In the CALIBRATION register, set the calibration value CALVAL ≥ 1 and select CALDIR = 0.

- The SEC timer and the calibration counter count up for every 1 Hz clock cycle.
- When the calibration counter reaches CALVAL, a calibration match occurs and the RTC timers are incremented by 2.
- When the calibration event occurs, the LSB of the ALSEC register is forced to be one so that the alarm interrupt will not be missed when skipping a second.

## 29.6.6 General purpose registers

### 29.6.6.1 General purpose registers 0 to 4 (GPREG0 to GPREG4 - addresses 0x4002 4044 to 0x4002 4054)

These registers can be used to store important information when the main power supply is off. The value in these registers is not affected by chip reset.

**Table 631. General purpose registers 0 to 4 (GPREG0 to GPREG4 - addresses 0x4002 4044 to 0x4002 4054) bit description**

Bit	Symbol	Description	Reset value
31:0	GP0 to GP4	General purpose storage.	N/A

## 29.6.7 Alarm register group

The alarm registers are shown in [Table 632](#). The values in these registers are compared with the time counters. If all the unmasked (See [Section 29.6.2.4 “Alarm Mask Register \(AMR - 0x4002 4010\)” on page 713](#)) alarm registers match their corresponding time counters then an interrupt is generated. The interrupt is cleared when a 1 is written to bit 1 of the Interrupt Location Register (ILR[1]).

**Table 632. Alarm registers**

Name	Size	Description	Access	Address
ALSEC	6	Alarm value for Seconds	R/W	0x4002 4060
ALMIN	6	Alarm value for Minutes	R/W	0x4002 4064
ALHOUR	5	Alarm value for Hours	R/W	0x4002 4068
ALDOM	5	Alarm value for Day of Month	R/W	0x4002 406C
ALDOW	3	Alarm value for Day of Week	R/W	0x4002 4070
ALDOY	9	Alarm value for Day of Year	R/W	0x4002 4074
ALMON	4	Alarm value for Months	R/W	0x4002 4078
ALYEAR	12	Alarm value for Years	R/W	0x4002 407C

## 29.7 RTC usage notes

If the RTC is used,  $V_{BAT}$  may be connected to an independent power supply (typically an external battery), or left floating. The RTC domain will always be internally powered if  $V_{DD(REG)(3V3)}$  is present, even if there is no power applied to  $V_{BAT}$ . As long as power is available on either  $V_{DD(REG)(3V3)}$  or  $V_{BAT}$ , the RTC will not lose its time value and backup register contents. If both  $V_{DD(REG)(3V3)}$  and  $V_{BAT}$  are not present, all RTC information will be lost. RTC incrementation will stop or be unpredictable if the clock source is lost, interrupted, or altered.

### 30.1 Features

---

- Internally resets chip if not reloaded during the programmable timeout period.
- Optional windowed operation requires reload to occur between a minimum and maximum timeout period, both programmable.
- Optional warning interrupt can be generated at a programmable time prior to watchdog timeout.
- Programmable 24 bit timer with internal fixed pre-scaler.
- Selectable time period from 1,024 watchdog clocks ( $T_{WDCLK} \times 256 \times 4$ ) to over 67 million watchdog clocks ( $T_{WDCLK} \times 2^{24} \times 4$ ) in increments of 4 watchdog clocks.
- “Safe” watchdog operation. Once enabled, requires a hardware reset or a Watchdog reset to be disabled.
- A dedicated on-chip watchdog oscillator provides a reliable clock source that cannot be turned off when the Watchdog Timer is running.
- Incorrect feed sequence causes immediate watchdog reset if the watchdog is enabled.
- The watchdog reload value can optionally be protected such that it can only be changed after the “warning interrupt” time is reached.
- Flag to indicate Watchdog reset.

### 30.2 Applications

---

The purpose of the Watchdog Timer is to reset the microcontroller within a reasonable amount of time if it enters an erroneous state. When enabled, a watchdog event will be generated if the user program fails to “feed” (or reload) the Watchdog within a predetermined amount of time. The Watchdog event will cause a chip reset if configured to do so.

When a watchdog window is programmed, an early watchdog feed is also treated as a watchdog event. This allows preventing situations where a system failure may still feed the watchdog. For example, application code could be stuck in an interrupt service that contains a watchdog feed. Setting the window such that this would result in an early feed will generate a watchdog event, allowing for system recovery.

For interaction of the on-chip watchdog and other peripherals, especially the reset and boot-up procedures, please refer to [Section 3.4](#).

### 30.3 Description

The Watchdog consists of a fixed divide by 4 pre-scaler and a 24 bit counter which decrements when clocked. The minimum value from which the counter decrements is 0xFF. Setting a value lower than 0xFF causes 0xFF to be loaded in the counter. Hence the minimum Watchdog interval is  $(T_{WDCLK} \times 256 \times 4)$  and the maximum Watchdog interval is  $(T_{WDCLK} \times 2^{24} \times 4)$  in multiples of  $(T_{WDCLK} \times 4)$ . The Watchdog should be used in the following manner:

- Set the Watchdog timer constant reload value in WDTC register.
- Setup the Watchdog timer operating mode in WDMOD register.
- Set a value for the watchdog window time in WDWINDOW register if windowed operation is required.
- Set a value for the watchdog warning interrupt in the WDWARNINT register if a warning interrupt is required.
- Enable the Watchdog by writing 0xAA followed by 0x55 to the WDFEED register.
- The Watchdog must be fed again before the Watchdog counter reaches zero in order to prevent a watchdog event. If a window value is programmed, the feed must also occur after the watchdog counter passes that value.

When the Watchdog Timer is configured so that a watchdog event will cause a reset and the counter reaches zero, the CPU will be reset, loading the stack pointer and program counter from the vector table as in the case of external reset. The Watchdog time-out flag (WDTOF) can be examined to determine if the Watchdog has caused the reset condition. The WDTOF flag must be cleared by software.

When the Watchdog Timer is configured to generate a warning interrupt, the interrupt will occur when the counter matches the value defined by the WDWARNINT register.

The watchdog timer block uses two clocks: PCLK and WDCLK. PCLK is used for the APB accesses to the watchdog registers. WDCLK runs the watchdog timer counter. This clock comes from a dedicated oscillator that is always on when the Watchdog Timer is enabled. This oscillator has a typical frequency of 500 kHz (see [Section 4.3.4](#)).

There is some synchronization logic between these two clock domains. When the WDMOD and WDTC registers are updated by APB operations, the new value will take effect on the logic in the WDCLK clock domain in 3 WDCLK cycles. When the watchdog timer is counting, the synchronization logic will first lock the value of the counter on WDCLK and then synchronize it with the PCLK when the WDTV register is read by the CPU.

The block diagram of the Watchdog is shown below in the [Figure 153](#). The synchronization logic (PCLK - WDCLK) is not shown in the block diagram.



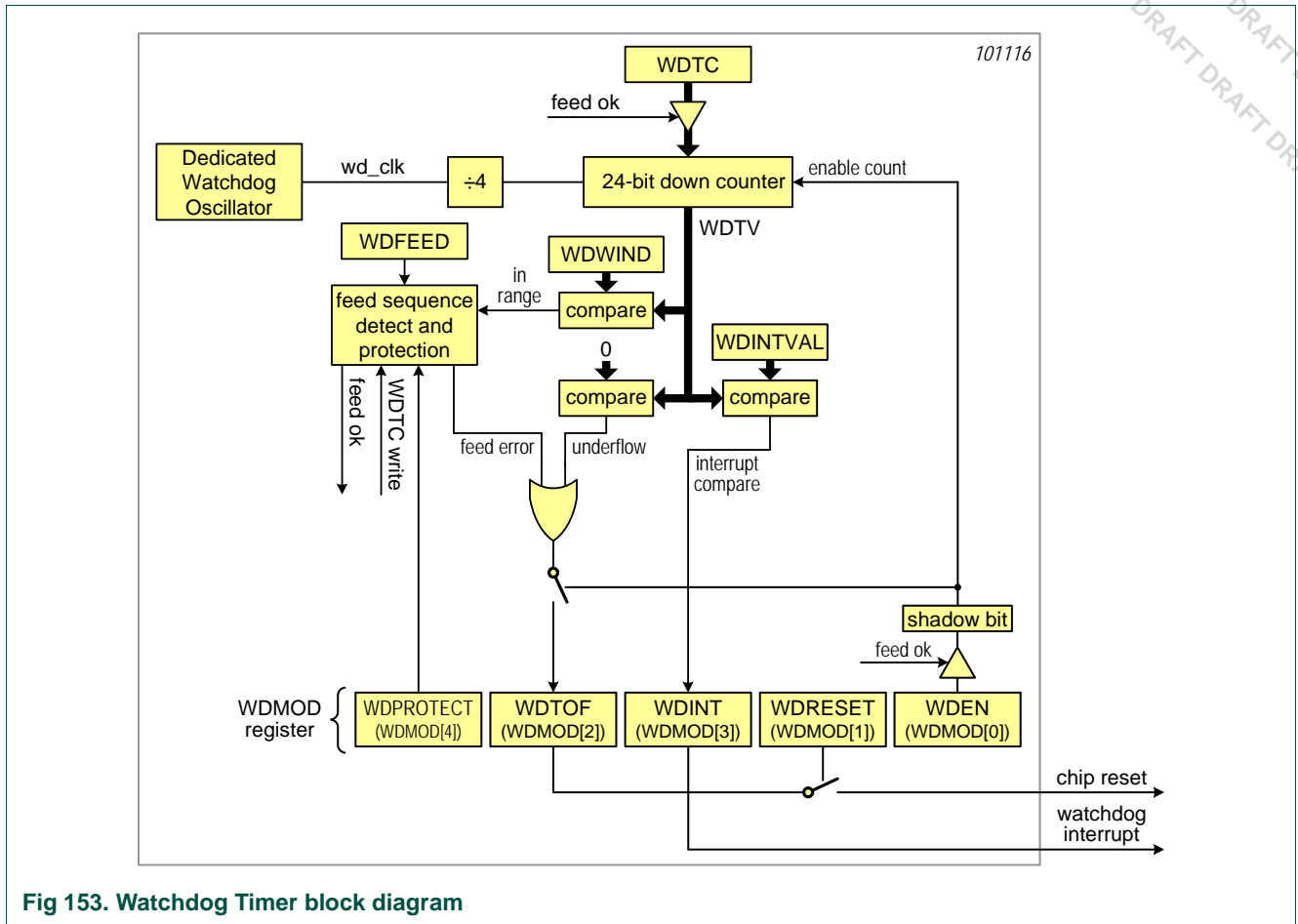


Fig 153. Watchdog Timer block diagram

### 30.4 Register description

The Watchdog contains 4 registers as shown in [Table 633](#) below.

**Table 633. Watchdog register map**

Name	Description	Access	Reset Value <sup>[1]</sup>	Address	Table
WDMOD	Watchdog mode register. This register determines the basic mode and status of the Watchdog Timer.	R/W	0	0x4000 0000	<a href="#">634</a>
WDTC	Watchdog timer constant register. The value in this register determines the time-out value.	R/W	0xFF	0x4000 0004	<a href="#">636</a>
WDFEED	Watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer with the value contained in WDTC.	WO	NA	0x4000 0008	<a href="#">637</a>
WDTV	Watchdog timer value register. This register reads out the current value of the Watchdog timer.	RO	0xFF	0x4000 000C	<a href="#">638</a>
WDWARNINT	Watchdog Warning Interrupt compare value.	R/W	0	0x4000 0014	<a href="#">639</a>
WDWINDOW	Watchdog Window compare value.	R/W	0xFF FFFF	0x4000 0018	<a href="#">640</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

#### 30.4.1 Watchdog Mode register (WDMOD - 0x4000 0000)

The WDMOD register controls the operation of the Watchdog as per the combination of WDEN and RESET bits. Note that a watchdog feed must be performed before any changes to the WDMOD register take effect.

**Table 634: Watchdog Mode register (WDMOD - 0x4000 0000) bit description**

Bit	Symbol	Value	Description	Reset Value
0	WDEN		Watchdog enable bit. This bit is Set Only.	0
		0	The watchdog timer is stopped.	
		1	The watchdog timer is running.	
1	WDRESET		Watchdog reset enable bit. This bit is Set Only.	0
		0	A watchdog timeout will not cause a chip reset.	
		1	A watchdog timeout will cause a chip reset.	
2	WDTOF		Watchdog time-out flag. Set when the watchdog timer times out, by a feed error, or by events associated with WDPROTECT, cleared by software. Causes a chip reset if WDRESET = 1.	0 (Only after external reset)
3	WDINT		Watchdog interrupt flag. Set when the timer reaches the value in WDWARNINT. Cleared by software.	0
4	WDPROTECT		Watchdog update mode. This bit is Set Only.	0
		0	The watchdog reload value (WDTC) can be changed at any time.	
		1	The watchdog reload value (WDTC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW. <b>Note:</b> this mode is intended for use only when WDRESET = 1.	
7:5	-		Reserved. Read value is undefined, only zero should be written.	NA

Once the **WDEN**, **WDPROTECT**, or **WDRESET** bits are set they can not be cleared by software. Both flags are cleared by an external reset or a Watchdog timer reset.

**WDTOF** The Watchdog time-out flag is set when the Watchdog times out, when a feed error occurs, or when **WDPROTECT** =1 and an attempt is made to write to the **WDTC** register. This flag is cleared by software writing a 0 to this bit.

**WDINT** The Watchdog interrupt flag is set when the Watchdog counter reaches the value specified by **WDWARNINT**. This flag is cleared when any reset occurs, and is cleared by software by writing a 1 to this bit.

Watchdog reset or interrupt will occur any time the watchdog is running. If a watchdog interrupt occurs in Sleep or Deep Sleep mode, it will wake up the device.

**Table 635. Watchdog operating modes selection**

WDEN	WDRESET	Mode of Operation
0	X (0 or 1)	Debug/Operate without the Watchdog running.
1	0	Watchdog interrupt mode: the watchdog warning interrupt will be generated but watchdog reset will not. When this mode is selected, the watchdog counter reaching the value specified by <b>WDWARNINT</b> will set the <b>WDINT</b> flag and the Watchdog interrupt request will be generated.
1	1	Watchdog reset mode: both the watchdog interrupt and watchdog reset are enabled. When this mode is selected, the watchdog counter reaching the value specified by <b>WDWARNINT</b> will set the <b>WDINT</b> flag and the Watchdog interrupt request will be generated, and the watchdog counter reaching zero will reset the microcontroller. A watchdog feed prior to reaching the value of <b>WDWINDOW</b> will also cause a watchdog reset.

### 30.4.2 Watchdog Timer Constant register (WDTC - base + -x04)

The **WDTC** register determines the time-out value. Every time a feed sequence occurs the **WDTC** content is reloaded in to the Watchdog timer. This is pre-loaded with the value 0x00 00FF upon reset. Writing values below 0xFF will cause 0x00 00FF to be loaded into the **WDTC**. Thus the minimum time-out interval is  $T_{WDCLK} \times 256 \times 4$ .

If the **WDPROTECT** bit in **WDMOD** = 1, an attempt to change the value of **WDTC** before the watchdog counter is below the values of **WDWARNINT** and **WDWINDOW** will cause a watchdog reset and set the **WDTOF** flag.

**Table 636: Watchdog Timer Constant register (WDTC - 0x4000 0004) bit description**

Bit	Symbol	Description	Reset Value
23:0	Count	Watchdog time-out interval.	0x00 00FF
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 30.4.3 Watchdog Feed register (WDFEED - 0x4000 0008)

Writing 0xAA followed by 0x55 to this register will reload the Watchdog timer with the **WDTC** value. This operation will also start the Watchdog if it is enabled via the **WDMOD** register. Setting the **WDEN** bit in the **WDMOD** register is not sufficient to enable the Watchdog. A valid feed sequence must be completed after setting **WDEN** before the Watchdog is capable of generating a reset. Until then, the Watchdog will ignore feed errors. After writing 0xAA to **WDFEED**, access to any Watchdog register other than writing 0x55 to **WDFEED** causes an immediate reset/interrupt when the Watchdog is enabled, and sets the **WDTOF** flag. The reset will be generated during the second **PCLK** following an incorrect access to a Watchdog register during a feed sequence.

**Table 637: Watchdog Feed register (WDFEED - 0x4000 0008) bit description**

Bit	Symbol	Description	Reset Value
7:0	Feed	Feed value should be 0xAA followed by 0x55.	NA

### 30.4.4 Watchdog Timer Value register (WDTV - 0x4000 000C)

The WDTV register is used to read the current value of Watchdog timer counter.

When reading the value of the 24 bit counter, the lock and synchronization procedure takes up to 6 WDCLK cycles plus 6 PCLK cycles, so the value of WDTV is older than the actual value of the timer when it's being read by the CPU.

**Table 638: Watchdog Timer Value register (WDTV - 0x4000 000C) bit description**

Bit	Symbol	Description	Reset Value
23:0	Count	Counter timer value.	0x00 00FF
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 30.4.5 Watchdog Timer Warning Interrupt register (WDWARNINT - 0x4000 0014)

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter matches the value defined by WDWARNINT, an interrupt will be generated after the subsequent WDCLK.

A match of the watchdog timer counter to WDWARNINT occurs when the bottom 10 bits of the counter have the same value as the 10 bits of WDWARNINT, and the remaining upper bits of the counter are all 0. This gives a maximum time of 1,023 watchdog timer counts (4,096 watchdog clocks) for the interrupt to occur prior to a watchdog event. If WDWARNINT is set to 0, the interrupt will occur at the same time as the watchdog event.

**Table 639: Watchdog Timer Warning Interrupt register (WDWINDOW - 0x4000 0014) bit description**

Bit	Symbol	Description	Reset Value
9:0	WARNINT	Watchdog warning interrupt compare value.	0
31:10	-	Reserved. Read value is undefined, only zero should be written.	NA

### 30.4.6 Watchdog Timer Window register (WDWINDOW - 0x4000 0018)

The WDWINDOW register determines the highest WDTV value allowed when a watchdog feed is performed. If a feed valid sequence completes prior to WDTV reaching the value in WDWINDOW, a watchdog event will occur.

WDWINDOW resets to the maximum possible WDTV value, so windowing is not in effect.

**Table 640: Watchdog Timer Window register (WDWINDOW - 0x4000 0018) bit description**

Bit	Symbol	Description	Reset Value
23:0	WINDOW	Watchdog window value.	0xFF FFFF
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 30.5 Watchdog timing examples

The following figures illustrate several aspects of Watchdog Timer operation is shown below in the [Figure 154](#).

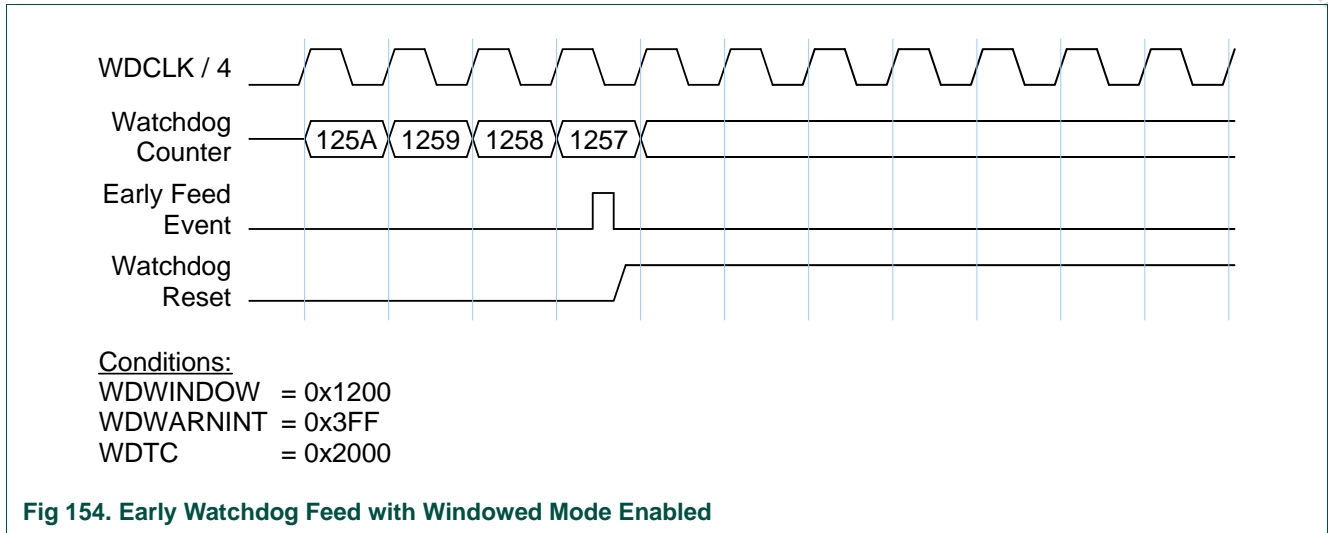


Fig 154. Early Watchdog Feed with Windowed Mode Enabled

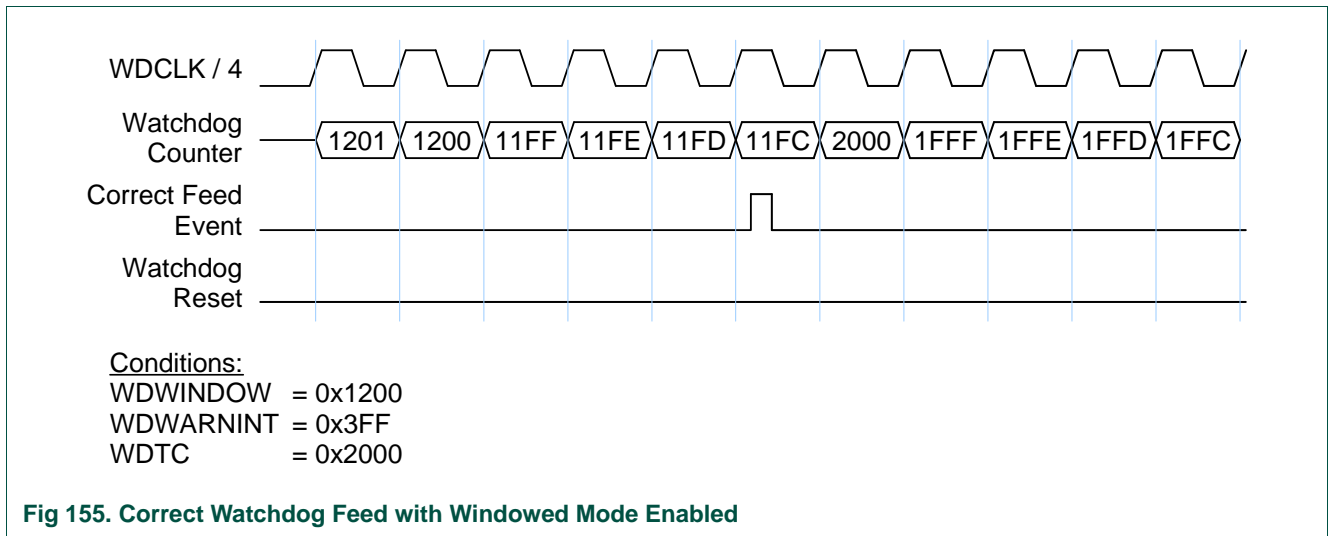
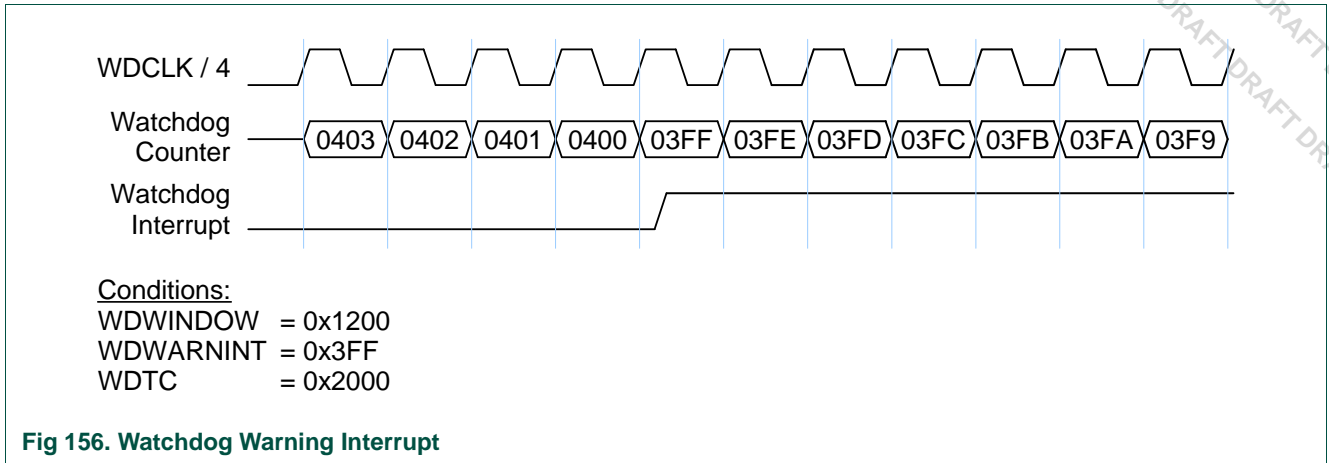


Fig 155. Correct Watchdog Feed with Windowed Mode Enabled



### 31.1 Basic configuration

---

The ADC is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set the PCADC bit.  
**Remark:** On reset, the ADC is disabled. To enable the ADC, first set the PCADC bit, and then enable the ADC in the AD0CR register (bit PDN [Table 643](#)). To disable the ADC, first clear the PDN bit, and then clear the PCADC bit.
2. Peripheral clock: The ADC operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#). To scale the clock for the ADC, see bits CLKDIV in [Table 643](#).
3. Pins: Enable ADC0 pins and pin modes for the port pins with ADC0 functions through the relevant IOCON registers ([Section 8.4.1](#)).
4. Interrupts: To enable interrupts in the ADC, see [Table 647](#). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register. Disable the ADC interrupt in the NVIC using the appropriate Interrupt Set Enable register.
5. DMA: See [Section 31.6.4](#). For GPDMA system connections, see [Table 655](#).

### 31.2 Features

---

- 12-bit successive approximation analog to digital converter.
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range  $V_{SS}$  to  $V_{REFP}$  (typically 3 V; not to exceed  $V_{DDA}$  voltage level).
- 12-bit conversion rate of 400 kHz.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.

### 31.3 Description

Basic clocking for the A/D converters is provided by the APB clock. A programmable divider is included in each converter to scale this clock to the clock (maximum 12.4 MHz) needed by the successive approximation process. A fully accurate conversion requires 31 of these clocks.

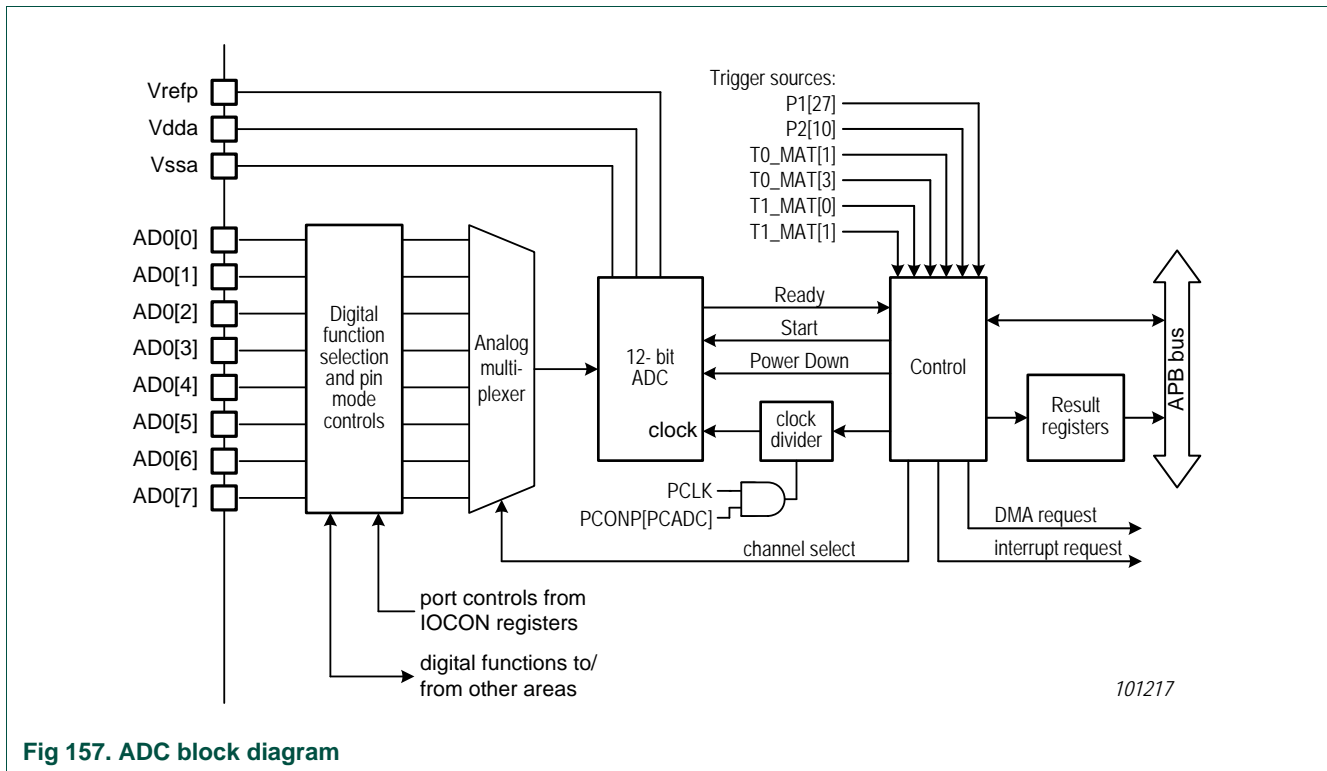


Fig 157. ADC block diagram

### 31.4 Pin description

Table 641 gives a brief summary of each of ADC related pins.

Table 641. ADC pin description

Pin	Type	Description
AD0[7] to AD0[0]	Input	<b>Analog Inputs.</b> The ADC cell can measure the voltage on any of these input signals. Digital signals are disconnected from the ADC input pins when the ADC function is selected on that pin in the Pin Select register. <b>Warning:</b> if the ADC is used, signal levels on analog input pins must not be above the level of $V_{DDA}$ at any time. Otherwise, A/D converter readings will be invalid. If the A/D converter is not used in an application then the pins associated with A/D inputs can be used as 5 V tolerant digital IO pins.
$V_{REFP}$	Reference	<b>Voltage Reference.</b> This pin provides a voltage reference level for the ADC and DAC. <b>Note:</b> $V_{REFP}$ should be tied to VDD(3V3) if the ADC and DAC are not used.
$V_{DDA}$ , $V_{SSA}$	Power	<b>Analog Power and Ground.</b> These should typically be the same voltages as $V_{DD}$ and $V_{SS}$ , but should be isolated to minimize noise and error. <b>Note:</b> $V_{DDA}$ should be tied to VDD(3V3) and $V_{SSA}$ should be tied to VSS if the ADC and DAC are not used.



## 31.5 Register description

The A/D Converter registers are shown in [Table 642](#).

**Table 642. ADC registers**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	AD0 Name & Address	Table
ADCR	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.	R/W	1	AD0CR - 0x4003 4000	<a href="#">643</a>
ADGDR	A/D Global Data Register. This register contains the ADC's DONE bit and the result of the most recent A/D conversion.	R/W	NA	AD0GDR - 0x4003 4004	<a href="#">644</a>
ADINTEN	A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt.	R/W	0x100	AD0INTEN - 0x4003 400C	<a href="#">645</a>
ADDR0	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0.	RO	NA	AD0DR0 - 0x4003 4010	<a href="#">646</a>
ADDR1	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	RO	NA	AD0DR1 - 0x4003 4014	<a href="#">646</a>
ADDR2	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	RO	NA	AD0DR2 - 0x4003 4018	<a href="#">646</a>
ADDR3	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	RO	NA	AD0DR3 - 0x4003 401C	<a href="#">646</a>
ADDR4	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	RO	NA	AD0DR4 - 0x4003 4020	<a href="#">646</a>
ADDR5	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	RO	NA	AD0DR5 - 0x4003 4024	<a href="#">646</a>
ADDR6	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	RO	NA	AD0DR6 - 0x4003 4028	<a href="#">646</a>
ADDR7	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	RO	NA	AD0DR7 - 0x4003 402C	<a href="#">646</a>
ADSTAT	A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt/DMA flag.	RO	0	AD0STAT - 0x4003 4030	<a href="#">647</a>
ADTRM	ADC trim register.	R/W	0	AD0TRM - 0x4003 4034	<a href="#">648</a>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

### 31.5.1 A/D Control Register (AD0CR - 0x4003 4000)

Table 643: A/D Control Register (AD0CR - address 0x4003 4000) bit description

Bit	Symbol	Value	Description	Reset value
7:0	SEL		Selects which of the AD0[7:0] pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0[0], and bit 7 selects pin AD0[7]. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.	0x01
15:8	CLKDIV		The APB clock (PCLK) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 12.4 MHz. Typically, software should program the smallest value in this field that yields a clock of 12.4 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	0
16	BURST	1	The AD converter does repeated conversions at up to 400 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed. <b>Remark:</b> START bits must be 000 when BURST = 1 or conversions will not start.	0
		0	Conversions are software controlled and require 31 clocks.	
20:17	-		Reserved. Read value is undefined, only zero should be written.	NA
21	PDN	1	The A/D converter is operational.	0
		0	The A/D converter is in power-down mode.	
23:22	-		Reserved. Read value is undefined, only zero should be written.	NA
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
		000	No start (this value should be used when clearing PDN to 0).	
		001	Start conversion now.	
		010	Start conversion when the edge selected by bit 27 occurs on the P2[10] pin.	
		011	Start conversion when the edge selected by bit 27 occurs on the P1[27] pin.	
		100	Start conversion when the edge selected by bit 27 occurs on MAT0.1. Note that this does not require that the MAT0.1 function appear on a device pin.	
		101	Start conversion when the edge selected by bit 27 occurs on MAT0.3. Note that it is not possible to cause the MAT0.3 function to appear on a device pin.	
		110	Start conversion when the edge selected by bit 27 occurs on MAT1.0. Note that this does not require that the MAT1.0 function appear on a device pin.	
		111	Start conversion when the edge selected by bit 27 occurs on MAT1.1. Note that this does not require that the MAT1.1 function appear on a device pin.	
27	EDGE		This bit is significant only when the START field contains 010-111. In these cases:	0
		1	Start conversion on a falling edge on the selected CAP/MAT signal.	
		0	Start conversion on a rising edge on the selected CAP/MAT signal.	
31:28	-		Reserved. Read value is undefined, only zero should be written.	NA

### 31.5.2 A/D Global Data Register (AD0GDR - 0x4003 4004)

The A/D Global Data Register holds the result of the most recent A/D conversion that has completed, and also includes copies of the status flags that go with that conversion.

Results of ADC conversion can be read in one of two ways. One is to use the A/D Global Data Register to read all data from the ADC. Another is to use the A/D Channel Data Registers. It is important to use one method consistently because the DONE and OVERRUN flags can otherwise get out of synch between the AD0GDR and the A/D Channel Data Registers, potentially causing erroneous interrupts or DMA activity.

**Table 644: A/D Global Data Register (AD0GDR - address 0x4003 4004) bit description**

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Read value is undefined, only zero should be written.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin selected by the SEL field, as it falls within the range of $V_{REFP}$ to $V_{SS}$ . Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on $V_{SS}$ , while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on $V_{REFP}$ .	NA
23:16	-	Reserved. Read value is undefined, only zero should be written.	NA
26:24	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 000 identifies channel 0, 001 channel 1...).	NA
29:27	-	Reserved. Read value is undefined, only zero should be written.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.	0
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.	0

### 31.5.3 A/D Interrupt Enable register (AD0INTEN - 0x4003 400C)

This register allows control over which A/D channels generate an interrupt when a conversion is complete. For example, it may be desirable to use some A/D channels to monitor sensors by continuously performing conversions on them. The most recent results are read by the application program whenever they are needed. In this case, an interrupt is not desirable at the end of each conversion for some A/D channels.

**Table 645: A/D Interrupt Enable register (AD0INTEN - address 0x4003 400C) bit description**

Bit	Symbol	Value	Description	Reset value
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.	
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.	
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.	
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.	
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.	
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.	

Table 645: A/D Interrupt Enable register (AD0INTEN - address 0x4003 400C) bit description

Bit	Symbol	Value	Description	Reset value
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.	
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.	
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts.	1
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

### 31.5.4 A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C)

The A/D Data Registers hold the result of the last conversion for each A/D channel, when an A/D conversion is complete. They also include the flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

Results of ADC conversion can be read in one of two ways. One is to use the A/D Global Data Register to read all data from the ADC. Another is to use the A/D Channel Data Registers. It is important to use one method consistently because the DONE and OVERRUN flags can otherwise get out of synch between the AD0GDR and the A/D Channel Data Registers, potentially causing erroneous interrupts or DMA activity.

Table 646: A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Read value is undefined, only zero should be written.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of $V_{REFP}$ to $V_{SS}$ . Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on $V_{SS}$ , while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on $V_{REFP}$ .	NA
29:16	-	Reserved. Read value is undefined, only zero should be written.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.	
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.	NA

### 31.5.5 A/D Status register (ADSTAT - 0x4003 4030)

The A/D Status register allows checking the status of all A/D channels simultaneously. The DONE and OVERRUN flags appearing in the ADDRn register for each A/D channel are mirrored in ADSTAT. The interrupt flag (the logical OR of all DONE flags) is also found in ADSTAT.

Table 647: A/D Status register (AD0STAT - address 0x4003 4030) bit description

Bit	Symbol	Description	Reset value
0	DONE0	This bit mirrors the DONE status flag from the result register for A/D channel 0.	0
1	DONE1	This bit mirrors the DONE status flag from the result register for A/D channel 1.	0
2	DONE2	This bit mirrors the DONE status flag from the result register for A/D channel 2.	0
3	DONE3	This bit mirrors the DONE status flag from the result register for A/D channel 3.	0
4	DONE4	This bit mirrors the DONE status flag from the result register for A/D channel 4.	0
5	DONE5	This bit mirrors the DONE status flag from the result register for A/D channel 5.	0
6	DONE6	This bit mirrors the DONE status flag from the result register for A/D channel 6.	0
7	DONE7	This bit mirrors the DONE status flag from the result register for A/D channel 7.	0
8	OVERRUN0	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 0.	0
9	OVERRUN1	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 1.	0
10	OVERRUN2	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 2.	0
11	OVERRUN3	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 3.	0
12	OVERRUN4	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 4.	0
13	OVERRUN5	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 5.	0
14	OVERRUN6	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 6.	0
15	OVERRUN7	This bit mirrors the OVERRRUN status flag from the result register for A/D channel 7.	0
16	ADINT	This bit is the A/D interrupt flag. It is one when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

### 31.5.6 A/D Trim register (ADTRIM - 0x4003 4034)

This register will be set by the bootcode on start-up. It contains the trim values for the DAC and the ADC. The offset trim values for the ADC can be overwritten by the user. All 12 bits are visible when this register is read.

Table 648: A/D Trim register (ADTRM - address 0x4003 4034) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Read value is undefined, only zero should be written.	NA
7:4	ADCOFFS	Offset trim bits for ADC operation. Initialized by the boot code. Can be overwritten by the user.	0
11:8	TRIM	written-to by boot code. Can <b>not</b> be overwritten by the user. These bits are locked after boot code write.	0
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

## 31.6 Operation

Once an ADC conversion is started, it cannot be interrupted. A new software write to launch a new conversion or a new edge-trigger event will be ignored while the previous conversion is in progress.

### 31.6.1 Hardware-triggered conversion

If the BURST bit in the ADCR is 0 and the START field contains 010-111, the ADC will start a conversion when a transition occurs on a selected pin or Timer Match signal. The choices include conversion on a specified edge of any of 4 Match signals, or conversion on a specified edge of either of 2 Capture/Match pins. The pin state from the selected pad or the selected Match signal, XORed with ADCR bit 27, is used in the edge detection logic.

### 31.6.2 Interrupts

An interrupt request is asserted to the NVIC when the DONE bit is 1. Software can use the Interrupt Enable bit for the A/D Converter in the NVIC to control whether this assertion results in an interrupt. DONE is negated when the ADDR is read.

### 31.6.3 Accuracy vs. digital receiver

The ADC function must be selected via the ADMODE bit in the related IOCON registers in order to obtain voltage readings on the monitored pin. The same IOCON registers should also be set to the mode for which neither pull-up nor pull-down resistor is enabled. For a pin hosting an ADC input, it is not possible to have a digital function selected and yet get valid ADC readings, the analog input is disabled when a digital function is selected on the pin.

### 31.6.4 DMA control

A DMA transfer request is generated from the ADC interrupt request line. To generate a DMA transfer the same conditions must be met as the conditions for generating an interrupt (see [Section 31.6.2](#) and [Section 31.5.3](#)).

**Remark:** If the DMA is used, the ADC interrupt must be disabled in the NVIC.

For DMA transfers, only burst requests are supported. The burst size can be set to one in the DMA channel control register (see [Section 33.5.20](#)). If the number of ADC channels is not equal to one of the other DMA-supported burst sizes (applicable DMA burst sizes are 1, 4, 8 - see [Section 33.5.20](#)), set the burst size to one.

The DMA transfer size determines when a DMA interrupt is generated. The transfer size can be set to the number of ADC channels being converted (see [Section 33.5.20](#)). Non-contiguous channels can be transferred by the DMA using the scatter/gather linked lists (see [Section 33.5.19](#)).

### 32.1 Basic configuration

---

The DAC is configured using the following registers:

1. Power: The DAC is always connected to  $V_{DDA}$ . Register access is determined by IOCON register settings (see below).
2. Peripheral clock: The DAC operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See [Section 4.6.4](#).
3. Pins: Enable the DAC pin and select the pin mode for DACOUT through the relevant IOCON register ([Section 8.4.1](#)). This must be done before accessing any DAC registers.
4. DMA: The DAC can be connected to the GPDMA controller (see [Section 32.5.2](#)). For GPDMA connections, see [Table 655](#).

### 32.2 Features

---

- 10-bit digital to analog converter
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable speed vs. power
- Maximum update rate of 1 MHz.

### 32.3 Architecture

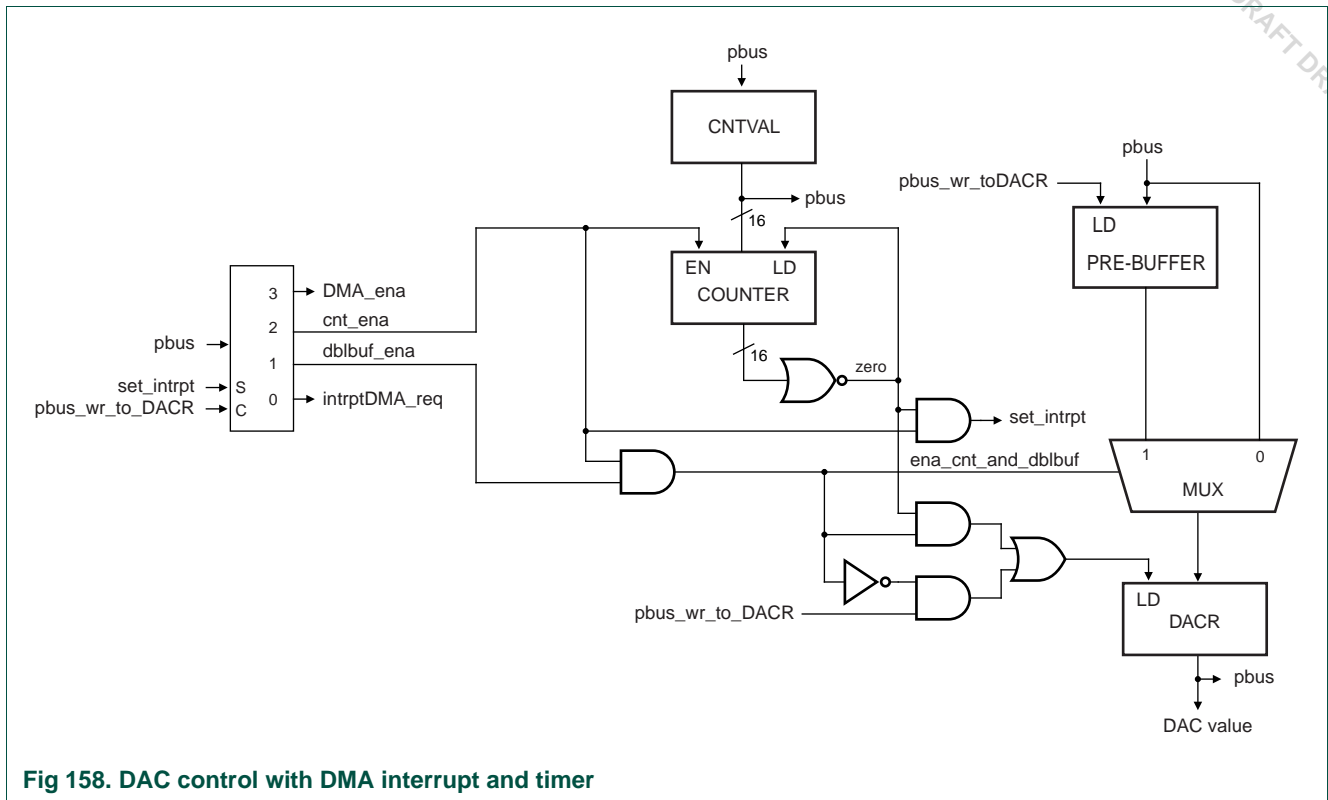


Fig 158. DAC control with DMA interrupt and timer

### 32.4 Pin description

Table 649 gives a brief summary of each of DAC related pins.

Table 649. D/A Pin Description

Pin	Type	Description
DAC_OUT	Output	<b>Analog Output.</b> After the selected settling time after the DACR is written with a new value, the voltage on this pin (with respect to V <sub>SSA</sub> ) is VALUE × ((V <sub>REFP</sub> - V <sub>REFN</sub> )/1024) + V <sub>REFN</sub> .
V <sub>REFP</sub>	Reference	<b>Voltage Reference.</b> This pin provides a voltage reference level for the ADC and DAC. <b>Note: V<sub>REFP</sub> should be tied to VDD(3V3) if the ADC and DAC are not used.</b>
V <sub>DDA</sub> , V <sub>SSA</sub>	Power	<b>Analog Power and Ground.</b> These should typically be the same voltages as V <sub>DD</sub> and V <sub>SS</sub> , but should be isolated to minimize noise and error. <b>Note: VDDA should be tied to VDD(3V3) and VSSA should be tied to VSS if the ADC and DAC are not used.</b>



### 32.5 Register description

The DAC registers are shown in [Table 650](#). Note that the DAC does not have a control bit in the PCONP register. To enable the DAC, its output must be selected to appear on the related pin, P0[26], by configuring the relevant IOCON register ([Section 8.4.1](#)). See [Section 8.4.1 “I/O configuration register contents \(IOCON\)”](#). the DAC must be enabled in this manner prior to accessing any DAC registers.

**Table 650. DAC registers**

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
DACR	D/A Converter Register. This register contains the digital value to be converted to analog and a power control bit.	R/W	0	0x4008 C000	<a href="#">651</a>
DACCTRL	DAC Control register. This register controls DMA and timer operation.	R/W	0	0x4008 C004	<a href="#">652</a>
DACCNTVAL	DAC Counter Value register. This register contains the reload value for the DAC DMA/Interrupt timer.	R/W	0	0x4008 C008	<a href="#">653</a>

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

#### 32.5.1 D/A Converter Register (DACR - 0x4008 C000)

This read/write register includes the digital value to be converted to analog, and a bit that trades off performance vs. power. Bits 5:0 are reserved for future, higher-resolution D/A converters.

**Table 651: D/A Converter Register (DACR - address 0x4008 C000) bit description**

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved. Read value is undefined, only zero should be written.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the DAC_OUT pin (with respect to V <sub>SSA</sub> ) is VALUE × ((V <sub>REFP</sub> - V <sub>REFN</sub> )/1024) + V <sub>REFN</sub> .	0
16	BIAS <sup>[1]</sup>	0	The settling time of the DAC is 1 μs max, and the maximum current is 700 μA. This allows a maximum update rate of 1 MHz.	0
		1	The settling time of the DAC is 2.5 μs and the maximum current is 350 μA. This allows a maximum update rate of 400 kHz.	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

[1] The settling times noted in the description of the BIAS bit are valid for a capacitance load on the DAC\_OUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time. One or more graph(s) of load impedance vs. settling time will be included in the final data sheet.

#### 32.5.2 D/A Converter Control register (DACCTRL - 0x4008 C004)

This read/write register enables the DMA operation and controls the DMA timer.

**Table 652. D/A Control register (DACCTRL - address 0x4008 C004) bit description**

Bit	Symbol	Value	Description	Reset Value
0	INT_DMA_REQ	0	This bit is cleared on any write to the DACR register.	0
		1	This bit is set by hardware when the timer times out.	
1	DBLBUF_ENA	0	DACR double-buffering is disabled.	0
		1	When this bit and the CNT_ENA bit are both set, the double-buffering feature in the DACR register will be enabled. Writes to the DACR register are written to a pre-buffer and then transferred to the DACR on the next time-out of the counter.	
2	CNT_ENA	0	Time-out counter operation is disabled.	0
		1	Time-out counter operation is enabled.	
3	DMA_ENA	0	DMA access is disabled.	0
		1	DMA Burst Request Input 7 is enabled for the DAC (see <a href="#">Table 655</a> ).	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

### 32.5.3 D/A Converter Counter Value register (DACCNTVAL - 0x4008 C008)

This read/write register contains the reload value for the Interrupt/DMA counter.

**Table 653: D/A Converter Counter Value register (DACCNTVAL - address 0x4008 C008) bit description**

Bit	Symbol	Description	Reset Value
15:0	VALUE	16-bit reload value for the DAC interrupt/DMA timer.	0

## 32.6 Operation

### 32.6.1 DMA counter

When the counter enable bit CNT\_ENA in DACCTRL is set, a 16-bit counter will begin counting down, at the rate selected by PCLK (see [Table 33](#)), from the value programmed into the DACNTVAL register. The counter is decremented Each time the counter reaches zero, the counter will be reloaded by the value of DACNTVAL and the DMA request bit INT\_DMA\_REQ will be set in hardware.

Note that the contents of the DACCTRL and DACNTVAL registers are read and write accessible, but the timer itself is not accessible for either read or write.

If the DMA\_ENA bit is set in the DACCTRL register, the DAC DMA request will be routed to the GPDMA. When the DMA\_ENA bit is cleared, the default state after a reset, DAC DMA requests are blocked.

### 32.6.2 Double buffering

Double-buffering is enabled only if both, the CNT\_ENA and the DBLBUF\_ENA bits are set in DACCTRL. In this case, any write to the DACR register will only load the pre-buffer, which shares its register address with the DACR register. The DACR itself will be loaded from the pre-buffer whenever the counter reaches zero and the DMA request is set. At the same time the counter is reloaded with the COUNTVAL register value.

Reading the DACR register will only return the contents of the DACR register itself, not the contents of the pre-buffer register.

If either the CNT\_ENA or the DBLBUF\_ENA bits are 0, any writes to the DACR address will go directly to the DACR register.

### 33.1 Basic configuration

---

The GPDMA is configured using the following registers:

1. Power: In the PCONP register ([Table 37](#)), set bit PCGPDMA.  
**Remark:** On reset, the GPDMA is disabled (PCGPDMA = 0).
2. Clock: The GPDMA operates at the AHB bus rate, which is the same as the CPU clock rate (CCLK).
3. Interrupts: Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.
4. Programming: see [Section 33.6](#).

### 33.2 Introduction

---

The DMA controller allows peripheral-to memory, memory-to-peripheral, and memory-to-memory transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral.

### 33.3 Features

---

- Eight DMA channels. Each channel can support one unidirectional transfer.
- 16 DMA request lines.
- Memory-to-memory, memory-to-peripheral, and peripheral-to-memory transfers are supported.
- GPDMA supports the SD card interface, all SSPs, the I<sup>2</sup>S, all UARTs, the A/D Converter, and the D/A Converter peripherals. DMA can also be triggered by selected timer match conditions. Memory-to-memory transfers and transfers to or from GPIO are also supported.
- Scatter or gather DMA is supported through the use of linked lists. This means that the source and destination areas do not have to occupy contiguous areas of memory.
- Hardware DMA channel priority.
- AHB slave DMA programming interface. The DMA Controller is programmed by writing to the DMA control registers over the AHB slave interface.
- One AHB bus master for transferring data. The interface transfers data when a DMA request goes active.
- 32-bit AHB master bus width.
- Incrementing or non-incrementing addressing for source and destination.
- Programmable DMA burst size. The DMA burst size can be programmed to more efficiently transfer data.
- Internal four-word FIFO per channel.

- Supports 8-bit, 16-bit, and 32-bit wide transactions.
- Big-endian and little-endian support. The DMA Controller defaults to little-endian mode on reset.
- An interrupt to the processor can be generated on a DMA completion or when a DMA error has occurred.
- Raw interrupt status. The DMA error and DMA count raw interrupt status can be read prior to masking.
- DMA can operate in Sleep mode. (Note that in Sleep mode the GPDMA cannot access the flash memory or the main SRAM).

### 33.4 Functional description

This section describes the major functional blocks of the DMA Controller.

#### 33.4.1 DMA controller functional description

The DMA Controller enables peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral, and memory-to-memory transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral, and can be accessed through the AHB master. [Figure 159](#) shows a block diagram of the DMA Controller.

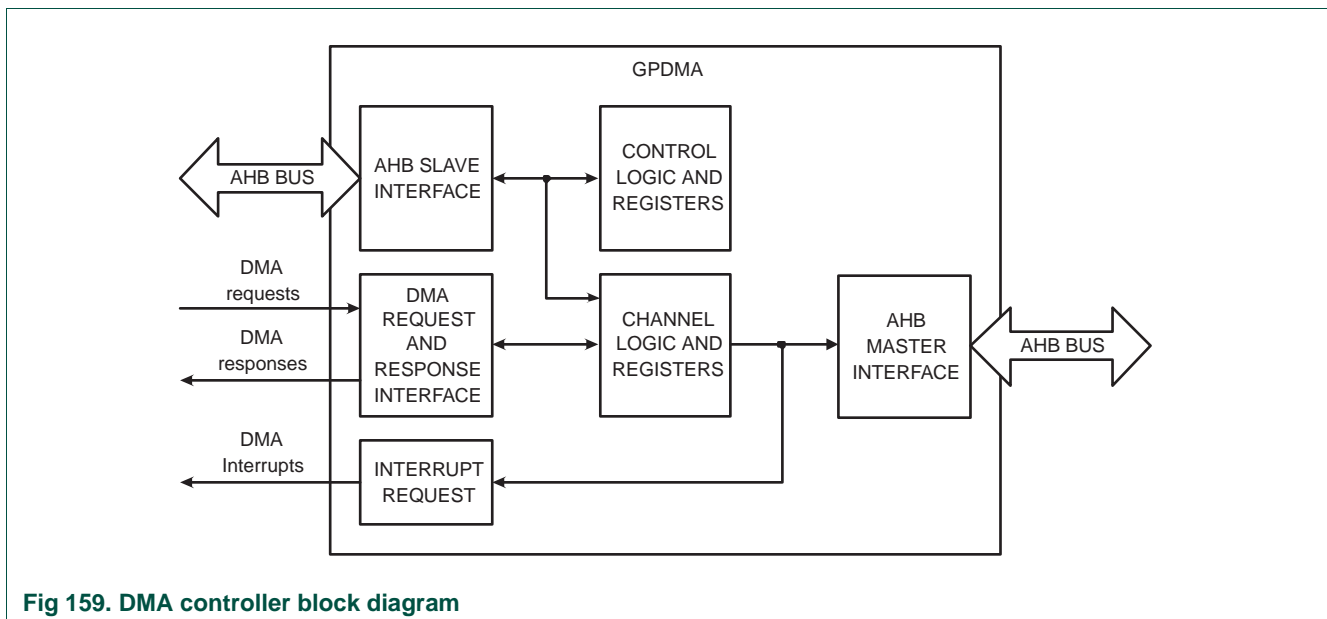


Fig 159. DMA controller block diagram

The functions of the DMA Controller are described in the following sections.

##### 33.4.1.1 AHB slave interface

All transactions to DMA Controller registers on the AHB slave interface are 32 bits wide. 8-bit and 16-bit accesses are not supported and will result in an exception.

### 33.4.1.2 Control logic and register bank

The register block stores data written or to be read across the AHB interface.

### 33.4.1.3 DMA request and response interface

See [Section 33.4.2](#) for information on the DMA request and response interface.

### 33.4.1.4 Channel logic and channel register bank

The channel logic and channel register bank contains registers and logic required for each DMA channel.

### 33.4.1.5 Interrupt request

The interrupt request generates the interrupt to the ARM processor.

### 33.4.1.6 AHB master interface

The DMA Controller contains one AHB master interface. The AHB master is capable of dealing with all types of AHB transactions, including:

- Split, retry, and error responses from slaves. If a peripheral performs a split or retry, the DMA Controller stalls and waits until the transaction can complete.
- Locked transfers for source and destination of each stream.
- Setting of protection bits for transfers on each stream.

#### 33.4.1.6.1 Bus and transfer widths

The physical width of the AHB bus is 32 bits. Source and destination transfers can be of differing widths and can be the same width or narrower than the physical bus width. The DMA Controller packs or unpacks data as appropriate.

#### 33.4.1.6.2 Endian behavior

The DMA Controller can cope with both little-endian and big-endian addressing.

Internally the DMA Controller treats all data as a stream of bytes instead of 16-bit or 32-bit quantities. This means that when performing mixed-endian activity, where the endianness of the source and destination are different, byte swapping of the data within the 32-bit data bus is observed.

Note: If byte swapping is not required, then use of different endianness between the source and destination addresses must be avoided. [Table 654](#) shows endian behavior for different source and destination combinations.

Table 654. Endian behavior

Source endian	Destination endian	Source width	Destination width	Source transfer no/byte lane	Source data	Destination transfer no/byte lane	Destination data
Little	Little	8	8	1/[7:0]	21	1/[7:0]	21212121
				2/[15:8]	43	2/[15:8]	43434343
				3/[23:16]	65	3/[23:16]	65656565
				4/[31:24]	87	4/[31:24]	87878787
Little	Little	8	16	1/[7:0]	21	1/[15:0]	43214321
				2/[15:8]	43	2/[31:16]	87658765
				3/[23:16]	65		
				4/[31:24]	87		
Little	Little	8	32	1/[7:0]	21	1/[31:0]	87654321
				2/[15:8]	43		
				3/[23:16]	65		
				4/[31:24]	87		
Little	Little	16	8	1/[7:0]	21	1/[7:0]	21212121
				1/[15:8]	43	2/[15:8]	43434343
				2/[23:16]	65	3/[23:16]	65656565
				2/[31:24]	87	4/[31:24]	87878787
Little	Little	16	16	1/[7:0]	21	1/[15:0]	43214321
				1/[15:8]	43	2/[31:16]	87658765
				2/[23:16]	65		
				2/[31:24]	87		
Little	Little	16	32	1/[7:0]	21	1/[31:0]	87654321
				1/[15:8]	43		
				2/[23:16]	65		
				2/[31:24]	87		
Little	Little	32	8	1/[7:0]	21	1/[7:0]	21212121
				1/[15:8]	43	2/[15:8]	43434343
				1/[23:16]	65	3/[23:16]	65656565
				1/[31:24]	87	4/[31:24]	87878787
Little	Little	32	16	1/[7:0]	21	1/[15:0]	43214321
				1/[15:8]	43	2/[31:16]	87658765
				1/[23:16]	65		
				1/[31:24]	87		
Little	Little	32	32	1/[7:0]	21	1/[31:0]	87654321
				1/[15:8]	43		
				1/[23:16]	65		
				1/[31:24]	87		
Big	Big	8	8	1/[31:24]	12	1/[31:24]	12121212
				2/[23:16]	34	2/[23:16]	34343434
				3/[15:8]	56	3/[15:8]	56565656
				4/[7:0]	78	4/[7:0]	78787878

Table 654. Endian behavior ...continued

Source endian	Destination endian	Source width	Destination width	Source transfer no/byte lane	Source data	Destination transfer no/byte lane	Destination data
Big	Big	8	16	1/[31:24]	12	1/[15:0]	12341234
				2/[23:16]	34	2/[31:16]	56785678
				3/[15:8]	56		
				4/[7:0]	78		
Big	Big	8	32	1/[31:24]	12	1/[31:0]	12345678
				2/[23:16]	34		
				3/[15:8]	56		
				4/[7:0]	78		
Big	Big	16	8	1/[31:24]	12	1/[31:24]	12121212
				1/[23:16]	34	2/[23:16]	34343434
				2/[15:8]	56	3/[15:8]	56565656
				2/[7:0]	78	4/[7:0]	78787878
Big	Big	16	16	1/[31:24]	12	1/[15:0]	12341234
				1/[23:16]	34	2/[31:16]	56785678
				2/[15:8]	56		
				2/[7:0]	78		
Big	Big	16	32	1/[31:24]	12	1/[31:0]	12345678
				1/[23:16]	34		
				2/[15:8]	56		
				2/[7:0]	78		
Big	Big	32	8	1/[31:24]	12	1/[31:24]	12121212
				1/[23:16]	34	2/[23:16]	34343434
				1/[15:8]	56	3/[15:8]	56565656
				1/[7:0]	78	4/[7:0]	78787878
Big	Big	32	16	1/[31:24]	12	1/[15:0]	12341234
				1/[23:16]	34	2/[31:16]	56785678
				1/[15:8]	56		
				1/[7:0]	78		
Big	Big	32	32	1/[31:24]	12	1/[31:0]	12345678
				1/[23:16]	34		
				1/[15:8]	56		
				1/[7:0]	78		

33.4.1.6.3 Error conditions

An error during a DMA transfer is flagged directly by the peripheral by asserting an Error response on the AHB bus during the transfer. The DMA Controller automatically disables the DMA stream after the current transfer has completed, and can optionally generate an error interrupt to the CPU. This error interrupt can be masked.



### 33.4.1.7 Channel hardware

Each stream is supported by a dedicated hardware channel, including source and destination controllers, as well as a FIFO. This enables better latency than a DMA controller with only a single hardware channel shared between several DMA streams and simplifies the control logic.

### 33.4.1.8 DMA request priority

DMA channel priority is fixed. DMA channel 0 has the highest priority and DMA channel 7 has the lowest priority.

If the DMA Controller is transferring data for the lower priority channel and then the higher priority channel goes active, it completes the number of transfers delegated to the master interface by the lower priority channel before switching over to transfer data for the higher priority channel. Transfers delegated to the master interface are staged in the DMA channel FIFO, so the amount of data that needs to transfer could be as large as a 4 words.

It is recommended that memory-to-memory transactions use the lowest priority channel.

### 33.4.1.9 Interrupt generation

A combined interrupt output is generated as an OR function of the individual interrupt requests of the DMA Controller and is connected to the interrupt controller.

## 33.4.2 DMA system connections

### 33.4.2.1 DMA request signals

The DMA request signals are used by peripherals to request a data transfer. The DMA request signals indicate whether a single or burst transfer of data is required. The DMA available request signals are:

**DMACBREQ[15:0]** — Burst request signals. These cause a programmed burst number of data to be transferred.

**DMACSREQ[15:0]** — Single transfer request signals. These cause a single data to be transferred. The DMA controller transfers a single transfer to or from the peripheral.

**DMACLBREQ[15:0]** — Last burst request signals.

**DMACLSREQ[15:0]** — Last single transfer request signals.

Note that most peripherals do not support “last” request types, and many peripherals do not support the single request type. See [Section 33.4.2.3](#).

### 33.4.2.2 DMA response signals

The DMA response signals indicate whether the transfer initiated by the DMA request signal has completed. The response signals can also be used to indicate whether a complete packet has been transferred. The DMA response signals from the DMA controller are:

**DMACCLR[15:0]** — DMA clear or acknowledge signals. The DMACCLR signal is used by the DMA controller to acknowledge a DMA request from the peripheral.

**DMACTC[15:0]** — DMA terminal count signals. The DMACTC signal can be used by the DMA controller to indicate to the peripheral that the DMA transfer is complete.

### 33.4.2.3 DMA request connections

The connection of the GPDMA to the supported peripheral devices depends on the DMA functions implemented in those peripherals. [Table 655](#) shows the DMA Request numbers used by the supported peripherals. Alternative requests on channels 0 through 7 and 10 through 15 are chosen via the DMACReqSel register, see [Section 33.5.15](#).

**Table 655. DMA connections**

DMA request	Burst Request	Single Request	Last Burst Request	Last Single Request	Comment
0	(unused) / T0_MAT[0]	-	-	-	Dedicated DMA request
1	SD card / T0_MAT[1]	SD card only	SD card only	SD card only	Dedicated DMA request
2	SSP0 Tx / T1_MAT[0]	SSP0 Tx only	-	-	Dedicated DMA request
3	SSP0 Rx / T1_MAT[1]	SSP0 Rx only	-	-	Dedicated DMA request
4	SSP1 Tx / T2_MAT[0]	SSP1 Tx only	-	-	Dedicated DMA request
5	SSP1 Rx / T2_MAT[1]	SSP1 Rx only	-	-	Dedicated DMA request
6	SSP2 TX / I <sup>2</sup> S channel 0	SSP2 Tx only	-	-	Dedicated DMA request
7	SSP2 Rx / I <sup>2</sup> S channel 1	SSP2 Rx only	-	-	Dedicated DMA request
8	ADC	-	-	-	ADC interrupt request <a href="#">[1]</a>
9	DAC	-	-	-	Dedicated DMA request
10	UART0 Tx / UART3 Tx	-	-	-	Dedicated DMA request
11	UART0 Rx / UART3 Rx	-	-	-	Dedicated DMA request
12	UART1 Tx / UART4 Tx	-	-	-	Dedicated DMA request
13	UART1 Rx / UART4 Rx	-	-	-	Dedicated DMA request
14	UART2 Tx / T3_MAT[0]	-	-	-	Dedicated DMA request
15	UART2 Rx / T3_MAT[1]	-	-	-	Dedicated DMA request

[1] Generates an interrupt and/or DMA request depending on software setup.

### 33.5 Register description

The DMA Controller supports 8 channels. Each channel has registers specific to the operation of that channel. Other registers controls aspects of how source peripherals relate to the DMA Controller. There are also global DMA control and status registers.

The DMA Controller registers are shown in [Table 656](#).

**Table 656. GPDMA register map**

Name	Description	Access	Reset Value	Address	Table
<b>General registers</b>					
DMACIntStat	DMA Interrupt Status Register	RO	0	0x2008 0000	<a href="#">657</a>
DMACIntTCStat	DMA Interrupt Terminal Count Request Status Register	RO	0	0x2008 0004	<a href="#">658</a>
DMACIntTCClear	DMA Interrupt Terminal Count Request Clear Register	WO	-	0x2008 0008	<a href="#">659</a>
DMACIntErrStat	DMA Interrupt Error Status Register	RO	0	0x2008 000C	<a href="#">660</a>
DMACIntErrClr	DMA Interrupt Error Clear Register	WO	-	0x2008 0010	<a href="#">661</a>
DMACRawIntTCStat	DMA Raw Interrupt Terminal Count Status Register	RO	0	0x2008 0014	<a href="#">662</a>
DMACRawIntErrStat	DMA Raw Error Interrupt Status Register	RO	0	0x2008 0018	<a href="#">663</a>
DMACEnbldChns	DMA Enabled Channel Register	RO	0	0x2008 001C	<a href="#">664</a>
DMACSoftBReq	DMA Software Burst Request Register	R/W	0	0x2008 0020	<a href="#">665</a>
DMACSoftSReq	DMA Software Single Request Register	R/W	0	0x2008 0024	<a href="#">666</a>
DMACSoftLBReq	DMA Software Last Burst Request Register	R/W	0	0x2008 0028	<a href="#">667</a>
DMACSoftLSReq	DMA Software Last Single Request Register	R/W	0	0x2008 002C	<a href="#">668</a>
DMACConfig	DMA Configuration Register	R/W	0	0x2008 0030	<a href="#">669</a>
DMACSync	DMA Synchronization Register	R/W	0	0x2008 0034	<a href="#">670</a>
DMACReqSel	Selects between alternative requests on channels 0 through 7 and 10 through 15	R/W	0	0x400F C1C4	<a href="#">671</a>
<b>Channel 0 registers</b>					
DMACC0SrcAddr	DMA Channel 0 Source Address Register	R/W	0	0x2008 0100	<a href="#">672</a>
DMACC0DestAddr	DMA Channel 0 Destination Address Register	R/W	0	0x2008 0104	<a href="#">673</a>
DMACC0LLI	DMA Channel 0 Linked List Item Register	R/W	0	0x2008 0108	<a href="#">674</a>
DMACC0Control	DMA Channel 0 Control Register	R/W	0	0x2008 010C	<a href="#">675</a>
DMACC0Config	DMA Channel 0 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 0110	<a href="#">676</a>
<b>Channel 1 registers</b>					
DMACC1SrcAddr	DMA Channel 1 Source Address Register	R/W	0	0x2008 0120	<a href="#">672</a>
DMACC1DestAddr	DMA Channel 1 Destination Address Register	R/W	0	0x2008 0124	<a href="#">673</a>
DMACC1LLI	DMA Channel 1 Linked List Item Register	R/W	0	0x2008 0128	<a href="#">674</a>
DMACC1Control	DMA Channel 1 Control Register	R/W	0	0x2008 012C	<a href="#">675</a>
DMACC1Config	DMA Channel 1 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 0130	<a href="#">676</a>
<b>Channel 2 registers</b>					
DMACC2SrcAddr	DMA Channel 2 Source Address Register	R/W	0	0x2008 0140	<a href="#">672</a>
DMACC2DestAddr	DMA Channel 2 Destination Address Register	R/W	0	0x2008 0144	<a href="#">673</a>
DMACC2LLI	DMA Channel 2 Linked List Item Register	R/W	0	0x2008 0148	<a href="#">674</a>
DMACC2Control	DMA Channel 2 Control Register	R/W	0	0x2008 014C	<a href="#">675</a>

Table 656. GPDMA register map

Name	Description	Access	Reset Value	Address	Table
DMACC2Config	DMA Channel 2 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 0150	<a href="#">676</a>
<b>Channel 3 registers</b>					
DMACC3SrcAddr	DMA Channel 3 Source Address Register	R/W	0	0x2008 0160	<a href="#">672</a>
DMACC3DestAddr	DMA Channel 3 Destination Address Register	R/W	0	0x2008 0164	<a href="#">673</a>
DMACC3LLI	DMA Channel 3 Linked List Item Register	R/W	0	0x2008 0168	<a href="#">674</a>
DMACC3Control	DMA Channel 3 Control Register	R/W	0	0x2008 016C	<a href="#">675</a>
DMACC3Config	DMA Channel 3 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 0170	<a href="#">676</a>
<b>Channel 4 registers</b>					
DMACC4SrcAddr	DMA Channel 4 Source Address Register	R/W	0	0x2008 0180	<a href="#">672</a>
DMACC4DestAddr	DMA Channel 4 Destination Address Register	R/W	0	0x2008 0184	<a href="#">673</a>
DMACC4LLI	DMA Channel 4 Linked List Item Register	R/W	0	0x2008 0188	<a href="#">674</a>
DMACC4Control	DMA Channel 4 Control Register	R/W	0	0x2008 018C	<a href="#">675</a>
DMACC4Config	DMA Channel 4 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 0190	<a href="#">676</a>
<b>Channel 5 registers</b>					
DMACC5SrcAddr	DMA Channel 5 Source Address Register	R/W	0	0x2008 01A0	<a href="#">672</a>
DMACC5DestAddr	DMA Channel 5 Destination Address Register	R/W	0	0x2008 01A4	<a href="#">673</a>
DMACC5LLI	DMA Channel 5 Linked List Item Register	R/W	0	0x2008 01A8	<a href="#">674</a>
DMACC5Control	DMA Channel 5 Control Register	R/W	0	0x2008 01AC	<a href="#">675</a>
DMACC5Config	DMA Channel 5 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 01B0	<a href="#">676</a>
<b>Channel 6 registers</b>					
DMACC6SrcAddr	DMA Channel 6 Source Address Register	R/W	0	0x2008 01C0	<a href="#">672</a>
DMACC6DestAddr	DMA Channel 6 Destination Address Register	R/W	0	0x2008 01C4	<a href="#">673</a>
DMACC6LLI	DMA Channel 6 Linked List Item Register	R/W	0	0x2008 01C8	<a href="#">674</a>
DMACC6Control	DMA Channel 6 Control Register	R/W	0	0x2008 01CC	<a href="#">675</a>
DMACC6Config	DMA Channel 6 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 01D0	<a href="#">676</a>
<b>Channel 7 registers</b>					
DMACC7SrcAddr	DMA Channel 7 Source Address Register	R/W	0	0x2008 01E0	<a href="#">672</a>
DMACC7DestAddr	DMA Channel 7 Destination Address Register	R/W	0	0x2008 01E4	<a href="#">673</a>
DMACC7LLI	DMA Channel 7 Linked List Item Register	R/W	0	0x2008 01E8	<a href="#">674</a>
DMACC7Control	DMA Channel 7 Control Register	R/W	0	0x2008 01EC	<a href="#">675</a>
DMACC7Config	DMA Channel 7 Configuration Register	R/W	0 <a href="#">[1]</a>	0x2008 01F0	<a href="#">676</a>

[1] Bit 17 of this register is a read-only status flag.

### 33.5.1 DMA Interrupt Status register (DMACIntStat - 0x2008 0000)

The DMACIntStat Register is read-only and shows the status of the interrupts after masking. A 1 bit indicates that a specific DMA channel interrupt request is active. The request can be generated from either the error or terminal count interrupt requests.

[Table 657](#) shows the bit assignments of the DMACIntStat Register.

**Table 657. DMA Interrupt Status register (DMACIntStat - 0x2008 0000)**

Bit	Name	Function
7:0	IntStat	Status of DMA channel interrupts after masking. Each bit represents one channel: 0 - the corresponding channel has no active interrupt request. 1 - the corresponding channel does have an active interrupt request.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.2 DMA Interrupt Terminal Count Request Status register (DMACIntTCStat - 0x2008 0004)

The DMACIntTCStat Register is read-only and indicates the status of the terminal count after masking. [Table 658](#) shows the bit assignments of the DMACIntTCStat Register.

**Table 658. DMA Interrupt Terminal Count Request Status register (DMACIntTCStat - 0x2008 0004)**

Bit	Name	Function
7:0	IntTCStat	Terminal count interrupt request status for DMA channels. Each bit represents one channel: 0 - the corresponding channel has no active terminal count interrupt request. 1 - the corresponding channel does have an active terminal count interrupt request.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.3 DMA Interrupt Terminal Count Request Clear register (DMACIntTCClear - 0x2008 0008)

The DMACIntTCClear Register is write-only and clears one or more terminal count interrupt requests. When writing to this register, each data bit that contains a 1 causes the corresponding bit in the status register (DMACIntTCStat) to be cleared. Data bits that are 0 have no effect. [Table 659](#) shows the bit assignments of the DMACIntTCClear Register.

**Table 659. DMA Interrupt Terminal Count Request Clear register (DMACIntTCClear - 0x2008 0008)**

Bit	Name	Function
7:0	IntTCClear	Allows clearing the Terminal count interrupt request (IntTCStat) for DMA channels. Each bit represents one channel: 0 - writing 0 has no effect. 1 - clears the corresponding channel terminal count interrupt.
31:8	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.4 DMA Interrupt Error Status register (DMACIntErrStat - 0x2008 000C)

The DMACIntErrStat Register is read-only and indicates the status of the error request after masking. [Table 660](#) shows the bit assignments of the DMACIntErrStat Register.

**Table 660. DMA Interrupt Error Status register (DMACIntErrStat - 0x2008 000C)**

Bit	Name	Function
7:0	IntErrStat	Interrupt error status for DMA channels. Each bit represents one channel: 0 - the corresponding channel has no active error interrupt request. 1 - the corresponding channel does have an active error interrupt request.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.5 DMA Interrupt Error Clear register (DMACIntErrClr - 0x2008 0010)

The DMACIntErrClr Register is write-only and clears the error interrupt requests. When writing to this register, each data bit that is 1 causes the corresponding bit in the status register to be cleared. Data bits that are 0 have no effect on the corresponding bit in the register. [Table 661](#) shows the bit assignments of the DMACIntErrClr Register.

**Table 661. DMA Interrupt Error Clear register (DMACIntErrClr - 0x2008 0010)**

Bit	Name	Function
7:0	IntErrClr	Writing a 1 clears the error interrupt request (IntErrStat) for DMA channels. Each bit represents one channel: 0 - writing 0 has no effect. 1 - clears the corresponding channel error interrupt.
31:8	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.6 DMA Raw Interrupt Terminal Count Status register (DMACRawIntTCStat - 0x2008 0014)

The DMACRawIntTCStat Register is read-only and indicates which DMA channel is requesting a transfer complete (terminal count interrupt) prior to masking. (Note: the DMACIntTCStat Register contains the same information after masking.) A 1 bit indicates that the terminal count interrupt request is active prior to masking. [Table 662](#) shows the bit assignments of the DMACRawIntTCStat Register.

**Table 662. DMA Raw Interrupt Terminal Count Status register (DMACRawIntTCStat - 0x2008 0014)**

Bit	Name	Function
7:0	RawIntTCStat	Status of the terminal count interrupt for DMA channels prior to masking. Each bit represents one channel: 0 - the corresponding channel has no active terminal count interrupt request. 1 - the corresponding channel does have an active terminal count interrupt request.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.7 DMA Raw Error Interrupt Status register (DMACRawIntErrStat - 0x2008 0018)

The DMACRawIntErrStat Register is read-only and indicates which DMA channel is requesting an error interrupt prior to masking. (Note: the DMACIntErrStat Register contains the same information after masking.) A 1 bit indicates that the error interrupt request is active prior to masking. [Table 663](#) shows the bit assignments of register of the DMACRawIntErrStat Register.

**Table 663. DMA Raw Error Interrupt Status register (DMACRawIntErrStat - 0x2008 0018)**

Bit	Name	Function
7:0	RawIntErrStat	Status of the error interrupt for DMA channels prior to masking. Each bit represents one channel: 0 - the corresponding channel has no active error interrupt request. 1 - the corresponding channel does have an active error interrupt request.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.8 DMA Enabled Channel register (DMACEnbldChns - 0x2008 001C)

The DMACEnbldChns Register is read-only and indicates which DMA channels are enabled, as indicated by the Enable bit in the DMACCxConfig Register. A 1 bit indicates that a DMA channel is enabled. A bit is cleared on completion of the DMA transfer. [Table 664](#) shows the bit assignments of the DMACEnbldChns Register.

**Table 664. DMA Enabled Channel register (DMACEnbldChns - 0x2008 001C)**

Bit	Name	Function
7:0	EnabledChannels	Enable status for DMA channels. Each bit represents one channel: 0 - DMA channel is disabled. 1 - DMA channel is enabled.
31:8	-	Reserved. The value read from a reserved bit is not defined.

### 33.5.9 DMA Software Burst Request register (DMACSoftBReq - 0x2008 0020)

The DMACSoftBReq Register is read/write and enables DMA burst requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting DMA burst transfers. A request can be generated from either a peripheral or the software request register. Each bit is cleared when the related transaction has completed. [Table 665](#) shows the bit assignments of the DMACSoftBReq Register.

**Table 665. DMA Software Burst Request register (DMACSoftBReq - 0x2008 0020)**

Bit	Name	Function
15:0	SoftBReq	Software burst request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function (refer to <a href="#">Table 655</a> for peripheral hardware connections to the DMA controller): 0 - writing 0 has no effect. 1 - writing 1 generates a DMA burst request for the corresponding request line.
31:16	-	Reserved. Read value is undefined, only zero should be written.

**Note:** It is recommended that software and hardware peripheral requests are not used at the same time.

### 33.5.10 DMA Software Single Request register (DMACSoftSReq - 0x2008 0024)

The DMACSoftSReq Register is read/write and enables DMA single transfer requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has

completed. Reading the register indicates which sources are requesting single DMA transfers. A request can be generated from either a peripheral or the software request register. [Table 666](#) shows the bit assignments of the DMACSoftSReq Register.

**Table 666. DMA Software Single Request register (DMACSoftSReq - 0x2008 0024)**

Bit	Name	Function
15:0	SoftSReq	Software single transfer request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function: 0 - writing 0 has no effect. 1 - writing 1 generates a DMA single transfer request for the corresponding request line.
31:16	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.11 DMA Software Last Burst Request register (DMACSoftLBReq - 0x2008 0028)

The DMACSoftLBReq Register is read/write and enables DMA last burst requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting last burst DMA transfers. A request can be generated from either a peripheral or the software request register. [Table 667](#) shows the bit assignments of the DMACSoftLBReq Register.

**Table 667. DMA Software Last Burst Request register (DMACSoftLBReq - 0x2008 0028)**

Bit	Name	Function
15:0	SoftLBReq	Software last burst request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function: 0 - writing 0 has no effect. 1 - writing 1 generates a DMA last burst request for the corresponding request line.
31:16	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.12 DMA Software Last Single Request register (DMACSoftLSReq - 0x2008 002C)

The DMACSoftLSReq Register is read/write and enables DMA last single requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting last single DMA transfers. A request can be generated from either a peripheral or the software request register. [Table 668](#) shows the bit assignments of the DMACSoftLSReq Register.

**Table 668. DMA Software Last Single Request register (DMACSoftLSReq - 0x2008 002C)**

Bit	Name	Function
15:0	SoftLSReq	Software last single transfer request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function: 0 - writing 0 has no effect. 1 - writing 1 generates a DMA last single transfer request for the corresponding request line.
31:16	-	Reserved. Read value is undefined, only zero should be written.



### 33.5.13 DMA Configuration register (DMACConfig - 0x2008 0030)

The DMACConfig Register is read/write and configures the operation of the DMA Controller. The endianness of the AHB master interface can be altered by writing to the M bit of this register. The AHB master interface is set to little-endian mode on reset.

[Table 669](#) shows the bit assignments of the DMACConfig Register.

**Table 669. DMA Configuration register (DMACConfig - 0x2008 0030)**

Bit	Name	Function
0	E	DMA Controller enable: 0 = disabled (default). Disabling the DMA Controller reduces power consumption. 1 = enabled.
1	M	AHB Master endianness configuration: 0 = little-endian mode (default). 1 = big-endian mode.
31:2	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.14 DMA Synchronization register (DMACSync - 0x2008 0034)

The DMACSync Register is read/write and enables or disables synchronization logic for the DMA request signals. The DMA request signals consist of the DMACBREQ[15:0], DMACSREQ[15:0], DMACLBREQ[15:0], and DMACLSREQ[15:0]. A bit set to 0 enables the synchronization logic for a particular group of DMA requests. A bit set to 1 disables the synchronization logic for a particular group of DMA requests. This register is reset to 0, enabling synchronization logic by default. [Table 670](#) shows the bit assignments of the DMACSync Register.

**Table 670. DMA Synchronization register (DMACSync - 0x2008 0034)**

Bit	Name	Function
15:0	DMACSync	Controls the synchronization logic for DMA request signals. Each bit represents one set of DMA request lines as described in the preceding text: 0 - synchronization logic for the corresponding DMA request signals are enabled. 1 - synchronization logic for the corresponding DMA request signals are disabled.
31:16	-	Reserved. Read value is undefined, only zero should be written.

**33.5.15 DMA Request Select register (DMACReqSel - 0x400F C1C4)**

DMACReqSel is a read/write register that allows selecting between potential DMA requests for DMA inputs 0 through 7 and 10 through 15. [Table 671](#) shows the bit assignments of the DMACReqSel Register.

**Table 671. DMA Request Select register (DMACReqSel - 0x400F C1C4)**

Bit	Name	Function
0	DMASEL00	Selects the DMA request for GPDMA input 0: 0 - (unused) 1 - Timer 0 match 0 is selected.
1	DMASEL01	Selects the DMA request for GPDMA input 1: 0 - SD card interface is selected. 1 - Timer 0 match 1 is selected.
2	DMASEL02	Selects the DMA request for GPDMA input 2: 0 - SSP0 transmit is selected. 1 - Timer 1 match 0 is selected.
3	DMASEL03	Selects the DMA request for GPDMA input 3: 0 - SSP0 receive is selected. 1 - Timer 1 match 1 is selected.
4	DMASEL04	Selects the DMA request for GPDMA input 4: 0 - SSP1 transmit is selected. 1 - Timer 2 match 0 is selected.
5	DMASEL05	Selects the DMA request for GPDMA input 5: 0 - SSP1 receive is selected. 1 - Timer 2 match 1 is selected.
6	DMASEL06	Selects the DMA request for GPDMA input 6: 0 - SSP2 transmit is selected. 1 - I <sup>2</sup> S channel 0 is selected.
7	DMASEL07	Selects the DMA request for GPDMA input 7: 0 - SSP2 receive is selected. 1 - I <sup>2</sup> S channel 1 is selected.
9:8	-	Reserved. Read value is undefined, only zero should be written.
10	DMASEL10	Selects the DMA request for GPDMA input 10: 0 - UART0 transmit is selected. 1 - UART3 transmit is selected.
11	DMASEL11	Selects the DMA request for GPDMA input 11: 0 - UART0 receive is selected. 1 - UART3 receive is selected.
12	DMASEL12	Selects the DMA request for GPDMA input 12: 0 - UART1 transmit is selected. 1 - UART4 transmit is selected.
13	DMASEL13	Selects the DMA request for GPDMA input 13: 0 - UART1 receive is selected. 1 - UART4 receive is selected.
14	DMASEL14	Selects the DMA request for GPDMA input 14: 0 - UART2 transmit is selected. 1 - Timer 3 match 0 is selected.
15	DMASEL15	Selects the DMA request for GPDMA input 15: 0 - UART2 receive is selected. 1 - Timer 3 match 1 is selected.
31:16	-	Reserved. Read value is undefined, only zero should be written.

**33.5.15.1 Timer DMA requests**

Timer DMA requests are generated by the timer when the timer value matches the related Match register (see [Section 24.6.12](#)). If the DMA controller is configured so that a timer DMA request is selected as an input to a DMA channel, and the DMA channel is enabled, the DMA controller will act on that request.

### 33.5.16 DMA Channel registers

The channel registers are used to program the eight DMA channels. These registers consist of:

- Eight DMACCxSrcAddr Registers.
- Eight DMACCxDestAddr Registers.
- Eight DMACCxLLI Registers.
- Eight DMACCxControl Registers.
- Eight DMACCxConfig Registers.

When performing scatter/gather DMA, the first four of these are automatically updated.

### 33.5.17 DMA Channel Source Address registers (DMACCxSrcAddr - 0x2008 01x0)

The eight read/write DMACCxSrcAddr Registers (DMACC0SrcAddr to DMACC7SrcAddr) contain the current source address (byte-aligned) of the data to be transferred. Each register is programmed directly by software before the appropriate channel is enabled. When the DMA channel is enabled this register is updated:

- As the source address is incremented.
- By following the linked list when a complete packet of data has been transferred.

Reading the register when the channel is active does not provide useful information. This is because by the time software has processed the value read, the address may have progressed. It is intended to be read-only when the channel has stopped, in which case it shows the source address of the last item read.

Note: The source and destination addresses must be aligned to the source and destination widths.

[Table 672](#) shows the bit assignments of the DMACCxSrcAddr Registers.

**Table 672. DMA Channel Source Address registers (DMACCxSrcAddr - 0x2008 01x0)**

Bit	Name	Function
31:0	SrcAddr	DMA source address. Reading this register will return the current source address.

### 33.5.18 DMA Channel Destination Address registers (DMACCxDestAddr - 0x2008 01x4)

The eight read/write DMACCxDestAddr Registers (DMACC0DestAddr to DMACC7DestAddr) contain the current destination address (byte-aligned) of the data to be transferred. Each register is programmed directly by software before the channel is enabled. When the DMA channel is enabled the register is updated as the destination address is incremented and by following the linked list when a complete packet of data has been transferred. Reading the register when the channel is active does not provide useful information. This is because by the time that software has processed the value read, the address may have progressed. It is intended to be read-only when a channel has stopped, in which case it shows the destination address of the last item read.

[Table 673](#) shows the bit assignments of the DMACCxDestAddr Register.

**Table 673. DMA Channel Destination Address registers (DMACCxDestAddr - 0x2008 01x4)**

Bit	Name	Function
31:0	DestAddr	DMA Destination address. Reading this register will return the current destination address.

### 33.5.19 DMA Channel Linked List Item registers (DMACCxLLI - 0x2008 01x8)

The eight read/write DMACCxLLI Registers (DMACC0LLI to DMACC7LLI) contain a word-aligned address of the next Linked List Item (LLI). If the LLI is 0, then the current LLI is the last in the chain, and the DMA channel is disabled when all DMA transfers associated with it are completed. Programming this register when the DMA channel is enabled may have unpredictable side effects. [Table 674](#) shows the bit assignments of the DMACCxLLI Register.

**Table 674. DMA Channel Linked List Item registers (DMACCxLLI - 0x2008 01x8)**

Bit	Name	Function
1:0	-	Reserved, and must be written as 0.
31:2	LLI	Linked list item. Bits [31:2] of the address for the next LLI. Address bits [1:0] are 0.

### 33.5.20 DMA channel control registers (DMACCxControl - 0x2008 01xC)

The eight read/write DMACCxControl Registers (DMACC0Control to DMACC7Control) contain DMA channel control information such as the transfer size, burst size, and transfer width. Each register is programmed directly by software before the DMA channel is enabled. When the channel is enabled the register is updated by following the linked list when a complete packet of data has been transferred. Reading the register while the channel is active does not give useful information. This is because by the time software has processed the value read, the channel may have advanced. It is intended to be read-only when a channel has stopped. [Table 675](#) shows the bit assignments of the DMACCxControl Register.

#### 33.5.20.1 Protection and access information

AHB access information is provided to the source and/or destination peripherals when a transfer occurs, although on the LPC178x/177x this has no effect. The transfer information is provided by programming the DMA channel (the Prot bits of the DMACCxControl Register, and the Lock bit of the DMACCxConfig Register). These bits are programmed by software, and can be used by peripherals. Three bits of information are provided, and are used as shown in [Table 675](#).

Table 675. DMA channel control registers (DMACCxControl - 0x2008 01xC)

Bit	Name	Function
11:0	TransferSize	<p>Transfer size. This field sets the size of the transfer when the DMA controller is the flow controller, in which case the value must be set before the channel is enabled. Transfer size is updated as data transfers are completed.</p> <p>A read from this field indicates the number of transfers completed on the destination bus. Reading the register when the channel is active does not give useful information because by the time that the software has processed the value read, the channel might have progressed. It is intended to be used only when a channel is enabled and then disabled. The transfer size value is not used if a peripheral is the flow controller.</p>
14:12	SBSize	<p>Source burst size. Indicates the number of transfers that make up a source burst. This value must be set to the burst size of the source peripheral, or if the source is memory, to the memory boundary size. The burst size is the amount of data that is transferred when the DMACBREQ signal goes active in the source peripheral.</p> <p>000 - 1                      001 - 4                      010 - 8                      011 - 16                      100 - 32                      101 - 64                      110 - 128                      111 - 256</p>
17:15	DBSize	<p>Destination burst size. Indicates the number of transfers that make up a destination burst transfer request. This value must be set to the burst size of the destination peripheral or, if the destination is memory, to the memory boundary size. The burst size is the amount of data that is transferred when the DMACBREQ signal goes active in the destination peripheral.</p> <p>000 - 1                      001 - 4                      010 - 8                      011 - 16                      100 - 32                      101 - 64                      110 - 128                      111 - 256</p>
20:18	SWidth	<p>Source transfer width. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data as required.</p> <p>000 - Byte (8-bit)                      001 - Halfword (16-bit)                      010 - Word (32-bit)                      011 to 111 - Reserved</p>
23:21	DWidth	<p>Destination transfer width. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data as required.</p> <p>000 - Byte (8-bit)                      001 - Halfword (16-bit)                      010 - Word (32-bit)                      011 to 111 - Reserved</p>
25:24	-	Reserved, and must be written as 0.

**Table 675. DMA channel control registers (DMACCxControl - 0x2008 01xC) ...continued**

Bit	Name	Function
26	SI	Source increment: 0 - the source address is not incremented after each transfer. 1 - the source address is incremented after each transfer.
27	DI	Destination increment: 0 - the destination address is not incremented after each transfer. 1 - the destination address is incremented after each transfer.
28	Prot1	This is provided to the peripheral during a DMA bus access and indicates that the access is in user mode or privileged mode. This information is not used in the LPC178x/177x. 0 - access is in user mode. 1 - access is in privileged mode.
29	Prot2	This is provided to the peripheral during a DMA bus access and indicates to the peripheral that the access is bufferable or not bufferable. This information is not used in the LPC178x/177x. 0 - access is not bufferable. 1 - access is bufferable.
30	Prot3	This is provided to the peripheral during a DMA bus access and indicates to the peripheral that the access is cacheable or not cacheable. This information is not used in the LPC178x/177x. 0 - access is not cacheable. 1 - access is cacheable.
31	I	Terminal count interrupt enable bit. 0 - the terminal count interrupt is disabled. 1 - the terminal count interrupt is enabled.

### 33.5.21 DMA Channel Configuration registers (DMACCxConfig - 0x2008 01x0)

The eight DMACCxConfig Registers (DMACC0Config to DMACC7Config) are read/write with the exception of bit[17] which is read-only. These registers configure each DMA channel. The registers are not updated when a new LLI is requested. [Table 676](#) shows the bit assignments of the DMACCxConfig Register.

Table 676. DMA Channel Configuration registers (DMACCxConfig - 0x2008 01x0)

Bit	Name	Function
0	E	<p>Channel enable. Reading this bit indicates whether a channel is currently enabled or disabled:                      0 = channel disabled.                      1 = channel enabled.</p> <p>The Channel Enable bit status can also be found by reading the DMACEnbldChns Register.                      A channel is enabled by setting this bit.</p> <p>A channel can be disabled by clearing the Enable bit. This causes the current AHB transfer (if one is in progress) to complete and the channel is then disabled. Any data in the FIFO of the relevant channel is lost. Restarting the channel by setting the Channel Enable bit has unpredictable effects, the channel must be fully re-initialized.</p> <p>The channel is also disabled, and Channel Enable bit cleared, when the last LLI is reached, the DMA transfer is completed, or if a channel error is encountered.</p> <p>If a channel must be disabled without losing data in the FIFO, the Halt bit must be set so that further DMA requests are ignored. The Active bit must then be polled until it reaches 0, indicating that there is no data left in the FIFO. Finally, the Channel Enable bit can be cleared.</p>
5:1	SrcPeripheral	Source peripheral. This value selects the DMA source request peripheral. This field is ignored if the source of the transfer is from memory. See <a href="#">Table 655</a> for peripheral identification.
10:6	DestPeripheral	Destination peripheral. This value selects the DMA destination request peripheral. This field is ignored if the destination of the transfer is to memory. See <a href="#">Table 655</a> for peripheral identification.
13:11	TransferType	<p>This value indicates the type of transfer and specifies the flow controller. The transfer type can be memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-to-peripheral. Flow can be controlled by the DMA controller, the source peripheral, or the destination peripheral.</p> <p>Refer to <a href="#">Table 677</a> for the encoding of this field.</p>
14	IE	Interrupt error mask. When cleared, this bit masks out the error interrupt of the relevant channel.
15	ITC	Terminal count interrupt mask. When cleared, this bit masks out the terminal count interrupt of the relevant channel.
16	L	Lock. When set, this bit enables locked transfers. This information is not used in the LPC178x/177x.
17	A	<p>Active:</p> <p>0 = there is no data in the FIFO of the channel.                      1 = the channel FIFO has data.</p> <p>This value can be used with the Halt and Channel Enable bits to cleanly disable a DMA channel. This is a read-only bit.</p>
18	H	<p>Halt:</p> <p>0 = enable DMA requests.                      1 = ignore further source DMA requests.</p> <p>The contents of the channel FIFO are drained.</p> <p>This value can be used with the Active and Channel Enable bits to cleanly disable a DMA channel.</p>
31:19	-	Reserved. Read value is undefined, only zero should be written.

### 33.5.21.1 Lock control

The lock control may set the lock bit by writing a 1 to bit 16 of the DMACCxConfig Register. When a burst occurs, the AHB arbiter will not de-grant the master during the burst until the lock is de-asserted. The DMA Controller can be locked for a single burst

such as a long source fetch burst or a long destination drain burst. The DMA Controller does not usually assert the lock continuously for a source fetch burst followed by a destination drain burst.

There are situations when the DMA Controller asserts the lock for source transfers followed by destination transfers. This is possible when internal conditions in the DMA Controller permit it to perform a source fetch followed by a destination drain back-to-back.

### 33.5.21.2 Transfer type

[Table 677](#) lists the bit values of the transfer type bits identified in [Table 676](#).

**Table 677. Transfer type bits**

Bit value	Transfer type	Flow controller
000	Memory to memory	DMA controller
001	Memory to peripheral	DMA controller
010	Peripheral to memory	DMA controller
011	Source peripheral to destination peripheral	DMA controller
100	Source peripheral to destination peripheral	Destination peripheral
101	Memory to peripheral	Destination peripheral
110	Peripheral to memory	Source peripheral
111	Source peripheral to destination peripheral	Source peripheral



## 33.6 Using the DMA controller

### 33.6.1 Programming the DMA controller

All accesses to the DMA Controller internal register must be word (32-bit) reads and writes.

#### 33.6.1.1 Enabling the DMA controller

To enable the DMA controller set the Enable bit in the DMACConfig register.

#### 33.6.1.2 Disabling the DMA controller

To disable the DMA controller:

- Read the DMACEnbldChns register and ensure that all the DMA channels have been disabled. If any channels are active, see Disabling a DMA channel.
- Disable the DMA controller by writing 0 to the DMA Enable bit in the DMACConfig register.

#### 33.6.1.3 Enabling a DMA channel

To enable the DMA channel set the channel enable bit in the relevant DMA channel configuration register. Note that the channel must be fully initialized before it is enabled.

#### 33.6.1.4 Disabling a DMA channel

A DMA channel can be disabled in three ways:

- By writing directly to the channel enable bit. Any outstanding data in the FIFO's is lost if this method is used.
- By using the active and halt bits in conjunction with the channel enable bit.
- By waiting until the transfer completes. This automatically clears the channel.

#### Disabling a DMA channel and losing data in the FIFO

Clear the relevant channel enable bit in the relevant channel configuration register. The current AHB transfer (if one is in progress) completes and the channel is disabled. Any data in the FIFO is lost. The channel must be fully re-initialized before it is enabled again.

#### Disabling the DMA channel without losing data in the FIFO

- Set the halt bit in the relevant channel configuration register. This causes any future DMA request to be ignored.
- Poll the active bit in the relevant channel configuration register until it reaches 0. This bit indicates whether there is any data in the channel that has to be transferred.
- Clear the channel enable bit in the relevant channel configuration register

#### 33.6.1.5 Setting up a new DMA transfer

To set up a new DMA transfer:

If the channel is not set aside for the DMA transaction:

1. Read the DMACEnbldChns controller register and find out which channels are inactive.
2. Choose an inactive channel that has the required priority.
3. Program the DMA controller

#### 33.6.1.6 Halting a DMA channel

Set the halt bit in the relevant DMA channel configuration register. The current source request is serviced. Any further source DMA request is ignored until the halt bit is cleared.

#### 33.6.1.7 Programming a DMA channel

1. Choose a free DMA channel with the priority needed. DMA channel 0 has the highest priority and DMA channel 7 the lowest priority.
2. Clear any pending interrupts on the channel to be used by writing to the DMACIntTCClear and DMACIntErrClear register. The previous channel operation might have left interrupt active.
3. Write the source address into the DMACCxSrcAddr register.
4. Write the destination address into the DMACCxDestAddr register.
5. Write the address of the next LLI into the DMACCxLLI register. If the transfer comprises of a single packet of data then 0 must be written into this register.
6. Write the control information into the DMACCxControl register.
7. Write the channel configuration information into the DMACCxConfig register. If the enable bit is set then the DMA channel is automatically enabled.

#### 33.6.2 Flow control

The device that controls the length of the packet is known as the flow controller. The flow controller is usually the DMA controller, where the packet length is programmed by software before the DMA channel is enabled. Most peripherals are not able to be the flow controller, but when this feature is supported, it can be used by either the source or destination peripheral.

When the DMA transfer is completed:

1. The DMA Controller issues an acknowledge to the peripheral in order to indicate that the transfer has finished.
2. A TC interrupt is generated, if enabled.
3. The DMA Controller moves on to the next LLI.

The following sections describe the DMA Controller data flow sequences for the four allowed transfer types:

- Memory-to-peripheral.
- Peripheral-to-memory.
- Memory-to-memory.
- Peripheral-to-peripheral.

Each transfer type other than memory-to-memory can have either the peripheral or the DMA controller as the flow controller, resulting in 8 possible control scenarios.

Table 678 indicates the request signals used for each type of transfer.

**Table 678. DMA request signal usage**

Transfer direction	Request generator	Flow controller
Memory-to-peripheral	Destination peripheral	DMA Controller
Peripheral-to-memory	Source peripheral	DMA Controller
Memory-to-memory	DMA Controller	DMA Controller
Source peripheral to destination peripheral	Both source and destination peripherals	DMA Controller
Memory-to-peripheral	Destination peripheral	Destination peripheral
Peripheral-to-memory	Source peripheral	Source peripheral
Source peripheral to destination peripheral	Both source and destination peripherals	Source peripheral
Source peripheral to destination peripheral	Both source and destination peripherals	Destination peripheral

### 33.6.2.1 Peripheral-to-memory or memory-to-peripheral DMA flow

For a peripheral-to-memory or memory-to-peripheral DMA flow, the following sequence occurs:

1. Program and enable the DMA channel.
2. Wait for a DMA request.
3. The DMA Controller starts transferring data when:
  - The DMA request goes active.
  - The DMA stream has the highest pending priority.
  - The DMA Controller is the bus master of the AHB bus.
4. If an error occurs while transferring the data, an error interrupt is generated and disables the DMA stream, and the flow sequence ends.
5. Decrement the transfer count.
6. If the transfer has completed (indicated by the transfer count reaching 0 if the DMA controller is performing flow control, or by the peripheral sending a DMA request if the peripheral is performing flow control):
  - The DMA Controller responds with a DMA acknowledge.
  - The terminal count interrupt is generated (this interrupt can be masked).
  - If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

### 33.6.2.2 Peripheral-to-peripheral DMA flow

For a peripheral-to-peripheral DMA flow, the following sequence occurs:

1. Program and enable the DMA channel.
2. Wait for a source DMA request.
3. The DMA Controller starts transferring data when:
  - The DMA request goes active.
  - The DMA stream has the highest pending priority.
  - The DMA Controller is the bus master of the AHB bus.

4. If an error occurs while transferring the data an error interrupt is generated, the DMA stream is disabled, and the flow sequence ends.
5. Decrement the transfer count.
6. If the transfer has completed (indicated by the transfer count reaching 0 if the DMA controller is performing flow control, or by the peripheral sending a DMA request if the peripheral is performing flow control):
  - The DMA Controller responds with a DMA acknowledge to the source peripheral.
  - Further source DMA requests are ignored.
7. When the destination DMA request goes active and there is data in the DMA Controller FIFO, transfer data into the destination peripheral.
8. If an error occurs while transferring the data, an error interrupt is generated, the DMA stream is disabled, and the flow sequence ends.
9. If the transfer has completed it is indicated by the transfer count reaching 0 if the DMA controller is the flow controller. or by the peripheral sending a DMA request if the peripheral is performing flow control. The following happens:
  - The DMA Controller responds with a DMA acknowledge to the destination peripheral.
  - The terminal count interrupt is generated (this interrupt can be masked).
  - If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl Registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

### 33.6.2.3 Memory-to-memory DMA flow

For a memory-to-memory DMA flow the following sequence occurs:

1. Program and enable the DMA channel.
2. Transfer data whenever the DMA channel has the highest pending priority and the DMA Controller gains mastership of the AHB bus.
3. If an error occurs while transferring the data, generate an error interrupt and disable the DMA stream.
4. Decrement the transfer count.
5. If the count has reached zero:
  - Generate a terminal count interrupt (the interrupt can be masked).
  - If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl Registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

**Note:** Memory-to-memory transfers should be programmed with a low channel priority, otherwise other DMA channels cannot access the bus until the memory-to-memory transfer has finished, or other AHB masters cannot perform any transaction.

### 33.6.3 Interrupt requests

Interrupt requests can be generated when an AHB error is encountered or at the end of a transfer (terminal count), after all the data corresponding to the current LLI has been transferred to the destination. The interrupts can be masked by programming bits in the relevant DMACCxControl and DMACCxConfig Channel Registers. The interrupt requests from all DMA channels can be found in the DMACRawIntTCStat and DMACRawIntErrStat registers. The masked versions of the DMA interrupt data is contained in the DMACIntTCStat and DMACIntErrStat registers. The DMACIntStat register then combines the DMACIntTCStat and DMACIntErrStat requests into a single register to enable the source of an interrupt to be found quickly. Writing to the DMACIntTCClear or the DMACIntErrClr Registers with a bit set to 1 enables selective clearing of interrupts.

#### 33.6.3.1 Hardware interrupt sequence flow

When a DMA interrupt request occurs, the Interrupt Service Routine needs to:

1. Read the DMACIntTCStat Register to determine whether the interrupt was generated due to the end of the transfer (terminal count). A 1 bit indicates that the transfer completed. If more than one request is active, it is recommended that the highest priority channels be checked first.
2. Read the DMACIntErrStat Register to determine whether the interrupt was generated due to an error occurring. A 1 bit indicates that an error occurred.
3. Service the interrupt request. The channel that caused the interrupt can be determined by reading the DMACIntStat register. If more than one request is active, the one with the highest priority should generally be serviced first.
4. For a terminal count interrupt, write a 1 to the relevant bit of the DMACIntTCClr Register. For an error interrupt write a 1 to the relevant bit of the DMACIntErrClr Register to clear the interrupt request.

### 33.6.4 Address generation

Address generation can be either incrementing or non-incrementing (address wrapping is not supported).

Some devices, especially memories, disallow burst accesses across certain address boundaries. The DMA controller assumes that this is the case with any source or destination area that is configured for incrementing addressing. This boundary is assumed to be aligned with the specified burst size. For example, if the channel is set for 16-transfer burst to a 32-bit wide device then the boundary is 64-bytes aligned (that is address bits [5:0] equal 0). If a DMA burst is to cross one of these boundaries, then, instead of a burst, that transfer is split into separate AHB transactions.

#### 33.6.4.1 Word-aligned transfers across a boundary

The channel is configured for 16-transfer bursts, each transfer 32-bits wide, to a destination for which address incrementing is enabled. The start address for the current burst is 0x0C00 0024, the next boundary (calculated from the burst size and transfer width) is 0x0C00 0040.

The transfer will be split into two AHB transactions:

- a 7-transfer burst starting at address 0x0C00 0024

- a 9-transfer burst starting at address 0x0C00 0040.

### 33.6.5 Scatter/gather

Scatter/gather is supported through the use of linked lists. This means that the source and destination areas do not have to occupy contiguous areas in memory. Where scatter/gather is not required, the DMACCxLLI Register must be set to 0.

The source and destination data areas are defined by a series of linked lists. Each Linked List Item (LLI) controls the transfer of one block of data, and then optionally loads another LLI to continue the DMA operation, or stops the DMA stream. The first LLI is programmed into the DMA Controller.

The data to be transferred described by an LLI (referred to as the packet of data) usually requires one or more DMA bursts (to each of the source and destination).

#### 33.6.5.1 Linked list items

A Linked List Item (LLI) consists of four words. These words are organized in the following order:

1. DMACCxSrcAddr.
2. DMACCxDestAddr.
3. DMACCxLLI.
4. DMACCxControl.

**Note:** The DMACCxConfig DMA channel Configuration Register is not part of the linked list item.

#### 33.6.5.1.1 Programming the DMA controller for scatter/gather DMA

To program the DMA Controller for scatter/gather DMA:

1. Write the LLIs for the complete DMA transfer to memory. Each linked list item contains four words:
  - Source address.
  - Destination address.
  - Pointer to next LLI.
  - Control word.

The last LLI has its linked list word pointer set to 0.

2. Choose a free DMA channel with the priority required. DMA channel 0 has the highest priority and DMA channel 7 the lowest priority.
3. Write the first linked list item, previously written to memory, to the relevant channel in the DMA Controller.
4. Write the channel configuration information to the channel Configuration Register and set the Channel Enable bit. The DMA Controller then transfers the first and then subsequent packets of data as each linked list item is loaded.
5. An interrupt can be generated at the end of each LLI depending on the Terminal Count bit in the DMACCxControl Register. If this bit is set an interrupt is generated at the end of the relevant LLI. The interrupt request must then be serviced and the relevant bit in the DMACIntTCClear Register must be set to clear the interrupt.

33.6.5.1.2 Example of scatter/gather DMA

See Figure 160 for an example of an LLI. A section of memory is to be transferred to a peripheral. The addresses of each LLI entry are given, in hexadecimal, at the left-hand side of the figure. In this example, the LLIs describing the transfer are to be stored contiguously from address 0x2002 0000, but they could be located anywhere. The right side of the figure shows the memory containing the data to be transferred.

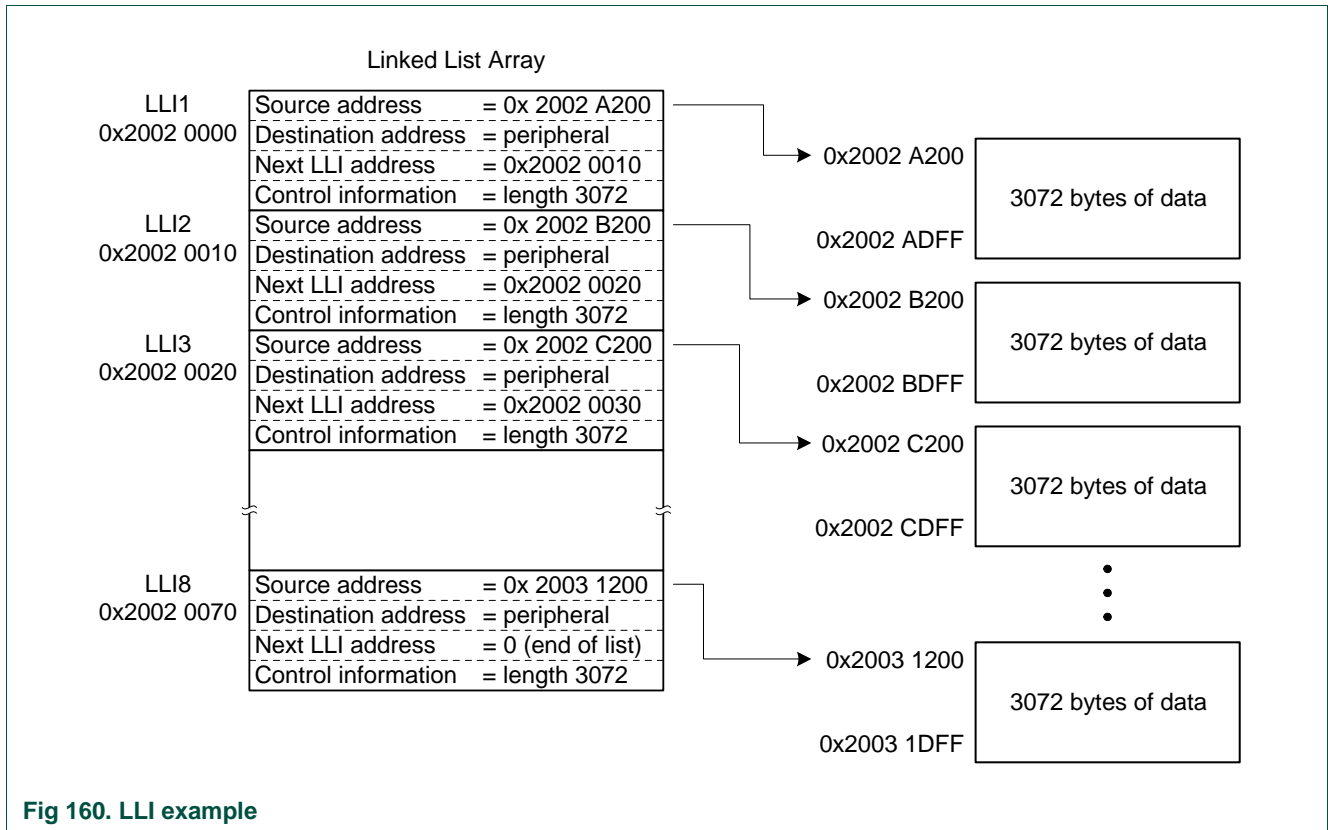


Fig 160. LLI example

The first LLI, stored at 0x2002 0000, defines the first block of data to be transferred, which is the data stored from address 0x2002 A200 to 0x2002 ADFF:

- Source start address 0x2002 A200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0XC00).
- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x2002 0010.

The second LLI, stored at 0x2002 0010, describes the next block of data to be transferred:

- Source start address 0x2002 B200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0xC00).

- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x2002 0020.

A chain of descriptors is built up, each one pointing to the next in the series. To initialize the DMA stream, the first LLI, 0x2002 0000, is programmed into the DMA Controller. When the first packet of data has been transferred the next LLI is automatically loaded.

The final LLI is stored at 0x2002 0070 and contains:

- Source start address 0x2003 1200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0xC00).
- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x0.

Because the next LLI address is set to zero, this is the last descriptor, and the DMA channel is disabled after transferring the last item of data. The channel is probably set to generate an interrupt at this point to indicate to the ARM processor that the channel can be reprogrammed.



### 34.1 Introduction

---

The Cyclic Redundancy Check (CRC) generator with programmable polynomial settings supports several CRC standards commonly used. To save system power and bus bandwidth, the CRC engine supports DMA transfers in addition to software PIO operations using the CPU.

### 34.2 Features

---

- Supports three common polynomials CRC-CCITT, CRC-16, and CRC-32.
  - CRC-CCITT:  $x^{16} + x^{12} + x^5 + 1$
  - CRC-16:  $x^{16} + x^{15} + x^2 + 1$
  - CRC-32:  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Bit order reverse and 1's complement programmable setting for input data and CRC sum.
- Programmable seed number setting.
- Supports CPU PIO or DMA back-to-back transfer.
- Accept any size of data width per write: 8, 16 or 32-bit.
  - 8-bit write: 1-cycle operation
  - 16-bit write: 2-cycle operation (8-bit x 2-cycle)
  - 32-bit write: 4-cycle operation (8-bit x 4-cycle)

### 34.3 Description

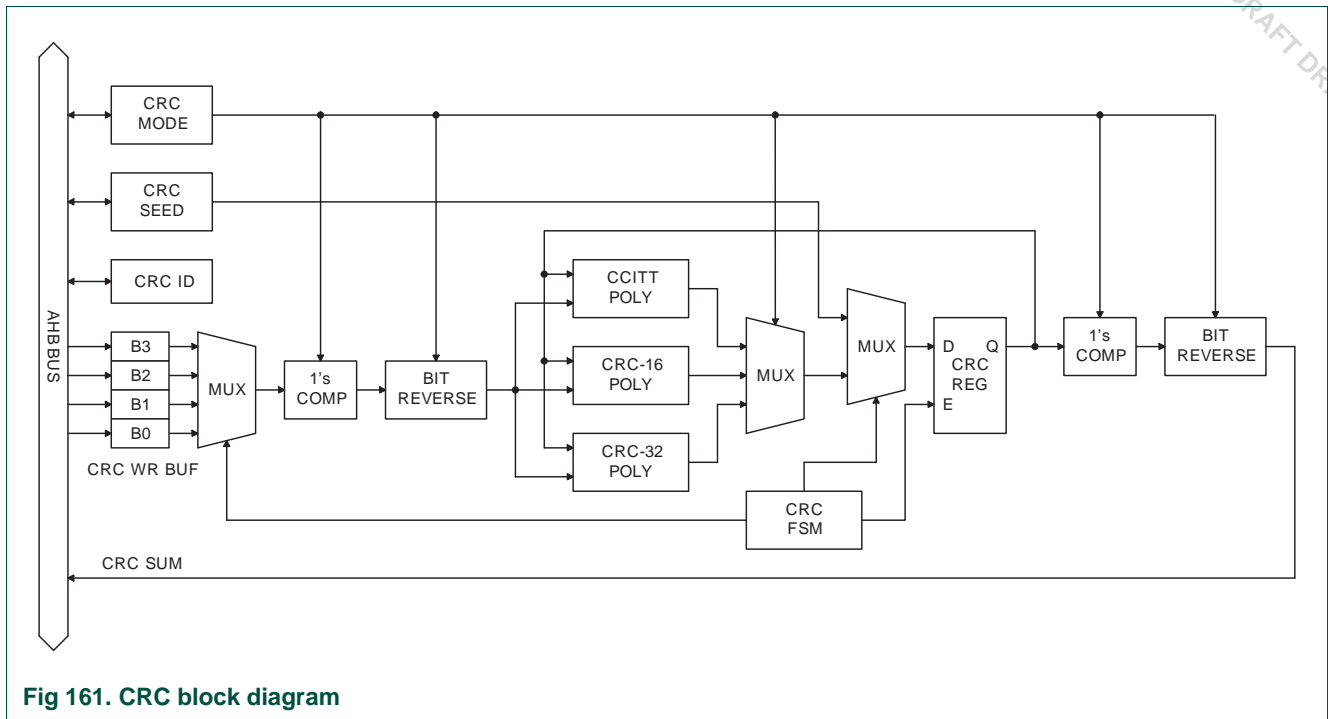


Fig 161. CRC block diagram

### 34.4 Register description

**Table 679. Register overview: CRC engine**

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
CRC_MODE	CRC mode register	R/W	0	0x2009 0000	<a href="#">680</a>
CRC_SEED	CRC seed register	R/W	0xFFFF	0x2009 0004	<a href="#">681</a>
CRC_SUM	CRC checksum register	RO	0xFFFF	0x2009 0008	<a href="#">682</a>
CRC_DATA	CRC data register	WO	-	0x2009 0008	<a href="#">683</a>

[1] Reset Value reflects the data stored in used bits only. It does not include content of reserved bits.

#### 34.4.1 CRC mode register (CRC\_MODE - 0x2009 0000)

**Table 680. CRC mode register (CRC\_MODE - 0x2009 0000) bit description**

Bit	Symbol	Description	Reset value
1:0	CRC_POLY	1X: CRC-32 polynomial 01: CRC-16 polynomial 00: CRC-CCITT polynomial	0
2	BIT_RVS_WR	1: Bit order reverse for CRC_WR_DATA (per byte) 0: No bit order reverse for CRC_WR_DATA (per byte)	0
3	CMPL_WR	1: 1's complement for CRC_WR_DATA 0: No 1's complement for CRC_WR_DATA	0
4	BIT_RVS_SUM	1: Bit order reverse for CRC_SUM 0: No bit order reverse for CRC_SUM	0
5	CMPL_SUM	1: 1's complement for CRC_SUM 0: No 1's complement for CRC_SUM	0
31:6	Reserved	Always '0' when read	0

#### 34.4.2 CRC seed register (CRC\_SEED - 0x2009 0004)

**Table 681. CRC seed register (CRC\_SEED - 0x2009 0004) bit description**

Bit	Symbol	Description	Reset value
31:0	CRC_SEED	A write access to this register will load CRC seed value to CRC_SUM register with selected bit order and 1's complement pre-processes. <b>Remark:</b> A write access to this register will overrule the CRC calculation in progresses.	0xFFFF

#### 34.4.3 CRC checksum register (CRC\_SUM - 0x2009 0008)

This register is a Read-only register containing the most recent checksum.

**Table 682. CRC checksum register (CRC\_SUM - 0x2009 0008) bit description**

Bit	Symbol	Description	Reset value
31:0	CRC_SUM	The most recent CRC sum can be read through this register with selected bit order and 1's complement post-processes.	0xFFFF

#### 34.4.4 CRC data register (CRC\_DATA - 0x2009 0008)

This register is a Write-only register containing the data block for which the CRC sum will be calculated.

**Table 683. CRC data register (CRC\_DATA - 0x2009 0008) bit description**

Bit	Symbol	Description	Reset value
31:0	CRC_WR_DATA	Data written to this register will be taken to perform CRC calculation with selected bit order and 1's complement pre-process. Any write size 8, 16 or 32-bit are allowed and accept back-to-back transactions.	-

## 34.5 Functional description

The following sections describe the register settings for each supported CRC standard:

### CRC-CCITT set-up

Polynomial =  $x^{16} + x^{12} + x^5 + 1$

Seed Value = 0xFFFF

Bit order reverse for data input: NO

1's complement for data input: NO

Bit order reverse for CRC sum: NO

1's complement for CRC sum: NO

CRC\_MODE = 0x0000 0000

CRC\_SEED = 0x0000 FFFF

### CRC-16 set-up

Polynomial =  $x^{16} + x^{15} + x^2 + 1$

Seed Value = 0x0000

Bit order reverse for data input: YES

1's complement for data input: NO

Bit order reverse for CRC sum: YES

1's complement for CRC sum: NO

CRC\_MODE = 0x0000 0015

CRC\_SEED = 0x0000 0000

### CRC-32 set-up

Polynomial =  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Seed Value = 0xFFFF FFFF

Bit order reverse for data input: YES

1's complement for data input: NO

Bit order reverse for CRC sum: YES

1's complement for CRC sum: YES

CRC\_MODE = 0x0000 0036

CRC\_SEED = 0xFFFF FFFF

### 35.1 Basic configuration

---

The EEPROM is configured using the following registers:

1. Power: The EEPROM is enabled on reset, but may be turned off if it is not needed. See [Section 35.4.1.7](#).
2. Clocking: Timing for the EEPROM must be set up before it can be used. See [Section 35.4.1.5](#) and [Section 35.4.1.6](#).
3. Interrupts: Interrupts are controlled using a set of registers, see [Section 35.4.2](#). Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register, see [Table 41](#).

### 35.2 Description

---

EEPROM is a non-volatile memory mostly used for storing relatively small amounts of data, for example for storing settings. The EEPROM is indirectly accessed through address and data registers, so the CPU can not execute code from EEPROM memory.

### 35.3 Features

---

- 4kB EEPROM
- Access via address and data registers on the AHB bus
- Less than 3 ms erase / program time
- Endurance of > 100k erase / program cycles

### 35.4 Register description

[Table 684](#) shows the registers related to EEPROM operation. Details of each register follow.

**Table 684. Register overview**

Name	Description	Access	Reset value <sup>[1]</sup>	Address	Table
<b>EEPROM registers</b>					
EECMD	EEPROM command register	R/W	0	0x0020 0080	<a href="#">685</a>
EEADDR	EEPROM address register	R/W	0	0x0020 0084	<a href="#">686</a>
EEWDATA	EEPROM write data register	WO	NA	0x0020 0088	<a href="#">687</a>
EERDATA	EEPROM read data register	RO	NA	0x0020 008C	<a href="#">688</a>
EEWSTATE	EEPROM wait state register	R/W	0	0x0020 0090	<a href="#">689</a>
EECLKDIV	EEPROM clock divider register	R/W	0	0x0020 0094	<a href="#">690</a>
EEPWRDWN	EEPROM power-down register	R/W	0	0x0020 0098	<a href="#">691</a>
<b>EEPROM interrupt registers:</b>					
EEINTEN	EEPROM interrupt enable	RO	0	0x0020 0FE4	<a href="#">692</a>
EEINTENCLR	EEPROM interrupt enable clear	WO	0	0x0020 0FD8	<a href="#">693</a>
EEINTENSET	EEPROM interrupt enable set	WO	0	0x0020 0FDC	<a href="#">694</a>
EEINTSTAT	EEPROM interrupt status	RO	0	0x0020 0FE0	<a href="#">695</a>
EEINTSTATCLR	EEPROM interrupt status clear	WO	0	0x0020 0FE8	<a href="#">696</a>
EEINTSTATSET	EEPROM interrupt status set	WO	0	0x0020 0FEC	<a href="#">697</a>

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

### 35.4.1 EEPROM control registers

#### 35.4.1.1 EEPROM command register (EECMD - 0x0020 0080)

The EEPROM command register is used to select and start a read, write or erase/program operation. Read and erase/program operations are started on the EEPROM device as a side-effect of writing to this register. (Write operations are started as a side-effect of writing to the write data register).

**Table 685. EEPROM command register (EECMD - address 0x0020 0080) bit description**

Bits	Symbol	Description	Reset value
2:0	CMD	Command 000: 8-bit read 001: 16-bit read 010: 32-bit read 011: 8-bit write 100: 16-bit write 101: 32-bit write 110: erase/program page 111: reserved	0
3	RDPREFETCH	Read data prefetch bit 0: do not prefetch next read data as result of reading from the read data register 1: prefetch read data as result of reading from the read data register When this bit is set multiple consecutive data elements can be read without the need of programming new address values in the address register. The address post-increment and the automatic read data prefetch (if enabled) allow only reading from the read data register to be done to read the data.	0
31:4	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 35.4.1.2 EEPROM address register (EEADDR - 0x0020 0084)

The EEPROM address register is used to program the address for read, write or erase/program operations. The width of the address field depends on the number of EEPROM devices which is selectable by a configuration parameter.

**Table 686. EEPROM address register (EEADDR - address 0x0020 0084) address description**

Bits	Symbol	Description	Reset value
11:0	ADDR	Address Address field with: 1 EEPROM device => x = 13 (8 MSB, 0 CS and 6 LSB bits)	0
31:12	-	Reserved. Read value is undefined, only zero should be written.	NA

#### 35.4.1.3 EEPROM write data register (EEWDATA - 0x0020 0088)

The EEPROM write data register is used to write data into the page register (write operations).



Writing this register will start the write operation as side-effect. The address is post-incremented, so consecutive writes to this register can be done to write a burst of data. The address will be incremented automatically according to the data size of the write operation.

If data is written to this register while a previous operation (read, write or an erase/program on the same device) is still pending the write command on the AHB is stalled by de-asserting the ready signal until the previous operation is finished. To avoid stalling of the system bus the interrupt status register can be used for polling the status of pending operations.

**Table 687. EEPROM write data register (EEWDATA - address 0x0020 0088) bit description**

Bits	Symbol	Description	Reset value
31:0	WDATA	Write data In case of: 8-bit write operations: bits [7:0] must contain valid write data. 16-bit write operations: bits [15:0] must contain valid write data 32-bit write operations: bits [31:0] must contain valid write data	-

#### 35.4.1.4 EEPROM read data register (EERDATA - 0x0020 008C)

The EEPROM read data register is used to read data from memory.

Reading this register will start the next read operation, and the address will be post-incremented. Consecutive reads from this register can be done to read a burst of data. The address will be incremented automatically according to the data size of the read operation.

If data is read from this register while the read operation is still pending the read command on the AHB bus is stalled by de-asserting the ready signal until the pending operation is finished. To avoid stalling of the system bus the interrupt status register can be used for polling the status of pending operations.

**Table 688. EEPROM read data register (EERDATA - address 0x0020 008C) bit description**

Bits	Symbol	Description	Reset value
31:0	RDATA	Read data In case of: 8-bit read operations: bits [7:0] contain read data, others are zero 16-bit read operations: bits [15:0] contain read data, others are zero 32-bit read operations: bits [31:0] contain read data	-

#### 35.4.1.5 EEPROM wait state register (EEWSTATE - 0x0020 0090)

The EEPROM controller uses the times specified by this register to perform various internal timing functions. The user must program the wait state fields to appropriate values in this wait state register. These fields are -1 encoded so programming zero will result in a one cycle wait state.

**Table 689. EEPROM wait state register (EEESTATE - address 0x0020 0090) bit description**

Bits	Symbol	Description	Reset value
7:0	PHASE3	Wait states 3 (minus 1 encoded) The number of system clock periods required to give a minimum time of 15 ns.	0
15:8	PHASE2	Wait states 2 (minus 1 encoded) The number of system clock periods required to give a minimum time of 55 ns.	0
23:16	PHASE1	Wait states 1 (minus 1 encoded) The number of system clock periods required to give a minimum time of 35 ns.	0
31:24	-	Reserved. Read value is undefined, only zero should be written.	NA

### 35.4.1.6 EEPROM clock divider register (EECLKDIV - 0x0020 0094)

The EEPROM device(s) require(s) a 375kHz clock. This clock is generated by dividing the system bus clock. The clock divider register contains the division factor.

If the division factor is 0 the clock will be IDLE to save power.

$$\frac{cclk}{CLKDIV + 1} \approx 375kHz \pm 6.67\%$$

For example, if the CPU clock is 80 MHz, CLKDIV could be set to 212 (decimal). 80 MHz / 213 (CLKDIV + 1) = 375.6 kHz.

**Table 690. EEPROM clock divider register (EECLKDIV - address 0x0020 0094) bit description**

Bits	Symbol	Description	Reset value
15:0	CLKDIV	Division factor (minus 1 encoded)	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

### 35.4.1.7 EEPROM power down register (EEPWRDWN - 0x0020 0098)

The EEPROM power down register can be used to put the EEPROM device(s) in power down mode.

The EEPROM may not be put in power-down mode during a pending EEPROM operation. After clearing this bit any EEPROM operation has to be suspended for 100 μs while the EEPROM wakes up.

**Table 691. EEPROM power down/DCM register (EEPWRDWN - address 0x0020 0098) bit description**

Bits	Symbol	Description	Reset value
0	PWRDWN	Power down mode bit 0: not in power down mode 1: power down mode (this will put all EEPROM devices in power down)	0
31:1	-	Reserved. Read value is undefined, only zero should be written.	NA

### 35.4.2 Interrupt registers

These registers control interrupts from the EEPROM.

#### 35.4.2.1 Interrupt enable register (EEINTEN - 0x0020 0FE4)

Table 692. Interrupt enable register (EEINTEN - address 0x0020 0FE4) bit description

Bits	Symbol	Description	Reset value
25:0	-	Reserved. The value read from a reserved bit is not defined.	NA
26	EE_RW_DONE	EEPROM read/write operation finished interrupt enable bit Bit is: - set when '1' is written to the corresponding bit of the EEINTENSET register. - cleared when '1' is written to the corresponding bit of the EEINTENCLR register.	0
27	-	Reserved. The value read from a reserved bit is not defined.	NA
28	EE_PROG_DONE	EEPROM program operation finished interrupt enable bit Bit is: - set when '1' is written in the corresponding bit of the EEINTENSET register - cleared when '1' is written to the corresponding bit of the EEINTENCLR register.	0
31:29	-	Reserved. The value read from a reserved bit is not defined.	NA

The interrupt request output is asserted when the bitwise AND of EEINTSTAT and EEINTEN is nonzero. For the EEPROM read/write operation finished interrupt it is better not to enable the interrupt, but to only poll the bit in the EEINTSTAT register. This is because these operations are relatively fast operations that do not justify calling a interrupt service subroutine in software.

#### 35.4.2.2 Interrupt enable clear register (EEINTENCLR, 0x0020 0FD8)

Table 693. Interrupt enable clear register (EEINTENCLR - address 0x0020 0FD8) bit description

Bits	Symbol	Description	Reset value
25:0	-	Reserved. Read value is undefined, only zero should be written.	NA
26	RDWR_CLR_EN	Clear read/write operation finished interrupt enable bit (EEPROM) 0: leave corresponding bit unchanged. 1: clear corresponding bit.	0
27	-	Reserved. Read value is undefined, only zero should be written.	NA
28	PROG1_CLR_EN	Clear program operation finished interrupt enable bit for EEPROM device 1 0: leave corresponding bit unchanged. 1: clear corresponding bit.	0
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA

35.4.2.3 Interrupt enable set register (EEINTENSET - 0x0020 0FDC)

Table 694. Interrupt enable set register (EEINTENSET - address 0x0020 0FDC) bit description

Bits	Symbol	Description	Reset value
25:0	-	Reserved. Read value is undefined, only zero should be written.	NA
26	RDWR_SET_EN	Set read/write operation finished interrupt enable bit (EEPROM) 0: leave corresponding bit unchanged. 1: set corresponding bit.	0
27	-	Reserved. Read value is undefined, only zero should be written.	NA
28	PROG1_SET_EN	Set program operation finished interrupt enable bit for EEPROM device 1 0: leave corresponding bit unchanged. 1: set corresponding bit.	0
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA

35.4.2.4 Interrupt status register (EEINTSTAT - 0x0020 0FE0)

Table 695. Interrupt status register (EEINTSTAT - address 0x0020 0FE0) bit description

Bits	Symbol	Description	Reset value
25:0	-	Reserved. The value read from a reserved bit is not defined.	NA
26	END_OF_RDWR	EEPROM read/write operation finished interrupt status bit Bit is: - set when this operation has finished OR when '1' is written in the corresponding bit of the EEINTSTATSET register. - cleared when '1' is written to the corresponding bit of the EEINTSTATCLR register.	0
27	-	Reserved. The value read from a reserved bit is not defined.	NA
28	END_OF_PROG1	EEPROM program operation finished interrupt status bit Bit is: - set when this operation has finished OR when '1' is written to the corresponding bit of the EEINTSTATSET register. - cleared when '1' is written to the corresponding bit of the EEINTSTATCLR register.	0
31:29	-	Reserved. The value read from a reserved bit is not defined.	NA

The interrupt request output is asserted when the bitwise AND of EEINTSTAT and EEINTEN is nonzero. For the EEPROM read/write operation finished interrupt it is better not to enable the interrupt, but to only poll the bit in the EEINTSTAT register. This is because these operations are relatively fast operations that do not justify calling a interrupt service subroutine in software.

35.4.2.5 Interrupt status clear register (0x0020 0FE8)

Table 696. Interrupt status clear register (EEINTSTATCLR - address 0x0020 0FE8) bit description

Bits	Symbol	Description	Reset value
25:0	-	Reserved. Read value is undefined, only zero should be written.	NA
26	RDWR_CLR_ST	Clear read/write operation finished interrupt status bit (EEPROM) 0 leave corresponding bit unchanged. 1 clear corresponding bit.	0
27	-	Reserved. Read value is undefined, only zero should be written.	NA
28	PROG1_CLR_ST	Clear program operation finished interrupt status bit for EEPROM device 1 0 leave corresponding bit unchanged. 1 clear corresponding bit.	0
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA

35.4.2.6 Interrupt status set (EEINTSTATSET - 0x0020 0FEC)

Table 697. Interrupt status set register (EEINTSTATSET - address 0x0020 0FEC)

Bits	Symbol	Description	Reset value
25:0	-	Reserved. Read value is undefined, only zero should be written.	NA
26	RDWR_SET_ST	Set read/write operation finished interrupt status bit (EEPROM). 0 leave corresponding bit unchanged. 1 set corresponding bit.	0
27	-	Reserved. Read value is undefined, only zero should be written.	NA
28	PROG1_SET_ST	Set program operation finished interrupt status bit for EEPROM device 1. 0 leave corresponding bit unchanged. 1 set corresponding bit.	0
31:29	-	Reserved. Read value is undefined, only zero should be written.	NA

## 35.5 EEPROM operation

### 35.5.1 EEPROM device description

EEPROM is a non-volatile memory mostly used for storing relatively small amounts of data, for example for storing settings.

There are three operations for accessing the memory: reading, writing and erase/program. "Writing" to memory is split up into two separate operations, writing and erase/program. The first operation which will be called "writing" in this document is not really updating the memory, but only updating the temporary data register called the "page register". The page register needs to be written with minimum 1 byte and maximum 8 bytes before the second operation which is called "erase/program" in this document can be used to actually update the non-volatile memory. Note that the data written to the page register is not "cached", it can't be read before it is actually programmed into non-volatile memory.

The 64-byte page register is the same size as a page in EEPROM memory. The 4kB EEPROM contains 64 pages.

### 35.5.2 EEPROM operations

An EEPROM device can not be programmed directly. Writing data to it and the actual erase/program of the memory are two separate steps. The page register (64 bytes) will temporarily hold write data. But as soon as this data needs to be read from the EEPROM or data needs to be written to another page, the contents of the page register first needs to be programmed into the EEPROM memory.

The following sections explain the EEPROM operations (read, write and program) in more detail.

#### 35.5.2.1 Writing

The EEPROM controller supports writing of 8-bit, 16-bit or 32-bit elements. Since the EEPROM device doesn't support 32-bit operations the controller splits the operation into two 16-bit operations.

For doing a write operation first an address needs to be written into the address register and the kind of write operation needs to be selected in the command register. This can be done in any order. After this the data is written to the write data register, which automatically starts the write operation on the EEPROM device.

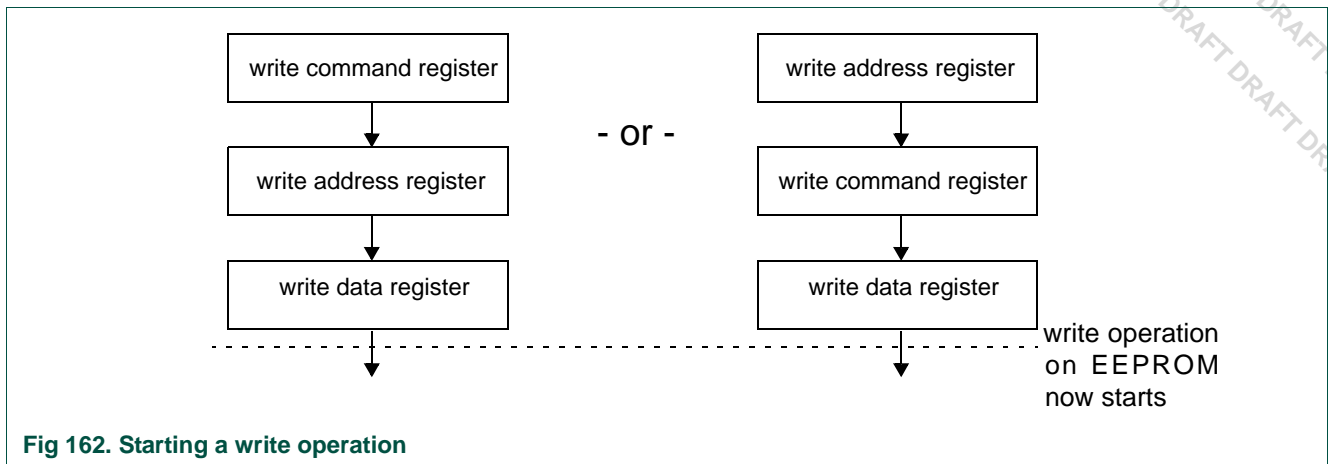


Fig 162. Starting a write operation

A write operation causes an automatic post-increment of the address. This allows consecutive writes to the page register without the need of writing a new address for every write operation. Of course the address register could be written with another address value to write to another location.

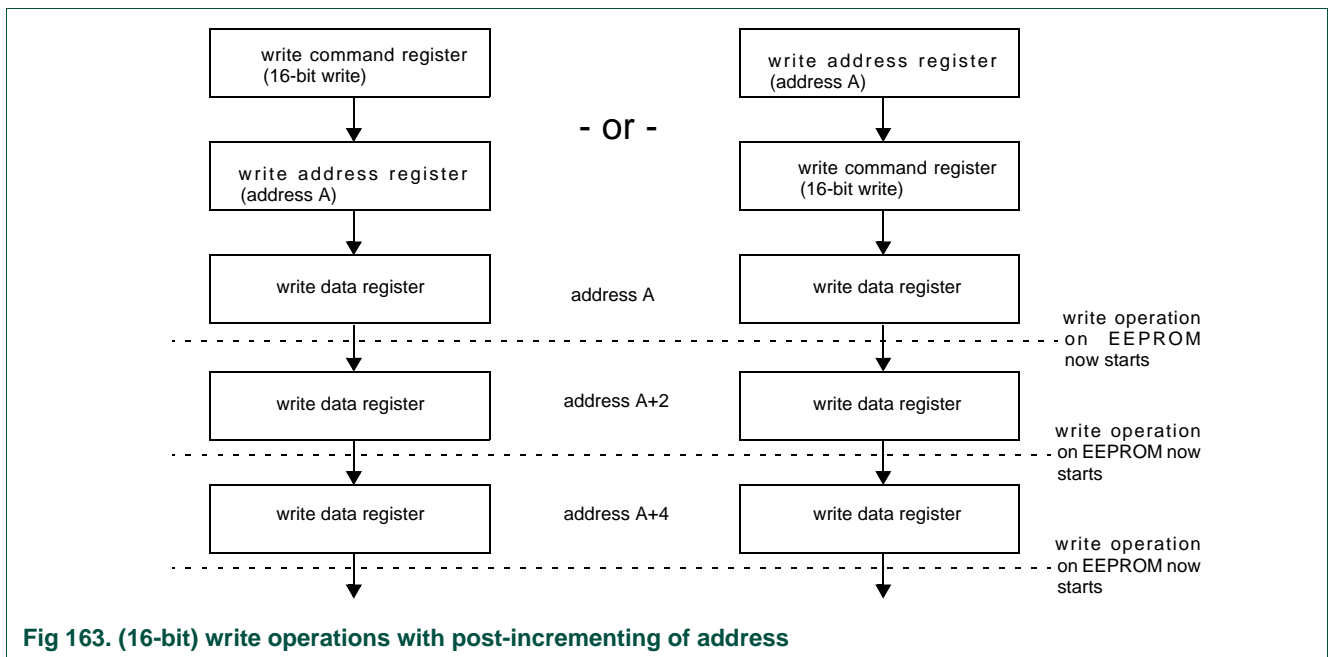


Fig 163. (16-bit) write operations with post-incrementing of address

If the data register is written while a previous EEPROM operation is still pending, the write transfer on the system bus is stalled by de-asserting the ready signal until the previous operation is finished. This can be avoided by polling the interrupt status register to see if an operation is still pending before starting the write operation. Polling is generally only helpful for systems running at a high frequency (>200MHz).

Software has to make sure that the following rules are followed:

- overwriting (writing it two times before a erase/program operation) one of the locations in a 64-byte page register is not allowed, it will cause the loss of the previously written data (this is a consequence of an EEPROM module property, see EEPROM spec)

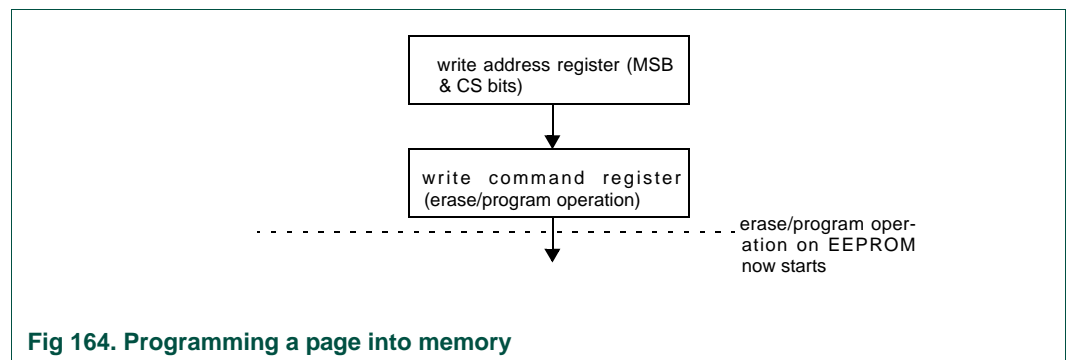
- in case the default address post-incrementing is used the upper boundary of the page register may not be crossed
- before reading the just written data the contents of the page register needs to be programmed into non-volatile memory
- write operations to a misaligned address will result on an error response on the write transfer to the write data register (for example a 32-bit write operation to an address other than a multiple of 0x4). The operation will not be performed.

### 35.5.2.2 Programming

When the page register has been written the data has to be programmed into non-volatile memory. This is a separate step, writing only the page register will not write the EEPROM memory.

Programming the page into memory takes a long time, therefore the corresponding interrupt can be enabled or the interrupt status bit can be polled to avoid stalling of the system bus.

An erase/program operation starts by providing the MSBs of the address that selects the page in memory of a device and the CS bits that selects a device. The 6 LSBs are “don't care”. The operation is started by writing the command register (selecting the erase/program operation). Before beginning a programming operation, the EEPROM status should be polled to insure that the last write operation is complete.

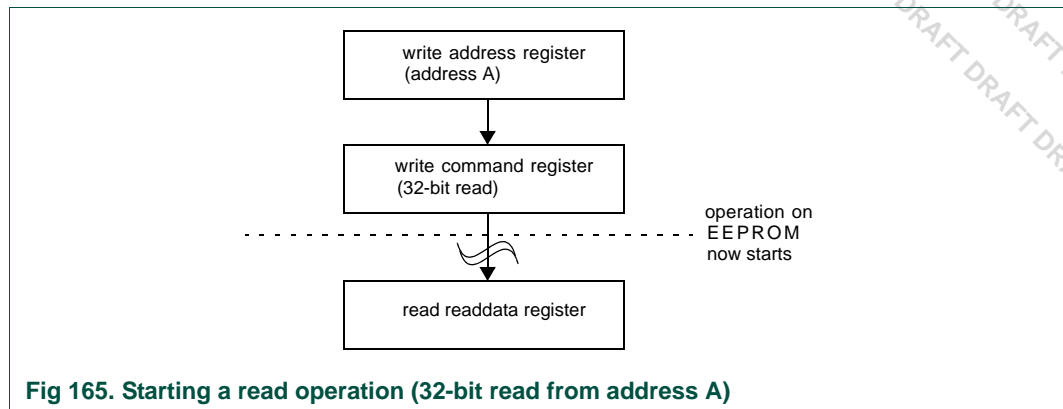


### 35.5.2.3 Reading

The EEPROM controller supports reading of 8-bit, 16-bit or 32-bit elements. Since the EEPROM device doesn't support 32-bit operations the controller splits the operation into two 16-bit operations.

For doing a read operation first an address needs to be written into the address register. Then the operation needs to be selected in the command register. Writing the command register will automatically start the read operation on the EEPROM device.





**Fig 165. Starting a read operation (32-bit read from address A)**

If the read data register is read while the read operation is still pending, then the read transfer on the system bus is stalled by de-asserting the ready signal until the previous read operation is finished. This can be avoided by polling the interrupt status register to see if the operation is still pending before reading the read data register.

Read operations will automatically post-increment the address register. This allows consecutive reads from the EEPROM memory without the need of writing a new address for every read operation. By setting the read data prefetch bit in the command register reading from the read data register automatically starts up a read operation from the next (incremented) address location. When doing consecutive reads in this way the first read operation is started as result of writing the command register. The following read operations are started as result of reading the read data register to obtain the result of the previous read operations.

Read operations from a misaligned address will result on an error response on the write transfer to the command register (for example a 32-bit read operation from an address other than a multiple of 0x4). The operation will not be performed.

#### 35.5.2.4 Error responses

The controller can generate the following EEPROM related error responses in the following situations:

- An erase/program operation on the protected page will result in an error response on the write transfer to the command register. Unless the protected page's protection is overruled by setting the OVP input signal.
- Writing a read-only register or reading a write-only register will result in an error response.
- A transfer to a non-existing register location will result in an error.

### 36.1 Introduction

---

The boot loader controls initial operation after reset and also provides the tools for programming the flash memory. This could be initial programming of a blank device, erasure and re-programming of a previously programmed device, or programming of the flash memory by the application program in a running system.

### 36.2 Features

---

- In-System Programming: In-System programming (ISP) is programming or reprogramming the on-chip flash memory, using the boot loader software and UART0 serial port. This can be done when the part resides in the end-user board.
- In Application Programming: In-Application (IAP) programming is performing erase and write operation on the on-chip flash memory, as directed by the end-user application code.
- Flash signature generation: built-in hardware can generate a signature for a range of flash addresses, or for the entire flash memory.

### 36.3 Description

---

The flash boot loader code is executed every time the part is powered on or reset. The loader can execute the ISP command handler or the user application code. A LOW level after reset at pin P2[10] is considered an external hardware request to start the ISP command handler using UART0 pins P0[2] (U0\_TXD) and P0[3] (U0\_RXD). Assuming that power supply pins are on their nominal levels when the rising edge on  $\overline{\text{RESET}}$  pin is generated, it may take up to 3 ms before P2[10] is sampled and the decision on whether to continue with user code or ISP handler is made. If P2[10] is sampled low and the watchdog overflow flag is set, the external hardware request to start the ISP command handler is ignored. If there is no request for the ISP command handler execution (P2[10] is sampled HIGH after reset), a search is made for a valid user program. If a valid user program is found then the execution control is transferred to it. If a valid user program is not found, the auto-baud routine is invoked.

Pin P2[10] is used as a hardware request signal for ISP and therefore requires special attention. Since P2[10] is in high impedance mode after reset, it is important that the user provides external hardware (a pull-up resistor or other device) to put the pin in a defined state. Otherwise unintended entry into ISP mode may occur.

When ISP mode is entered after a power on reset, the IRC frequency of 12 MHz is used to operate the CPU and peripherals. The baud rates that can easily be obtained in this case are: 9600 baud, 19200 baud, 38400 baud, 57600 baud, 115200 baud, and 230400 baud. This may not be the case when ISP is invoked by the user application (see [Section 36.8.9 "Re-invoke ISP" on page 807](#)).

A hardware flash signature generation capability is built into the flash memory. This feature can be used to create a signature that can then be used to verify flash contents. Details of flash signature generation are in [Section 36.10](#).

### 36.3.1 Memory map after any reset

When a user program begins execution after reset, the interrupt vectors are set to point to the beginning of flash memory.

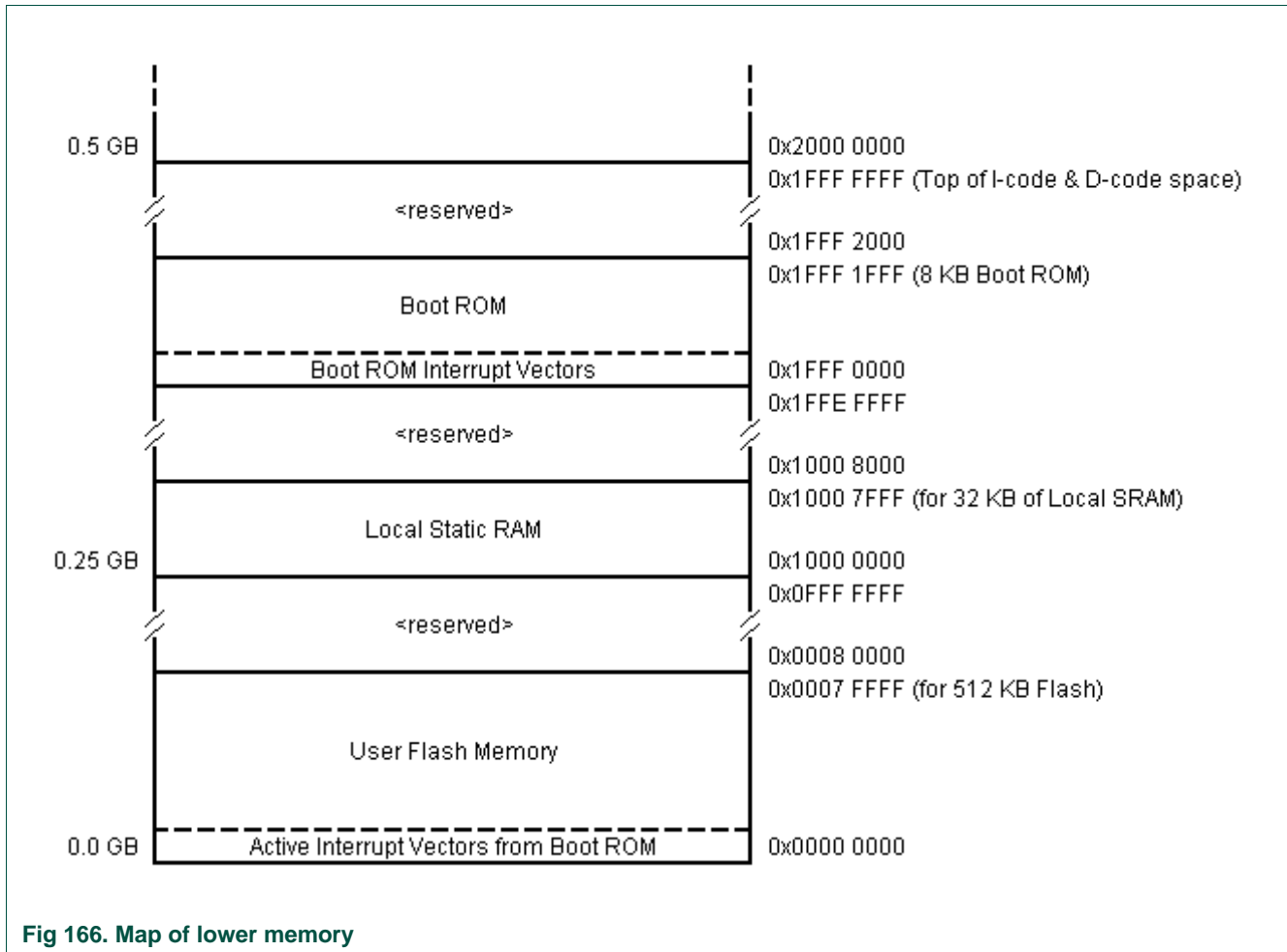


Fig 166. Map of lower memory

#### 36.3.1.1 Criterion for Valid User Code

The reserved Cortex-M3 exception vector location 7 (offset 0x001C in the vector table) should contain the 2's complement of the check-sum of table entries 0 through 6. This causes the checksum of the first 8 table entries to be 0. The boot loader code checksums the first 8 locations in sector 0 of the flash. If the result is 0, then execution control is transferred to the user code.

If the signature is not valid, the auto-baud routine synchronizes with the host via serial port 0. The host should send a "?" (0x3F) as a synchronization character and wait for a response. The host side serial port settings should be 8 data bits, 1 stop bit and no parity. The auto-baud routine measures the bit time of the received synchronization character in terms of its own frequency and programs the baud rate generator of the serial port. It also

sends an ASCII string ("Synchronized<CR><LF>") to the host. In response to this the host should send the same string ("Synchronized<CR><LF>"). The auto-baud routine looks at the received characters to verify synchronization. If synchronization is verified then "OK<CR><LF>" string is sent to the host. The host should respond by sending the crystal frequency (in kHz) at which the part is running. For example, if the part is running at 10 MHz, the response from the host should be "10000<CR><LF>". "OK<CR><LF>" string is sent to the host after receiving the crystal frequency. If synchronization is not verified then the auto-baud routine waits again for a synchronization character. For auto-baud to work correctly in case of user invoked ISP, the CCLK frequency should be greater than or equal to 10 MHz.

For more details on Reset, PLL and startup/boot code interaction see [Section 4.5.1 "PLL and startup/boot code interaction"](#).

Once the crystal frequency is received the part is initialized and the ISP command handler is invoked. For safety reasons an "Unlock" command is required before executing the commands resulting in flash erase/write operations and the "Go" command. The rest of the commands can be executed without the unlock command. The Unlock command is required to be executed once per ISP session. The Unlock command is explained in [Section 36.7 "ISP commands" on page 794](#).

### 36.3.2 Communication protocol

All ISP commands should be sent as single ASCII strings. Strings should be terminated with Carriage Return (CR) and/or Line Feed (LF) control characters. Extra <CR> and <LF> characters are ignored. All ISP responses are sent as <CR><LF> terminated ASCII strings. Data is sent and received in UU-encoded format.

#### 36.3.2.1 ISP command format

"Command Parameter\_0 Parameter\_1 ... Parameter\_n<CR><LF>" "Data" (Data only for Write commands).

#### 36.3.2.2 ISP response format

"Return\_Code<CR><LF>Response\_0<CR><LF>Response\_1<CR><LF> ... Response\_n<CR><LF>" "Data" (Data only for Read commands).

#### 36.3.2.3 ISP data format

The data stream is in UU-encoded format. The UU-encode algorithm converts 3 bytes of binary data in to 4 bytes of printable ASCII character set. It is more efficient than Hex format which converts 1 byte of binary data in to 2 bytes of ASCII hex. The sender should send the check-sum after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. The receiver should compare it with the check-sum of the received bytes. If the check-sum matches then the receiver should respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match the receiver should respond with "RESEND<CR><LF>". In response the sender should retransmit the bytes.

#### 36.3.2.4 ISP flow control

A software XON/XOFF flow control scheme is used to prevent data loss due to buffer overrun. When the data arrives rapidly, the ASCII control character DC3 (0x13) is sent to stop the flow of data. Data flow is resumed by sending the ASCII control character DC1 (0x11). The host should also support the same flow control scheme.

#### 36.3.2.5 ISP command abort

Commands can be aborted by sending the ASCII control character "ESC" (0x1B). This feature is not documented as a command under "ISP Commands" section. Once the escape code is received the ISP command handler waits for a new command.

#### 36.3.2.6 Interrupts during IAP

The on-chip flash memory is not accessible during erase/write operations. When the user application code starts executing the interrupt vectors from the user flash area are active. The user should either disable interrupts, or ensure that user interrupt vectors are active in RAM and that the interrupt handlers reside in RAM, before making a flash erase/write IAP call. The IAP code does not use or disable interrupts.

#### 36.3.2.7 RAM used by ISP command handler

ISP commands use on-chip RAM from 0x1000 0118 to 0x1000 01FF. The user could use this area, but the contents may be lost upon reset. Flash programming commands use the top 32 bytes of on-chip RAM. The stack is located at RAM top - 32. The maximum stack usage is 256 bytes, growing downwards.

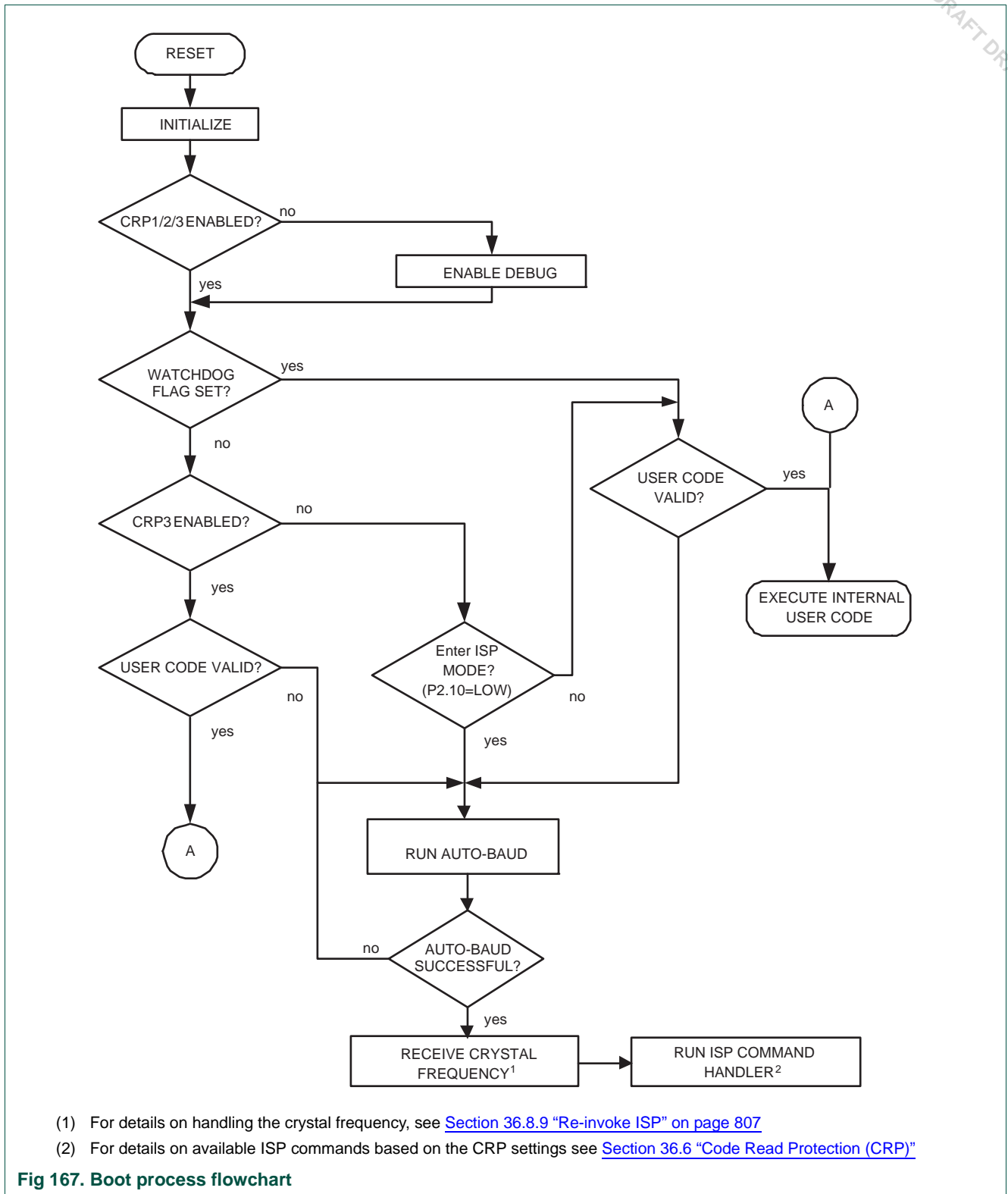
#### 36.3.2.8 RAM used by Boot process prior to entering user program

Following chip reset, the Boot program uses a subset of the RAM that is used for ISP command handling. This includes location 0x1000 0120 and also parts of the top 32 bytes of on-chip RAM. The stack is located at RAM top - 32. The maximum stack usage is 32 bytes. If the user program assumes that RAM is unchanged during a reset where power is not removed from the device, it is important to be aware of these exceptions.

#### 36.3.2.9 RAM used by IAP command handler

Flash programming commands use the top 32 bytes of on-chip RAM. The maximum stack usage in the user allocated stack space is 128 bytes, growing downwards.

36.4 Boot process flowchart



## 36.5 Sector numbers

Some IAP and ISP commands operate on "sectors" and specify sector numbers. The following table indicate the correspondence between sector numbers and memory addresses for LPC178x/177x devices containing 32, 64, 128, 256 and 512 kB of flash respectively. IAP and ISP routines are located in the Boot ROM.

**Table 698. Sectors in a LPC178x/177x device**

Sector Number	Sector Size [kB]	Start Address	End Address	32 kB Part	64 kB Part	128 kB Part	256 kB Part	512 kB Part
0	4	0X0000 0000	0X0000 0FFF	x	x	x	x	x
1	4	0X0000 1000	0X0000 1FFF	x	x	x	x	x
2	4	0X0000 2000	0X0000 2FFF	x	x	x	x	x
3	4	0X0000 3000	0X0000 3FFF	x	x	x	x	x
4	4	0X0000 4000	0X0000 4FFF	x	x	x	x	x
5	4	0X0000 5000	0X0000 5FFF	x	x	x	x	x
6	4	0X0000 6000	0X0000 6FFF	x	x	x	x	x
7	4	0X0000 7000	0X0000 7FFF	x	x	x	x	x
8	4	0X0000 8000	0X0000 8FFF		x	x	x	x
9	4	0X0000 9000	0X0000 9FFF		x	x	x	x
10 (0x0A)	4	0X0000 A000	0X0000 AFFF		x	x	x	x
11 (0x0B)	4	0X0000 B000	0X0000 BFFF		x	x	x	x
12 (0x0C)	4	0X0000 C000	0X0000 CFFF		x	x	x	x
13 (0x0D)	4	0X0000 D000	0X0000 DFFF		x	x	x	x
14 (0X0E)	4	0X0000 E000	0X0000 EFFF		x	x	x	x
15 (0x0F)	4	0X0000 F000	0X0000 FFFF		x	x	x	x
16 (0x10)	32	0x0001 0000	0x0001 7FFF			x	x	x
17 (0x11)	32	0x0001 8000	0x0001 FFFF			x	x	x
18 (0x12)	32	0x0002 0000	0x0002 7FFF				x	x
19 (0x13)	32	0x0002 8000	0x0002 FFFF				x	x
20 (0x14)	32	0x0003 0000	0x0003 7FFF				x	x
21 (0x15)	32	0x0003 8000	0x0003 FFFF				x	x
22 (0x16)	32	0x0004 0000	0x0004 7FFF					x
23 (0x17)	32	0x0004 8000	0x0004 FFFF					x
24 (0x18)	32	0x0005 0000	0x0005 7FFF					x
25 (0x19)	32	0x0005 8000	0x0005 FFFF					x
26 (0x1A)	32	0x0006 0000	0x0006 7FFF					x
27 (0x1B)	32	0x0006 8000	0x0006 FFFF					x
28 (0x1C)	32	0x0007 0000	0x0007 7FFF					x
29 (0x1D)	32	0x0007 8000	0x0007 FFFF					x

### 36.6 Code Read Protection (CRP)

Code Read Protection is a mechanism that allows user to enable different levels of security in the system so that access to the on-chip flash and use of the ISP can be restricted. When needed, CRP is invoked by programming a specific pattern in flash location at 0x0000 02FC. IAP commands are not affected by the code read protection.

**Important: Any CRP change becomes effective only after the device has gone through a power cycle.**

**Table 699. Code Read Protection options**

Name	Pattern programmed in 0x0000 02FC	Description
CRP1	0x1234 5678	<p>Access to chip via the JTAG pins is disabled. This mode allows partial flash update using the following ISP commands and restrictions:</p> <ul style="list-style-type: none"> <li>• Write to RAM command can not access RAM below 0x1000 0200. This is due to use of the RAM by the ISP code, see <a href="#">Section 36.3.2.7</a>.</li> <li>• Read Memory command: disabled.</li> <li>• Copy RAM to Flash command: cannot write to Sector 0.</li> <li>• Go command: disabled.</li> <li>• Erase sector(s) command: can erase any individual sector except sector 0 only, or can erase all sectors at once.</li> <li>• Compare command: disabled</li> </ul> <p>This mode is useful when CRP is required and flash field updates are needed but all sectors can not be erased. The compare command is disabled, so in the case of partial flash updates the secondary loader should implement a checksum mechanism to verify the integrity of the flash.</p>
CRP2	0x8765 4321	<p>This is similar to CRP1 with the following additions:</p> <ul style="list-style-type: none"> <li>• Write to RAM command: disabled.</li> <li>• Copy RAM to Flash: disabled.</li> <li>• Erase command: only allows erase of all sectors.</li> </ul>
CRP3	0x4321 8765	<p>This is similar to CRP2, but ISP entry by pulling P2[10] LOW is disabled if a valid user code is present in flash sector 0.</p> <p>This mode effectively disables ISP override using the P2[10] pin. It is up to the user's application to provide for flash updates by using IAP calls or by invoking ISP with UART0.</p> <p><b>Caution: If CRP3 is selected, no future factory testing can be performed on the device.</b></p>



**Table 700. Code Read Protection hardware/software interaction**

CRP option	User Code Valid	P2[10] pin at reset	JTAG enabled	LPC178x/177x enters ISP mode	partial flash update in ISP mode
None	No	X	Yes	Yes	Yes
	Yes	High	Yes	No	NA
	Yes	Low	Yes	Yes	Yes
CRP1	No	x	No	Yes	Yes
	Yes	High	No	No	NA
	Yes	Low	No	Yes	Yes
CRP2	No	x	No	Yes	No
	Yes	High	No	No	NA
	Yes	Low	No	Yes	No
CRP3	No	x	No	Yes	No
	Yes	x	No	No	NA

If any CRP mode is enabled and access to the chip is allowed via the ISP, an unsupported or restricted ISP command will be terminated with return code `CODE_READ_PROTECTION_ENABLED`.

### 36.7 ISP commands

The following commands are accepted by the ISP command handler. Detailed status codes are supported for each command. The command handler sends the return code INVALID\_COMMAND when an undefined command is received. Commands and return codes are in ASCII format.

CMD\_SUCCESS is sent by ISP command handler only when received ISP command has been completely executed and the new ISP command can be given by the host. Exceptions from this rule are "Set Baud Rate", "Write to RAM", "Read Memory", and "Go" commands.

**Table 701. ISP command summary**

ISP Command	Usage	Described in
Unlock	U <Unlock Code>	<a href="#">Table 702</a>
Set Baud Rate	B <Baud Rate> <stop bit>	<a href="#">Table 703</a>
Echo	A <setting>	<a href="#">Table 705</a>
Write to RAM	W <start address> <number of bytes>	<a href="#">Table 706</a>
Read Memory	R <address> <number of bytes>	<a href="#">Table 707</a>
Prepare sector(s) for write operation	P <start sector number> <end sector number>	<a href="#">Table 708</a>
Copy RAM to Flash	C <flash address> <RAM address> <number of bytes>	<a href="#">Table 709</a>
Go	G <address> <Mode>	<a href="#">Table 710</a>
Erase sector(s)	E <start sector number> <end sector number>	<a href="#">Table 711</a>
Blank check sector(s)	I <start sector number> <end sector number>	<a href="#">Table 712</a>
Read Part ID	J	<a href="#">Table 713</a>
Read Boot Code version	K	<a href="#">Table 715</a>
Read serial number	N	<a href="#">Table 716</a>
Compare	M <address1> <address2> <number of bytes>	<a href="#">Table 717</a>

#### 36.7.1 Unlock <Unlock code>

**Table 702. ISP Unlock command**

Command	U
Input	Unlock code: 23130 decimal
Return Code	CMD_SUCCESS   INVALID_CODE   PARAM_ERROR
Description	This command is used to unlock Flash Write, Erase, and Go commands.
Example	"U 23130<CR><LF>" unlocks the Flash Write/Erase & Go commands.

### 36.7.2 Set Baud Rate <Baud Rate> <stop bit>

Table 703. ISP Set Baud Rate command

Command	B
Input	Baud Rate: 9600   19200   38400   57600   115200   230400 Stop bit: 1   2
Return Code	CMD_SUCCESS   INVALID_BAUD_RATE   INVALID_STOP_BIT   PARAM_ERROR
Description	This command is used to change the baud rate. The new baud rate is effective after the command handler sends the CMD_SUCCESS return code.
Example	"B 57600 1<CR><LF>" sets the serial port to baud rate 57600 bps and 1 stop bit.

When the Set Baud Rate command is used after ISP has been re-invoked by a user program (using the “Re-invoke ISP” IAP command, see [Section 36.8.9](#)), the baud rate settings will be based on the current peripheral clock frequency. Some examples of frequencies and baud rates supported are shown in [Table 704](#) below. This table shows cases where the rate error is less than ±1.5%. This provides margin for frequencies that may be generated from the IRC, which has a ±1% tolerance of its own, to obtain a total rate error of less than ±2.5% in all cases.

Table 704. Examples of supported ISP baud rates and PCLK frequency (in MHz)

ISP Baud rate vs. PCLK Frequency	9600	19200	38400	57600	115200	230400
10.0000	+	+		+		
11.0592	+	+	+	+	+	+
12.0000 <sup>[1]</sup>	+	+		+		
12.2880	+	+	+			
14.7456	+	+	+	+	+	+
15.3600	+	+	+			
18.4320	+	+	+	+	+	+
19.6608	+	+	+			
24.5760	+	+	+	+		
25.0000	+	+	+	+		

[1] ISP entry after reset uses the on chip IRC to run the device at PCLK = 12 MHz.

### 36.7.3 Echo <setting>

Table 705. ISP Echo command

Command	A
Input	Setting: ON = 1   OFF = 0
Return Code	CMD_SUCCESS   PARAM_ERROR
Description	The default setting for echo command is ON. When ON the ISP command handler sends the received serial data back to the host.
Example	"A 0<CR><LF>" turns echo off.

### 36.7.4 Write to RAM <start address> <number of bytes>

The host should send the data only after receiving the CMD\_SUCCESS return code. The host should send the check-sum after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines then the check-sum should be of the actual number of bytes sent. The ISP command handler compares it with the check-sum of the received bytes. If the check-sum matches, the ISP command handler responds with "OK<CR><LF>" to continue further transmission. If the check-sum does not match, the ISP command handler responds with "RESEND<CR><LF>". In response the host should retransmit the bytes.

Table 706. ISP Write to RAM command

Command	W
Input	<b>Start Address:</b> RAM address where data bytes are to be written. This address should be a word boundary. <b>Number of Bytes:</b> Number of bytes to be written. Count should be a multiple of 4
Return Code	CMD_SUCCESS   ADDR_ERROR (Address not on word boundary)   ADDR_NOT_MAPPED   COUNT_ERROR (Byte count is not multiple of 4)   PARAM_ERROR   CODE_READ_PROTECTION_ENABLED
Description	This command is used to download data to RAM. Data should be in UU-encoded format. This command is blocked when code read protection levels CRP2 or CRP3 are enabled.
Example	"W 268435968 4<CR><LF>" writes 4 bytes of data to address 0x1000 0200.

### 36.7.5 Read Memory <address> <no. of bytes>

The data stream is followed by the command success return code. The check-sum is sent after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines then the check-sum is of actual number of bytes sent. The host should compare it with the checksum of the received bytes. If the check-sum matches then the host should respond with

"OK<CR><LF>" to continue further transmission. If the check-sum does not match then the host should respond with "RESEND<CR><LF>". In response the ISP command handler sends the data again.

**Table 707. ISP Read Memory command**

Command	R
Input	<b>Start Address:</b> Address from where data bytes are to be read. This address should be a word boundary. <b>Number of Bytes:</b> Number of bytes to be read. Count should be a multiple of 4.
Return Code	CMD_SUCCESS followed by <actual data (UU-encoded)>   ADDR_ERROR (Address not on word boundary)   ADDR_NOT_MAPPED   COUNT_ERROR (Byte count is not a multiple of 4)   PARAM_ERROR   CODE_READ_PROTECTION_ENABLED
Description	This command is used to read data from RAM or flash memory. This command is blocked when any level of code read protection is enabled.
Example	"R 268435968 4<CR><LF>" reads 4 bytes of data from address 0x1000 0200.

### 36.7.6 Prepare sector(s) for write operation <start sector number> <end sector number>

This command makes flash write/erase operation a two step process.

**Table 708. ISP Prepare sector(s) for write operation command**

Command	P
Input	<b>Start Sector Number</b> <b>End Sector Number:</b> Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS   BUSY   INVALID_SECTOR   PARAM_ERROR
Description	This command must be executed before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. To prepare a single sector use the same "Start" and "End" sector numbers.
Example	"P 0 0<CR><LF>" prepares the flash sector 0.

### 36.7.7 Copy RAM to Flash <flash address> <RAM address> <no of bytes>

Table 709. ISP Copy command

Command	C
Input	<p><b>Flash Address(DST):</b> Destination flash address where data bytes are to be written. The destination address should be a 256 byte boundary.</p> <p><b>RAM Address(SRC):</b> Source RAM address from where data bytes are to be read.</p> <p><b>Number of Bytes:</b> Number of bytes to be written. Should be 256   512   1024   4096.</p>
Return Code	<p>CMD_SUCCESS  </p> <p>SRC_ADDR_ERROR (Address not on word boundary)  </p> <p>DST_ADDR_ERROR (Address not on correct boundary)  </p> <p>SRC_ADDR_NOT_MAPPED  </p> <p>DST_ADDR_NOT_MAPPED  </p> <p>COUNT_ERROR (Byte count is not 256   512   1024   4096)  </p> <p>SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION  </p> <p>BUSY  </p> <p>CMD_LOCKED  </p> <p>PARAM_ERROR  </p> <p>CODE_READ_PROTECTION_ENABLED</p>
Description	<p>This command is used to program the flash memory. The "Prepare Sector(s) for Write Operation" command should precede this command. The affected sectors are automatically protected again once the copy command is successfully executed. This command is blocked when code read protection levels CRP2 or CRP3 are enabled. When code read protection level CRP1 is enabled, individual sectors other than sector 0 can be written.</p>
Example	<p>"C 0 268468224 512&lt;CR&gt;&lt;LF&gt;" copies 512 bytes from the RAM address 0x1000 8000 to the flash address 0.</p>

### 36.7.8 Go <address> <mode>

Table 710. ISP Go command

Command	G
Input	<p><b>Address:</b> Flash or RAM address from which the code execution is to be started. This address should be on a word boundary.</p> <p><b>Mode (retained for backward compatibility):</b> T (Execute program in Thumb Mode)   A (not allowed).</p>
Return Code	<p>CMD_SUCCESS  </p> <p>ADDR_ERROR  </p> <p>ADDR_NOT_MAPPED  </p> <p>CMD_LOCKED  </p> <p>PARAM_ERROR  </p> <p>CODE_READ_PROTECTION_ENABLED</p>
Description	<p>This command is used to execute a program residing in RAM or flash memory. It may not be possible to return to the ISP command handler once this command is successfully executed. This command is blocked when any level of code read protection is enabled.</p>
Example	<p>"G 0 T&lt;CR&gt;&lt;LF&gt;" branches to address 0x0000 0000.</p>

When the GO command is used, execution begins at the specified address (assuming it is an executable address) with the device left as it was configured for the ISP code. This means that some things are different than they would be for entering user code directly following a chip reset. The CPU will be running from the 12 MHz IRC with the PLLs turned off.

### 36.7.9 Erase sector(s) <start sector number> <end sector number>

Table 711. ISP Erase sector command

Command	E
Input	<b>Start Sector Number</b> <b>End Sector Number:</b> Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS   BUSY   INVALID_SECTOR   SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION   CMD_LOCKED   PARAM_ERROR   CODE_READ_PROTECTION_ENABLED
Description	This command is used to erase one or more sector(s) of on-chip flash memory. This command is blocked when code read protection level CRP3 is enabled. When code read protection level CRP1 is enabled, individual sectors other than sector 0 can be erased. All sectors can be erased at once in CRP1 and CRP2.
Example	"E 2 3<CR><LF>" erases the flash sectors 2 and 3.

### 36.7.10 Blank check sector(s) <sector number> <end sector number>

Table 712. ISP Blank check sector command

Command	I
Input	<b>Start Sector Number:</b> <b>End Sector Number:</b> Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS   SECTOR_NOT_BLANK (followed by <Offset of the first non blank word location> <Contents of non blank word location>)   INVALID_SECTOR   PARAM_ERROR
Description	This command is used to blank check one or more sectors of on-chip flash memory.
Example	"I 2 3<CR><LF>" blank checks the flash sectors 2 and 3.

### 36.7.11 Read Part Identification number

Table 713. ISP Read Part Identification command

Command	J
Input	None.
Return Code	CMD_SUCCESS followed by part identification number in ASCII (see <a href="#">Table 714 "LPC178x/177x part identification numbers"</a> ).
Description	This command is used to read the part identification number. The part identification number maps to a feature subset within a device family. This number will not normally change as a result of technical revisions.

**Table 714. LPC178x/177x part identification numbers**

Device	ASCII/dec coding	Hex coding
LPC1788	673005383	0x281D 3F47
LPC1787	673003335	0x281D 3747
LPC1786	672997187	0x281D 1F43
LPC1785	672995139	0x281D 1743
LPC1778	655966023	0x2719 3F47
LPC1777	655963975	0x2719 3747
LPC1776	655957827	0x2719 1F43
LPC1774	654381362	0x2701 1132

### 36.7.12 Read Boot Code version number

**Table 715. ISP Read Boot Code version number command**

Command	K
Input	None
Return Code	CMD_SUCCESS followed by 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)>.
Description	This command is used to read the boot code version number.

### 36.7.13 Read device serial number

**Table 716. ISP Read device serial number command**

Command	N
Input	None.
Return Code	CMD_SUCCESS followed by the device serial number in 4 decimal ASCII groups, each representing a 32-bit value.
Description	This command is used to read the device serial number. The serial number may be used to uniquely identify a single unit among all LPC178x/177x devices.

### 36.7.14 Compare <address1> <address2> <no of bytes>

**Table 717. ISP Compare command**

Command	M
Input	<p><b>Address1 (DST):</b> Starting flash or RAM address of data bytes to be compared. This address should be a word boundary.</p> <p><b>Address2 (SRC):</b> Starting flash or RAM address of data bytes to be compared. This address should be a word boundary.</p> <p><b>Number of Bytes:</b> Number of bytes to be compared; should be a multiple of 4.</p>



Table 717. ISP Compare command

Command	M
Return Code	CMD_SUCCESS   (Source and destination data are equal) COMPARE_ERROR   (Followed by the offset of first mismatch) COUNT_ERROR (Byte count is not a multiple of 4)   ADDR_ERROR   ADDR_NOT_MAPPED   PARAM_ERROR
Description	This command is used to compare the memory contents at two locations. This command is blocked when any level of code read protection is enabled.
Example	"M 8192 268435968 4<CR><LF>" compares 4 bytes from the RAM address 0x1000 0200 to the 4 bytes from the flash address 0x2000.

### 36.7.15 ISP Return Codes

Table 718. ISP Return Codes Summary

Return Code	Mnemonic	Description
0	CMD_SUCCESS	Command is executed successfully. Sent by ISP handler only when command given by the host has been completely and successfully executed.
1	INVALID_COMMAND	Invalid command.
2	SRC_ADDR_ERROR	Source address is not on word boundary.
3	DST_ADDR_ERROR	Destination address is not on a correct boundary.
4	SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken into consideration where applicable.
5	DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken into consideration where applicable.
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid or end sector number is greater than start sector number.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data not equal.
11	BUSY	Flash programming hardware interface is busy.
12	PARAM_ERROR	Insufficient number of parameters or invalid parameter.
13	ADDR_ERROR	Address is not on word boundary.
14	ADDR_NOT_MAPPED	Address is not mapped in the memory map. Count value is taken in to consideration where applicable.
15	CMD_LOCKED	Command is locked.
16	INVALID_CODE	Unlock code is invalid.
17	INVALID_BAUD_RATE	Invalid baud rate setting.
18	INVALID_STOP_BIT	Invalid stop bit setting.
19	CODE_READ_PROTECTION_ENABLED	Code read protection enabled.

## 36.8 IAP commands

For in application programming the IAP routine should be called with a word pointer in register r0 pointing to memory (RAM) containing command code and parameters. The result from the IAP command is returned in the table pointed to by register r1. The user can reuse the command table for result by passing the same pointer in registers r0 and r1. The parameter table should be big enough to hold all the results in case if number of results are more than number of parameters. Parameter passing is illustrated in the [Figure 168](#). The number of parameters and results vary according to the IAP command. The maximum number of parameters is 5, passed to the "Copy RAM to Flash" command. The maximum number of results is 4, returned by the "Read device serial number" command. The command handler sends the status code INVALID\_COMMAND when an undefined command is received. The IAP routine resides at location 0x1FFF1FF0.

The IAP function could be called in the following way using C.

Define the IAP location entry point. Bit 0 of the IAP location is set since the Cortex-M3 uses only Thumb mode.

```
#define IAP_LOCATION 0x1FFF1FF1
```

Define data structure or pointers to pass IAP command table and result table to the IAP function:

```
unsigned long command[5];
unsigned long output[5];
```

or

```
unsigned long * command;
unsigned long * output;
command=(unsigned long *) 0x...
output= (unsigned long *) 0x...
```

Define a pointer to function type, which takes two parameters and returns void. Note the IAP returns the result with the base address of the table residing in R1.

```
typedef void (*IAP)(unsigned int [],unsigned int[]);
IAP iap_entry;
```

Setting function pointer:

```
iap_entry=(IAP) IAP_LOCATION;
```

Whenever you wish to call IAP you could use the following statement.

```
iap_entry (command, output);
```

The IAP call could be simplified further by using the symbol definition file feature supported by ARM Linker in ADS (ARM Developer Suite). You could also call the IAP routine using assembly code.

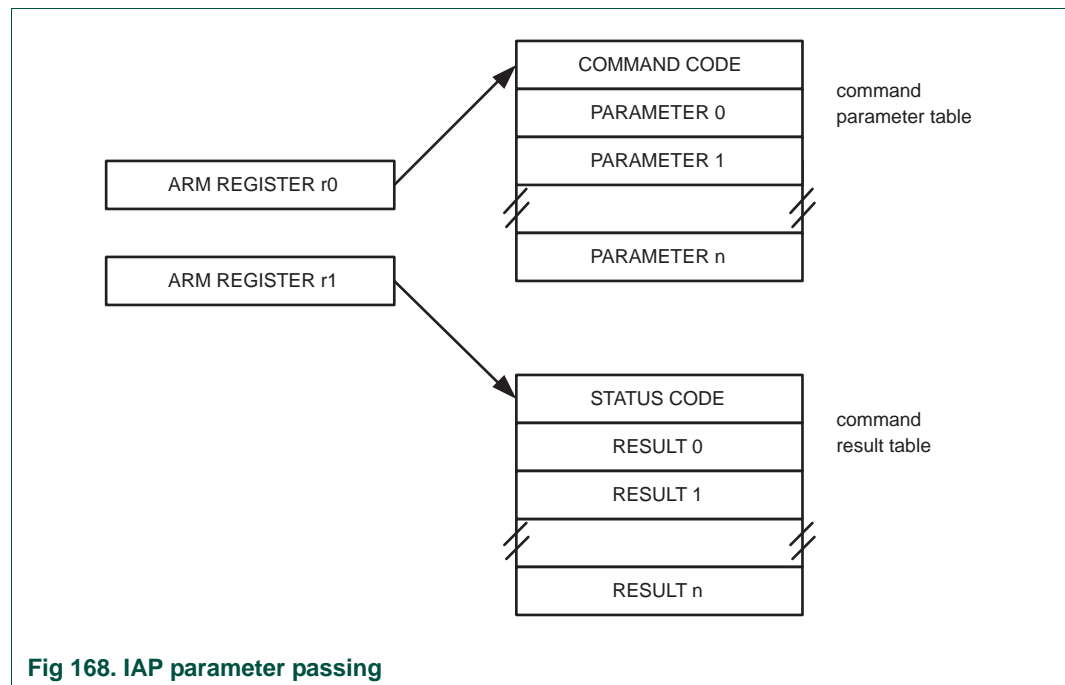
Note that the first entry in the command table is the IAP command, followed by any required command parameters, starting with Param0. The first entry in the output table is the Return Code, followed by any other results, starting with Result0.

As per the ARM specification (The ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05) up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively. Additional parameters are passed on the stack. Up to 4 parameters can be returned in the r0, r1, r2 and r3 registers respectively. Additional parameters are returned indirectly via memory. Some of the IAP calls require more than 4 parameters. If the ARM suggested scheme is used for the parameter passing/returning then it might create problems due to difference in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such risk.

The flash memory is not accessible during a write or erase operation. IAP commands, which results in a flash write/erase operation, use 32 bytes of space in the top portion of the on-chip RAM for execution. The user program should not be use this space if IAP flash programming is permitted in the application.

**Table 719. IAP Command Summary**

IAP Command	Command Code	Described in
Prepare sector(s) for write operation	50 decimal	<a href="#">Table 720</a>
Copy RAM to Flash	51 decimal	<a href="#">Table 721</a>
Erase sector(s)	52 decimal	<a href="#">Table 722</a>
Blank check sector(s)	53 decimal	<a href="#">Table 723</a>
Read part ID	54 decimal	<a href="#">Table 724</a>
Read Boot Code version	55 decimal	<a href="#">Table 725</a>
Read device serial number	58 decimal	<a href="#">Table 726</a>
Compare	56 decimal	<a href="#">Table 727</a>
Reinvoke ISP	57 decimal	<a href="#">Table 728</a>



**Fig 168. IAP parameter passing**

### 36.8.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two step process.

**Table 720. IAP Prepare sector(s) for write operation command**

Command	Prepare sector(s) for write operation
Input	<p><b>Command code: 50 decimal</b></p> <p><b>Param0:</b> Start Sector Number</p> <p><b>Param1:</b> End Sector Number (should be greater than or equal to start sector number).</p>
Return Code	<p>CMD_SUCCESS  </p> <p>BUSY  </p> <p>INVALID_SECTOR</p>
Result	None
Description	<p>This command must be executed before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. To prepare a single sector use the same "Start" and "End" sector numbers.</p>

### 36.8.2 Copy RAM to Flash

**Table 721. IAP Copy RAM to Flash command**

Command	Copy RAM to Flash
Input	<p><b>Command code: 51 decimal</b></p> <p><b>Param0(DST):</b> Destination flash address where data bytes are to be written. This address should be a 256 byte boundary.</p> <p><b>Param1(SRC):</b> Source RAM address from which data bytes are to be read. This address should be a word boundary.</p> <p><b>Param2:</b> Number of bytes to be written. Should be 256   512   1024   4096.</p> <p><b>Param3:</b> CPU Clock Frequency (CCLK) in kHz.</p>
Return Code	<p>CMD_SUCCESS  </p> <p>SRC_ADDR_ERROR (Address not a word boundary)  </p> <p>DST_ADDR_ERROR (Address not on correct boundary)  </p> <p>SRC_ADDR_NOT_MAPPED  </p> <p>DST_ADDR_NOT_MAPPED  </p> <p>COUNT_ERROR (Byte count is not 256   512   1024   4096)  </p> <p>SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION  </p> <p>BUSY  </p>
Result	None
Description	<p>This command is used to program the flash memory. The affected sectors should be prepared first by calling "Prepare Sector for Write Operation" command. The affected sectors are automatically protected again once the copy command is successfully executed.</p>

### 36.8.3 Erase Sector(s)

Table 722. IAP Erase Sector(s) command

Command	Erase Sector(s)
Input	<b>Command code: 52 decimal</b> <b>Param0:</b> Start Sector Number <b>Param1:</b> End Sector Number (should be greater than or equal to start sector number). <b>Param2:</b> CPU Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS   BUSY   SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION   INVALID_SECTOR
Result	None
Description	This command is used to erase a sector or multiple sectors of on-chip flash memory. To erase a single sector use the same "Start" and "End" sector numbers.

### 36.8.4 Blank check sector(s)

Table 723. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	<b>Command code: 53 decimal</b> <b>Param0:</b> Start Sector Number <b>Param1:</b> End Sector Number (should be greater than or equal to start sector number).
Return Code	CMD_SUCCESS   BUSY   SECTOR_NOT_BLANK   INVALID_SECTOR
Result	<b>Result0:</b> Offset of the first non blank word location if the Status Code is SECTOR_NOT_BLANK. <b>Result1:</b> Contents of non blank word location.
Description	This command is used to blank check a sector or multiple sectors of on-chip flash memory. To blank check a single sector use the same "Start" and "End" sector numbers.

### 36.8.5 Read part identification number

Table 724. IAP Read part identification number command

Command	Read part identification number
Input	<b>Command code: 54 decimal</b> <b>Parameters:</b> None
Return Code	CMD_SUCCESS
Result	<b>Result0:</b> Part Identification Number.
Description	This command is used to read the part identification number. The value returned is the hexadecimal version of the part ID. See <a href="#">Table 714 "LPC178x/177x part identification numbers"</a> .

### 36.8.6 Read Boot Code version number

Table 725. IAP Read Boot Code version number command

<b>Command</b>	<b>Read boot code version number</b>
Input	<b>Command code: 55 decimal</b> <b>Parameters:</b> None
Return Code	CMD_SUCCESS
Result	<b>Result0:</b> 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)>
Description	This command is used to read the boot code version number.

### 36.8.7 Read device serial number

Table 726. IAP Read device serial number command

<b>Command</b>	<b>Read device serial number</b>
Input	<b>Command code: 58 decimal</b> <b>Parameters:</b> None
Return Code	CMD_SUCCESS
Result	<b>Result0:</b> First 32-bit word of Device Identification Number (at the lowest address) <b>Result1:</b> Second 32-bit word of Device Identification Number <b>Result2:</b> Third 32-bit word of Device Identification Number <b>Result3:</b> Fourth 32-bit word of Device Identification Number
Description	This command is used to read the device identification number. The serial number may be used to uniquely identify a single unit among all LPC178x/177x devices.

### 36.8.8 Compare <address1> <address2> <no of bytes>

Table 727. IAP Compare command

<b>Command</b>	<b>Compare</b>
Input	<b>Command code: 56 decimal</b> <b>Param0(DST):</b> Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. <b>Param1(SRC):</b> Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. <b>Param2:</b> Number of bytes to be compared; should be a multiple of 4.
Return Code	CMD_SUCCESS   COMPARE_ERROR   COUNT_ERROR (Byte count is not a multiple of 4)   ADDR_ERROR   ADDR_NOT_MAPPED
Result	<b>Result0:</b> Offset of the first mismatch if the Status Code is COMPARE_ERROR.
Description	This command is used to compare the memory contents at two locations. <b>The result may not be correct when the source or destination includes any of the first 64 bytes starting from address zero. The first 64 bytes can be re-mapped to RAM.</b>

### 36.8.9 Re-invoke ISP

Table 728. Re-invoke ISP

Command	Compare
Input	<b>Command code: 57 decimal</b>
Return Code	None
Result	<b>None.</b>
Description	<p>This command is used to invoke the boot loader in ISP mode. It maps boot vectors, sets PCLK = CCLK / 4, configures UART0 pins Rx and Tx, resets TIMER1 and resets the U0FDR (see <a href="#">Section 18.6.11</a>). This command may be used when a valid user program is present in the internal flash memory and the P2[10] pin is not accessible to force the ISP mode. The command does not disable the PLL hence it is possible to invoke the boot loader when the part is running off the PLL. In such case the ISP utility should pass the CCLK (crystal or PLL output depending on the clock source selection <a href="#">Section 4.6</a>) frequency after auto-baud handshake.</p> <p>Another option is to disable the PLL and select the IRC as the clock source before making this IAP call. In this case frequency sent by ISP is ignored and the 12 MHz IRC is directly for the CPU clock.</p>

### 36.8.10 IAP Status Codes

Table 729. IAP Status Codes Summary

Status Code	Mnemonic	Description
0	CMD_SUCCESS	Command is executed successfully.
1	INVALID_COMMAND	Invalid command.
2	SRC_ADDR_ERROR	Source address is not on a word boundary.
3	DST_ADDR_ERROR	Destination address is not on a correct boundary.
4	SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken in to consideration where applicable.
5	DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable.
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data is not same.
11	BUSY	Flash programming hardware interface is busy.

## 36.9 JTAG flash programming interface

Debug tools can write parts of the flash image to the RAM and then execute the IAP call "Copy RAM to Flash" repeatedly with proper offset.

## 36.10 Flash signature generation

The flash module contains a built-in signature generator. This generator can produce a 128-bit signature from a range of flash memory. A typical usage is to verify the flashed contents against a calculated signature (e.g. during programming).

The address range for generating a signature must be aligned on flash-word boundaries, i.e. 128-bit boundaries. Once started, signature generation completes independently. While signature generation is in progress, the flash memory cannot be accessed for other purposes, and an attempted read will cause a wait state to be asserted until signature generation is complete. Code outside of the flash (e.g. internal RAM) can be executed during signature generation. This can include interrupt services, if the interrupt vector table is re-mapped to memory other than the flash memory. The code that initiates signature generation should also be placed outside of the flash memory.

### 36.10.1 Register description for signature generation

**Table 730. Register overview: FMC (base address 0x0020 0000)**

Name	Description	Access	Reset Value	Address	Reference
FMSSTART	Signature start address register	R/W	0	0x0020 0020	<a href="#">Table 731</a>
FMSSTOP	Signature stop-address register	R/W	0	0x0020 0024	<a href="#">Table 732</a>
FMSW0	128-bit signature Word 0	RO	-	0x0020 002C	<a href="#">Table 733</a>
FMSW1	128-bit signature Word 1	RO	-	0x0020 0030	<a href="#">Table 734</a>
FMSW2	128-bit signature Word 2	RO	-	0x0020 0034	<a href="#">Table 735</a>
FMSW3	128-bit signature Word 3	RO	-	0x0020 0038	<a href="#">Table 736</a>
FMSTAT	Signature generation status register	RO	0	0x0020 0FE0	<a href="#">Section 36.10.1.3</a>
FMSTATCLR	Signature generation status clear register	WO	-	0x0020 0FE8	<a href="#">Section 36.10.1.4</a>



### 36.10.1.1 Signature generation address and control registers

These registers control automatic signature generation. A signature can be generated for any part of the flash memory contents. The address range to be used for generation is defined by writing the start address to the signature start address register (FMSSTART) and the stop address to the signature stop address register (FMSSTOP. The start and stop addresses must be aligned to 128-bit boundaries and can be derived by dividing the byte address by 16.

Signature generation is started by setting the SIG\_START bit in the FMSSTOP register. Setting the SIG\_START bit is typically combined with the signature stop address in a single write.

[Table 731](#) and [Table 732](#) show the bit assignments in the FMSSTART and FMSSTOP registers respectively.

**Table 731. Flash Module Signature Start register (FMSSTART - 0x0020 0020) bit description**

Bit	Symbol	Description	Reset Value
16:0	START	Signature generation start address (corresponds to AHB byte address bits[20:4]).	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

**Table 732. Flash Module Signature Stop register (FMSSTOP - 0x0020 0024) bit description**

Bit	Symbol	Value	Description	Reset Value
16:0	STOP		BIST stop address divided by 16 (corresponds to AHB byte address [20:4]).	0
17	SIG_START		Start control bit for signature generation.	0
		0	Signature generation is stopped	
		1	Initiate signature generation	
31:18	-		Reserved. Read value is undefined, only zero should be written.	NA

### 36.10.1.2 Signature generation result registers

The signature generation result registers return the flash signature produced by the embedded signature generator. The 128-bit signature is reflected by the four registers FMSW0, FMSW1, FMSW2 and FMSW3.

The generated flash signature can be used to verify the flash memory contents. The generated signature can be compared with an expected signature and thus makes saves time and code space. The method for generating the signature is described in [Section 36.10.2](#).

[Table 736](#) show bit assignment of the FMSW0 and FMSW1, FMSW2, FMSW3 registers respectively.

**Table 733. FMSW0 register bit description (FMSW0, address: 0x0020 002C)**

Bit	Symbol	Description	Reset Value
31:0	SW0[31:0]	Word 0 of 128-bit signature (bits 31 to 0).	-

**Table 734. FMSW1 register bit description (FMSW1, address: 0x0020 0030)**

Bit	Symbol	Description	Reset Value
31:0	SW1[63:32]	Word 1 of 128-bit signature (bits 63 to 32).	-

**Table 735. FMSW2 register bit description (FMSW2, address: 0x0020 0034)**

Bit	Symbol	Description	Reset Value
31:0	SW2[95:64]	Word 2 of 128-bit signature (bits 95 to 64).	-

**Table 736. FMSW3 register bit description (FMSW3, address: 0x0020 0038)**

Bit	Symbol	Description	Reset Value
31:0	SW3[127:96]	Word 3 of 128-bit signature (bits 127 to 96).	-

### 36.10.1.3 Flash Module Status register (FMSTAT - 0x0x0020 0FE0)

The read-only FMSTAT register provides a means of determining when signature generation has completed. Completion of signature generation can be checked by polling the SIG\_DONE bit in FMSTAT. SIG\_DONE should be cleared via the FMSTATCLR register before starting a signature generation operation, otherwise the status might indicate completion of a previous operation.

**Table 737. Flash module Status register (FMSTAT - 0x0020 0FE0) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. The value read from a reserved bit is not defined.	NA
2	SIG_DONE	When 1, a previously started signature generation has completed. See FMSTATCLR register description for clearing this flag.	0
31:2	-	Reserved. The value read from a reserved bit is not defined.	NA

### 36.10.1.4 Flash Module Status Clear register (FMSTATCLR - 0x0x0020 0FE8)

The FMSTATCLR register is used to clear the signature generation completion flag.

**Table 738. Flash Module Status Clear register (FMSTATCLR - 0x0x0020 0FE8) bit description**

Bit	Symbol	Description	Reset Value
1:0	-	Reserved. Read value is undefined, only zero should be written.	NA
2	SIG_DONE_CLR	Writing a 1 to this bits clears the signature generation completion flag (SIG_DONE) in the FMSTAT register.	0
31:2	-	Reserved. Read value is undefined, only zero should be written.	NA

### 36.10.2 Algorithm and procedure for signature generation

#### Signature generation

A signature can be generated for any part of the flash contents. The address range to be used for signature generation is defined by writing the start address to the FMSSTART register, and the stop address to the FMSSTOP register.

The signature generation is started by writing a '1' to FMSSTOP.MISR\_START. Starting the signature generation is typically combined with defining the stop address, which is done in another field FMSSTOP.FMSSTOP of the same register.

The time that the signature generation takes is proportional to the address range for which the signature is generated. Reading of the flash memory for signature generation uses a self-timed read mechanism and does not depend on any configurable timing settings for the flash. A safe estimation for the duration of the signature generation is:

$$\text{Duration} = \text{int}((60 / t_{cy}) + 3) \times (\text{FMSSTOP} - \text{FMSSTART} + 1)$$

When signature generation is triggered via software, the duration is in AHB clock cycles, and  $t_{cy}$  is the time in ns for one AHB clock. The SIG\_DONE bit in FMSTAT can be polled by software to determine when signature generation is complete.

If signature generation is triggered via JTAG, the duration is in JTAG tck cycles, and  $t_{cy}$  is the time in ns for one JTAG clock. Polling the SIG\_DONE bit in FMSTAT is not possible in this case.

After signature generation, a 128-bit signature can be read from the FMSW0 to FMSW3 registers. The 128-bit signature reflects the corrected data read from the flash. The 128-bit signature reflects flash parity bits and check bit values.

#### Content verification

The signature as it is read from the FMSW0 to FMSW3 registers must be equal to the reference signature. The algorithms to derive the reference signature is given in [Figure 169](#).

```

sign = 0
FOR address = FMSTART.FMSTART TO FMSTOP.FMSTOP
{
    FOR i = 0 TO 126
        nextSign[i] = f_Q[address][i] XOR sign[i+1]
        nextSign[127] = f_Q[address][127] XOR sign[0] XOR sign[2] XOR
            sign[27] XOR sign[29]
    sign = nextSign
}
signature128 = sign

```

**Fig 169. Algorithm for generating a 128 bit signature**

### 37.1 Features

---

- Supports both standard JTAG and ARM Serial Wire Debug modes.
- Direct debug access to all memories, registers, and peripherals.
- No target resources are required for the debugging session.
- Trace port provides CPU instruction trace capability.
- Eight Breakpoints. Six instruction breakpoints that can also be used to remap instruction addresses for code patches. Two data comparators that can be used to remap addresses for patches to literal values.
- Four data Watchpoints that can also be used as trace triggers.
- Instrumentation Trace Macrocell allows additional software controlled trace capability.

### 37.2 Introduction

---

Debug and trace functions are integrated into the ARM Cortex-M3. Serial wire debug and trace functions are supported in addition to a standard JTAG debug and parallel trace functions. The ARM Cortex-M3 is configured to support up to eight breakpoints and four watchpoints.

### 37.3 Description

---

Debugging with the LPC178x/177x defaults to JTAG. Once in the JTAG debug mode, the debug tool can switch to Serial Wire Debug mode.

Instruction trace is supported by a 4-bit parallel interface using 5 pins. Note that the trace function available for the Cortex-M3 is functionally very different than the trace that was available for previous ARM7 based devices, using only 5 pins instead of 10.

### 37.4 Pin description

The tables below indicate the various pin functions related to debug and trace. Some of these functions share pins with other functions which therefore may not be used at the same time. Use of the JTAG port excludes use of Serial Wire Debug and Serial Wire Output. Use of the parallel trace requires 5 pins that may be part of the user application, limiting debug possibilities for those features.

**Table 739. JTAG pin description**

Pin Name	Type	Description
JTAG_TCK	Input	<b>JTAG Test Clock.</b> This pin is the clock for debug logic when in the JTAG debug mode.
JTAG_TMS	Input	<b>JTAG Test Mode Select.</b> The TMS pin selects the next state in the TAP state machine. This pin includes an internal pull-up for compliance with IEEE 1149.1.
JTAG_TDI	Input	<b>JTAG Test Data In.</b> This is the serial data input for the shift register. This pin includes an internal pull-up for compliance with IEEE 1149.1.
JTAG_TDO	Output	<b>JTAG Test Data Output.</b> This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal.
JTAG_TRST	Input	<b>JTAG Test Reset.</b> The $\overline{\text{JTAG\_TRST}}$ pin can be used to reset the test logic within the debug logic. This pin includes an internal pull-up for compliance with IEEE 1149.1.

**Table 740. Serial Wire Debug pin description**

Pin Name	Type	Description
SWDCLK	Input	<b>Serial Wire Clock.</b> This pin is the clock for debug logic when in the Serial Wire Debug mode. This is an internally selected alternate function for the JTAG_TCK pin.
SWDIO	Input / Output	<b>Serial wire debug data input/output.</b> The SWDIO pin is used by an external debug tool to communicate with and control the Cortex-M3 CPU. This is an internally selected alternate function for the JTAG_TMS pin.
SWO	Output	<b>Serial Wire Output.</b> The SWO pin optionally provides data from the ITM and/or the ETM for an external debug tool to evaluate. This is an internally selected alternate function for the JTAG_TDO pin.

**Table 741. Parallel Trace pin description**

Pin Name	Type	Description
TRACECLK	Input	<b>Trace Clock.</b> This pin provides the sample clock for trace data on the TRACEDATA pins when tracing is enabled by an external debug tool.
TRACEDATA[3:0]	Output	<b>Trace Data bits 3 to 0.</b> These pins provide ETM trace data when tracing is enabled by an external debug tool. The debug tool can then interpret the compressed information and make it available to the user.

### 37.5 Debug connections

The LPC178x/7x supplies dedicated pins for JTAG and Serial Wire Debug (SWD). When a debug session is started, the part will be in JTAG debug mode as recommended by ARM Ltd at the time of design. Once in debug mode, the debugger can switch the device to SWD mode.

Connections from a target board to the debugger can vary. Selecting a debug connector to add to a new board design depends on the debug tool(s) that will be used. For example, debug tools for ARM-based devices in the past have used a standard connection as shown in [Figure 170](#). This diagram has been adapted to fit the LPC178x/7x, taking into account the pins that have built-in pull-ups.

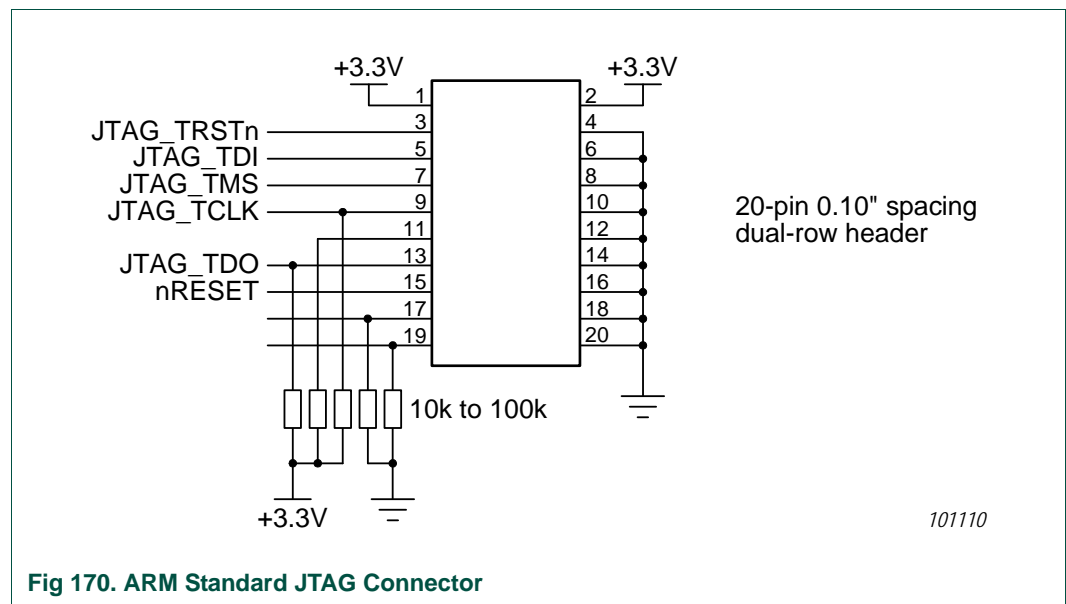
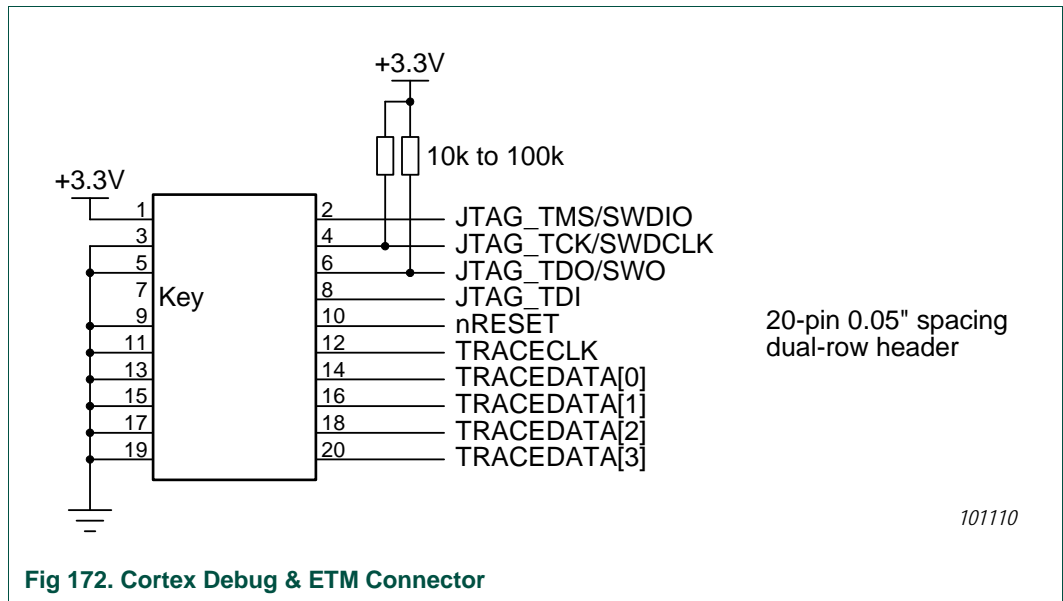
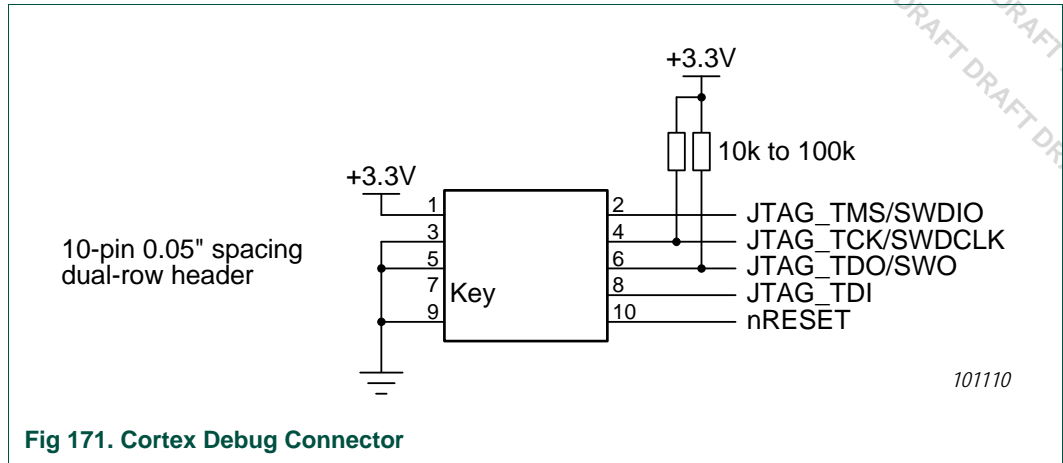


Fig 170. ARM Standard JTAG Connector

Newer tools may use a small debug-only connector as shown in [Figure 171](#). If the debug trace feature will be used, there is also a debug-with-trace connector specification as shown in [Figure 172](#). These 2 connector pinouts are defined in ARM Ltd's CoreSight™ Components Technical Reference Manual. Please note that any debug connection scheme should be checked with the tool vendor before an application board is designed.



### 37.6 JTAG TAP Identification

The JTAG TAP controller contains device ID that can be used by debugging software to identify the general type of device. More detailed device information is available through ISP/IAP commands (see [Section 36.7](#) and [Section 36.8](#)). For the LPC178x/177x family, this ID value is 0x4BA0 0477.

## 37.7 Debug Notes

**Important:** The user should be aware of certain limitations during debugging. The most important is that, due to limitations of the Cortex-M3 integration, the LPC178x/177x cannot wake up in the usual manner from Deep Sleep and Power-down modes. It is recommended not to use these modes during debug.

Once an application is downloaded via JTAG/SWD interface, the USB to SWD/JTAG debug adapter should be removed from the target board, and thereafter, power cycle the LPC178x/177x to allow wake-up from Deep Sleep and Power-down modes.

Another issue is that debug mode changes the way in which reduced power modes are handled by the Cortex-M3 CPU. This causes power modes at the debug level to be different from normal mode operation. These differences mean that power measurements should not be made while debugging, the results will be higher than during normal operation in an application.

During a debugging session, the System Tick Timer is automatically stopped whenever the CPU is stopped. Other peripherals are not affected.

Debugging is disabled if code read protection is enabled.

## 37.8 Debug memory re-mapping

Following chip reset, a portion of the Boot ROM is mapped to address 0 so that it will be automatically executed. The Boot ROM switches the map to point to Flash memory prior to user code being executed. In this way a user normally does not need to know that this re-mapping occurs.

However, when a debugger halts CPU execution immediately following reset, the Boot ROM is still mapped to address 0 and can cause confusion. Ideally, the debugger should correct the mapping automatically in this case, so that a user does not need to deal with it.

### 37.8.1 Memory Mapping Control register (MEMMAP - 0x400F C040)

The MEMMAP register allows switch the mapping of the bottom of memory, including default reset and interrupt vectors, between the Boot ROM and the bottom of on-chip Flash memory.

**Table 742. Memory Mapping Control register (MEMMAP - 0x400F C040) bit description**

Bit	Symbol	Value	Description	Reset value
0	MAP		Memory map control.	0
		0	Boot mode. A portion of the Boot ROM is mapped to address 0.	
		1	User mode. The on-chip Flash memory is mapped to address 0.	
31:1	-		Reserved. Read value is undefined, only zero should be written.	NA



### 38.1 ARM Cortex-M3 User Guide: Introduction

The material in this appendix is provided by ARM Limited for inclusion in the User Manuals of devices containing the Cortex-M3 CPU. Minimal changes have been made to reflect implementation options and other distinctions that apply specifically to LPC178x/177x devices.

#### 38.1.1 About the processor and core peripherals

The Cortex-M3 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including:

- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- efficient processor core, system and memories
- ultra-low power consumption with integrated sleep modes
- platform security, with optional integrated **memory protection unit (MPU)**.

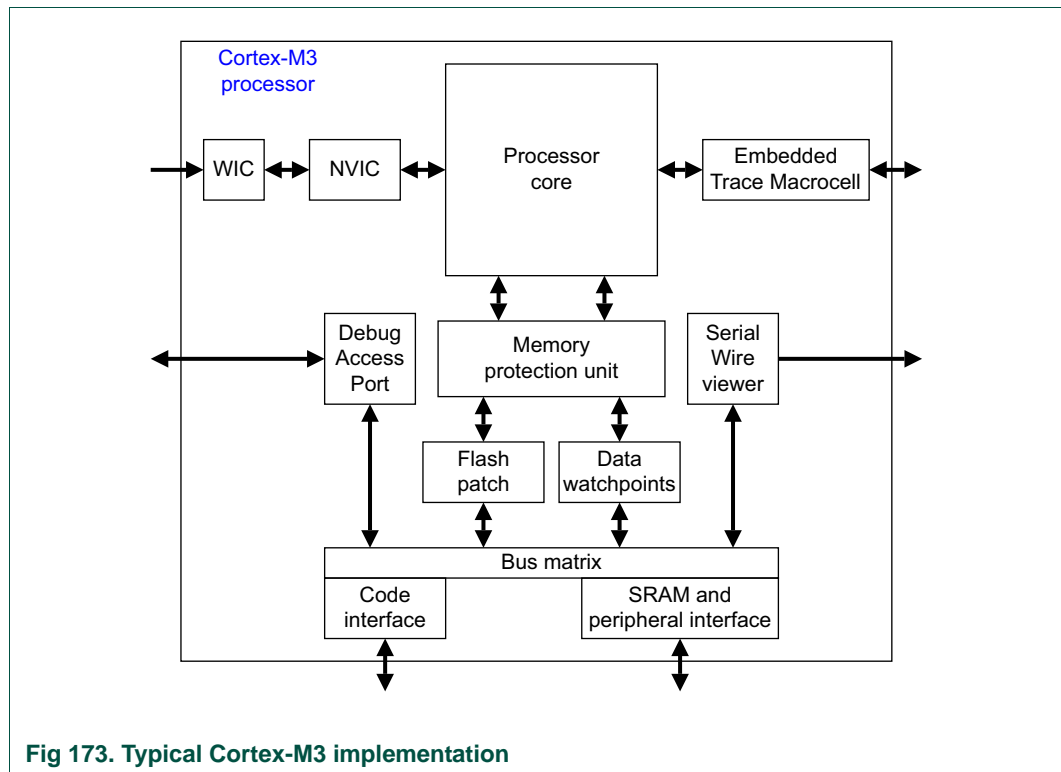


Fig 173. Typical Cortex-M3 implementation

The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including single-cycle 32x32 multiplication and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb instruction set, ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a configurable **nested interrupt controller** (NVIC), to deliver industry-leading interrupt performance. The NVIC includes a **non-maskable interrupt** (NMI), and provides up to 256 interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of **interrupt service routines** (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require any assembler stubs, removing any code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down.

LPC178x/177x devices support additional reduced power modes, see [Section 4.7 “Power control”](#) for details.

### 38.1.1.1 System level interface

The Cortex-M3 processor provides multiple interfaces using AMBA technology to provide high speed, low latency memory accesses. It supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks and thread-safe Boolean data handling.

The Cortex-M3 processor has an optional **memory protection unit** (MPU) that provides fine grain memory control, enabling applications to implement security privilege levels, separating code, data and stack on a task-by-task basis. Such requirements are becoming critical in many embedded applications such as automotive. The MPU is included in LPC178x/177x devices.

### 38.1.1.2 Integrated configurable debug

The Cortex-M3 processor implements a complete hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin **Serial Wire Debug** (SWD) port that is ideal for microcontrollers and other small package devices. The MCU vendor determines the debug feature configuration and therefore this can differ across different devices and families.

For system trace the processor integrates an **Instrumentation Trace Macrocell** (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system events these generate, a **Serial Wire Viewer** (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

The optional **Embedded Trace Macrocell** (ETM) delivers unrivalled instruction trace capture in an area far smaller than traditional trace units, enabling many low cost MCUs to implement full instruction trace for the first time.

LPC178x/177x devices support JTAG and Serial Wire Debug, Serial Wire Viewer, and include the Embedded Trace Macrocell. See [Section 37.1](#) for additional information.

### 38.1.1.3 Cortex-M3 processor features and benefits summary

- tight integration of system peripherals reduces area and development costs
- Thumb instruction set combines high code density with 32-bit performance
- code-patch ability for ROM system updates
- power control optimization of system components
- integrated sleep modes for low power consumption
- fast code execution permits slower processor clock or increases sleep mode time
- hardware division and fast multiplier
- deterministic, high-performance interrupt handling for time-critical applications
- optional **memory protection unit** (MPU) for safety-critical applications
- extensive debug and trace capabilities:
  - Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging and tracing.

### 38.1.1.4 Cortex-M3 core peripherals

These are:

- **Nested Vectored Interrupt Controller**

The **Nested Vectored Interrupt Controller** (NVIC) is an embedded interrupt controller that supports low latency interrupt processing.

- **System control block**

The **System control block** (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

- **System timer**

The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

- **Memory protection unit**

The **Memory protection unit** (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

## 38.2 ARM Cortex-M3 User Guide: Instruction Set

### 38.2.1 Instruction set summary

The processor implements a version of the Thumb instruction set. [Table 743](#) lists the supported instructions.

**Note**

In [Table 743](#):

- angle brackets, <>, enclose alternative forms of the operand
- braces, {}, enclose optional operands
- the Operands column is not exhaustive
- Op2 is a flexible second operand that can be either a register or a constant
- most instructions can use an optional condition code suffix.

For more information on the instructions and operands, see the instruction descriptions.

**Table 743. Cortex-M3 instructions**

Mnemonic	Operands	Brief description	Flags	Page
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
ADD, ADDW	{Rd,} Rn, #imm12	Add	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
ADR	Rd, label	Load PC-relative address	-	<a href="#">Section 38.2.4.1</a>
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C	<a href="#">Section 38.2.5.2</a>
ASR, ASRS	Rd, Rm, <Rs #n>	Arithmetic Shift Right	N,Z,C	<a href="#">Section 38.2.5.3</a>
B	label	Branch	-	<a href="#">Section 38.2.9.1</a>
BFC	Rd, #lsb, #width	Bit Field Clear	-	<a href="#">Section 38.2.8.1</a>
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	-	<a href="#">Section 38.2.8.1</a>
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N,Z,C	<a href="#">Section 38.2.5.2</a>
BKPT	#imm	Breakpoint	-	<a href="#">Section 38.2.10.1</a>
BL	label	Branch with Link	-	<a href="#">Section 38.2.9.1</a>
BLX	Rm	Branch indirect with Link	-	<a href="#">Section 38.2.9.1</a>
BX	Rm	Branch indirect	-	<a href="#">Section 38.2.9.1</a>
CBNZ	Rn, label	Compare and Branch if Non Zero	-	<a href="#">Section 38.2.9.2</a>
CBZ	Rn, label	Compare and Branch if Zero	-	<a href="#">Section 38.2.9.2</a>
CLREX	-	Clear Exclusive	-	<a href="#">Section 38.2.4.9</a>
CLZ	Rd, Rm	Count leading zeros	-	<a href="#">Section 38.2.5.4</a>
CMN, CMNS	Rn, Op2	Compare Negative	N,Z,C,V	<a href="#">Section 38.2.5.5</a>
CMP, CMPS	Rn, Op2	Compare	N,Z,C,V	<a href="#">Section 38.2.5.5</a>
CPSID	iflags	Change Processor State, Disable Interrupts	-	<a href="#">Section 38.2.10.2</a>
CPSIE	iflags	Change Processor State, Enable Interrupts	-	<a href="#">Section 38.2.10.2</a>
DMB	-	Data Memory Barrier	-	<a href="#">Section 38.2.10.3</a>
DSB	-	Data Synchronization Barrier	-	<a href="#">Section 38.2.10.4</a>
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C	<a href="#">Section 38.2.5.2</a>

Table 743. Cortex-M3 instructions ...continued

Mnemonic	Operands	Brief description	Flags	Page
ISB	-	Instruction Synchronization Barrier	-	<a href="#">Section 38.2.10.5</a>
IT	-	If-Then condition block	-	<a href="#">Section 38.2.9.3</a>
LDM	Rn{!}, reglist	Load Multiple registers, increment after	-	<a href="#">Section 38.2.4.6</a>
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	-	<a href="#">Section 38.2.4.6</a>
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	-	<a href="#">Section 38.2.4.6</a>
LDR	Rt, [Rn, #offset]	Load Register with word	-	<a href="#">Section 38.2.4</a>
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	-	<a href="#">Section 38.2.4</a>
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes	-	<a href="#">Section 38.2.4.2</a>
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	-	<a href="#">Section 38.2.4.8</a>
LDREXB	Rt, [Rn]	Load Register Exclusive with byte	-	<a href="#">Section 38.2.4.8</a>
LDREXH	Rt, [Rn]	Load Register Exclusive with halfword	-	<a href="#">Section 38.2.4.8</a>
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with halfword	-	<a href="#">Section 38.2.4</a>
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with signed byte	-	<a href="#">Section 38.2.4</a>
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with signed halfword	-	<a href="#">Section 38.2.4</a>
LDRT	Rt, [Rn, #offset]	Load Register with word	-	<a href="#">Section 38.2.4</a>
LSL, LSLS	Rd, Rm, <Rs #n>	Logical Shift Left	N,Z,C	<a href="#">Section 38.2.5.3</a>
LSR, LSRS	Rd, Rm, <Rs #n>	Logical Shift Right	N,Z,C	<a href="#">Section 38.2.5.3</a>
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	-	<a href="#">Section 38.2.6.1</a>
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	-	<a href="#">Section 38.2.6.1</a>
MOV, MOVS	Rd, Op2	Move	N,Z,C	<a href="#">Section 38.2.5.6</a>
MOVT	Rd, #imm16	Move Top	-	<a href="#">Section 38.2.5.7</a>
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C	<a href="#">Section 38.2.5.6</a>
MRS	Rd, spec_reg	Move from special register to general register	-	<a href="#">Section 38.2.10.6</a>
MSR	spec_reg, Rm	Move from general register to special register	N,Z,C,V	<a href="#">Section 38.2.10.7</a>
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z	<a href="#">Section 38.2.6.1</a>
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C	<a href="#">Section 38.2.5.6</a>
NOP	-	No Operation	-	<a href="#">Section 38.2.10.8</a>
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C	<a href="#">Section 38.2.5.2</a>
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C	<a href="#">Section 38.2.5.2</a>
POP	reglist	Pop registers from stack	-	<a href="#">Section 38.2.4.7</a>
PUSH	reglist	Push registers onto stack	-	<a href="#">Section 38.2.4.7</a>
RBIT	Rd, Rn	Reverse Bits	-	<a href="#">Section 38.2.5.8</a>
REV	Rd, Rn	Reverse byte order in a word	-	<a href="#">Section 38.2.5.8</a>
REV16	Rd, Rn	Reverse byte order in each halfword	-	<a href="#">Section 38.2.5.8</a>
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	-	<a href="#">Section 38.2.5.8</a>
ROR, RORS	Rd, Rm, <Rs #n>	Rotate Right	N,Z,C	<a href="#">Section 38.2.5.3</a>
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C	<a href="#">Section 38.2.5.3</a>
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	-	<a href="#">Section 38.2.8.2</a>

Table 743. Cortex-M3 instructions ...continued

Mnemonic	Operands	Brief description	Flags	Page
SDIV	{Rd,} Rn, Rm	Signed Divide	-	<a href="#">Section 38.2.6.3</a>
SEV	-	Send Event	-	<a href="#">Section 38.2.10.9</a>
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	-	<a href="#">Section 38.2.6.2</a>
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	-	<a href="#">Section 38.2.6.2</a>
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q	<a href="#">Section 38.2.7.1</a>
STM	Rn{!}, reglist	Store Multiple registers, increment after	-	<a href="#">Section 38.2.4.6</a>
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	-	<a href="#">Section 38.2.4.6</a>
STMPD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	-	<a href="#">Section 38.2.4.6</a>
STR	Rt, [Rn, #offset]	Store Register word	-	<a href="#">Section 38.2.4</a>
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	-	<a href="#">Section 38.2.4</a>
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	-	<a href="#">Section 38.2.4.2</a>
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	-	<a href="#">Section 38.2.4.8</a>
STREXB	Rd, Rt, [Rn]	Store Register Exclusive byte	-	<a href="#">Section 38.2.4.8</a>
STREXH	Rd, Rt, [Rn]	Store Register Exclusive halfword	-	<a href="#">Section 38.2.4.8</a>
STRH, STRHT	Rt, [Rn, #offset]	Store Register halfword	-	<a href="#">Section 38.2.4</a>
STRT	Rt, [Rn, #offset]	Store Register word	-	<a href="#">Section 38.2.4</a>
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V	<a href="#">Section 38.2.5.1</a>
SVC	#imm	Supervisor Call	-	<a href="#">Section 38.2.10.10</a> <a href="#">0</a>
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	-	<a href="#">Section 38.2.8.3</a>
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	-	<a href="#">Section 38.2.8.3</a>
TBB	[Rn, Rm]	Table Branch Byte	-	<a href="#">Section 38.2.9.4</a>
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword	-	<a href="#">Section 38.2.9.4</a>
TEQ	Rn, Op2	Test Equivalence	N,Z,C	<a href="#">Section 38.2.5.9</a>
TST	Rn, Op2	Test	N,Z,C	<a href="#">Section 38.2.5.9</a>
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	-	<a href="#">Section 38.2.8.2</a>
UDIV	{Rd,} Rn, Rm	Unsigned Divide	-	<a href="#">Section 38.2.6.3</a>
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	-	<a href="#">Section 38.2.6.2</a>
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	-	<a href="#">Section 38.2.6.2</a>
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q	<a href="#">Section 38.2.7.1</a>
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a byte	-	<a href="#">Section 38.2.8.3</a>
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a halfword	-	<a href="#">Section 38.2.8.3</a>
WFE	-	Wait For Event	-	<a href="#">Section 38.2.10.1</a> <a href="#">1</a>
WFI	-	Wait For Interrupt	-	<a href="#">Section 38.2.10.1</a> <a href="#">2</a>

### 38.2.2 Intrinsic functions

ANSI cannot directly access some Cortex-M3 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ANSI cannot directly access:

**Table 744. CMSIS intrinsic functions to generate some Cortex-M3 instructions**

Instruction	CMSIS intrinsic function
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

**Table 745. CMSIS intrinsic functions to access the special registers**

Special register	Access	CMSIS function
PRIMASK	Read	uint32_t __get_PRIMASK (void)
	Write	void __set_PRIMASK (uint32_t value)
FAULTMASK	Read	uint32_t __get_FAULTMASK (void)
	Write	void __set_FAULTMASK (uint32_t value)
BASEPRI	Read	uint32_t __get_BASEPRI (void)
	Write	void __set_BASEPRI (uint32_t value)
CONTROL	Read	uint32_t __get_CONTROL (void)
	Write	void __set_CONTROL (uint32_t value)
MSP	Read	uint32_t __get_MSP (void)
	Write	void __set_MSP (uint32_t TopOfMainStack)
PSP	Read	uint32_t __get_PSP (void)
	Write	void __set_PSP (uint32_t TopOfProcStack)

### 38.2.3 About the instruction descriptions

The following sections give more information about using the instructions:

- [Section 38.2.3.1 “Operands”](#)
- [Section 38.2.3.2 “Restrictions when using PC or SP”](#)
- [Section 38.2.3.3 “Flexible second operand”](#)
- [Section 38.2.3.4 “Shift Operations”](#)
- [Section 38.2.3.5 “Address alignment”](#)
- [Section 38.2.3.6 “PC-relative expressions”](#)
- [Section 38.2.3.7 “Conditional execution”](#)
- [Section 38.2.3.8 “Instruction width selection”](#).

### 38.2.3.1 Operands

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the operands.

Operands in some instructions are flexible in that they can either be a register or a constant. See [Section 38.2.3.3](#).

### 38.2.3.2 Restrictions when using PC or SP

Many instructions have restrictions on whether you can use the **Program Counter** (PC) or **Stack Pointer** (SP) for the operands or destination register. See instruction descriptions for more information.

**Remark:** Bit[0] of any address you write to the PC with a `BX`, `BLX`, `LDM`, `LDR`, or `POP` instruction must be 1 for correct execution, because this bit indicates the required instruction set, and the Cortex-M3 processor only supports Thumb instructions.

### 38.2.3.3 Flexible second operand

Many general data processing instructions have a flexible second operand. This is shown as *Operand2* in the descriptions of the syntax of each instruction.

*Operand2* can be a:

- [Section 38.2.3.3.1 “Constant”](#)
- [Section 38.2.3.3.2 “Register with optional shift”](#)

#### 38.2.3.3.1 Constant

You specify an *Operand2* constant in the form:

`#constant`

where `constant` can be:

- any constant that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word
- any constant of the form `0x00XY00XY`
- any constant of the form `0xXY00XY00`
- any constant of the form `0xXYXYXYXY`.



**Remark:** In the constants shown above, *x* and *y* are hexadecimal digits.

In addition, in a small number of instructions, `constant` can take a wider range of values. These are described in the individual instruction descriptions.

When an Operand2 constant is used with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to bit[31] of the constant, if the constant is greater than 255 and can be produced by shifting an 8-bit value. These instructions do not affect the carry flag if Operand2 is any other constant.

**Instruction substitution:** Your assembler might be able to produce an equivalent instruction in cases where you specify a constant that is not permitted. For example, an assembler might assemble the instruction `CMP Rd, #0xFFFFFFFF` as the equivalent instruction `CMN Rd, #0x2`.

### 38.2.3.3.2 Register with optional shift

You specify an Operand2 register in the form:

*Rm* {, *shift*}

where:

*Rm* is the register holding the data for the second operand.

*shift* is an optional shift to be applied to *Rm*. It can be one of:

`ASR#n`: arithmetic shift right *n* bits,  $1 \leq n \leq 32$ .

`LSL#n`: logical shift left *n* bits,  $1 \leq n \leq 31$ .

`LSR#n`: logical shift right *n* bits,  $1 \leq n \leq 32$ .

`ROR#n`: rotate right *n* bits,  $1 \leq n \leq 31$ .

`RRX`: rotate right one bit, with extend.

—: if omitted, no shift occurs, equivalent to `LSL#0`.

If you omit the shift, or specify `LSL #0`, the instruction uses the value in *Rm*.

If you specify a shift, the shift is applied to the value in *Rm*, and the resulting 32-bit value is used by the instruction. However, the contents in the register *Rm* remains unchanged. Specifying a register with shift also updates the carry flag when used with certain instructions. For information on the shift operations and how they affect the carry flag, see [Section 38.2.3.4 “Shift Operations”](#)

### 38.2.3.4 Shift Operations

Register shift operations move the bits in a register left or right by a specified number of bits, the **shift length**. Register shift can be performed:

- directly by the instructions `ASR`, `LSR`, `LSL`, `ROR`, and `RRX`, and the result is written to a destination register
- during the calculation of *Operand2* by the instructions that specify the second operand as a register with shift, see [Section 38.2.3.3](#). The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description or [Section 38.2.3.3](#). If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions, *Rm* is the register containing the value to be shifted, and *n* is the shift length.

**38.2.3.4.1 ASR**

Arithmetic shift right by *n* bits moves the left-hand  $32-n$  bits of the register *Rm*, to the right by *n* places, into the right-hand  $32-n$  bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See [Figure 174](#).

You can use the ASR *#n* operation to divide the value in the register *Rm* by  $2^n$ , with the result being rounded towards negative-infinity.

When the instruction is ASRS or when ASR *#n* is used in *Operand2* with the instructions MOVs, MVNS, ANDs, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

**Note**

- If *n* is 32 or more, then all the bits in the result are set to the value of bit[31] of *Rm*.
- If *n* is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of *Rm*.

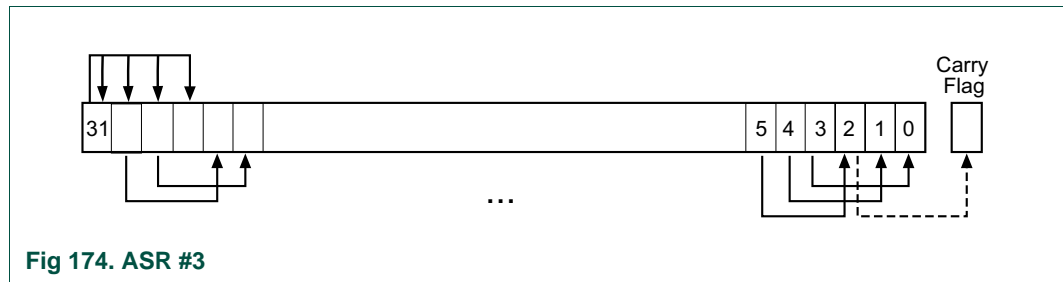


Fig 174. ASR #3

**38.2.3.4.2 LSR**

Logical shift right by *n* bits moves the left-hand  $32-n$  bits of the register *Rm*, to the right by *n* places, into the right-hand  $32-n$  bits of the result. And it sets the left-hand *n* bits of the result to 0. See [Figure 175](#).

You can use the LSR *#n* operation to divide the value in the register *Rm* by  $2^n$ , if the value is regarded as an unsigned integer.

When the instruction is LSRS or when LSR *#n* is used in *Operand2* with the instructions MOVs, MVNS, ANDs, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

**Note**

- If *n* is 32 or more, then all the bits in the result are cleared to 0.

- If  $n$  is 33 or more and the carry flag is updated, it is updated to 0.

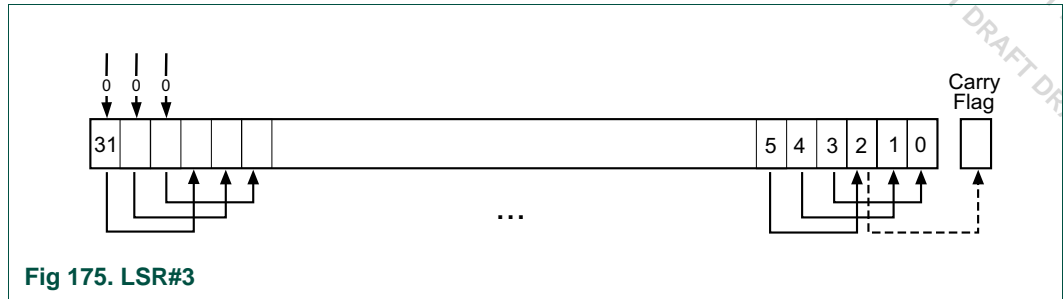


Fig 175. LSR#3

38.2.3.4.3 LSL

Logical shift left by  $n$  bits moves the right-hand  $32-n$  bits of the register  $Rm$ , to the left by  $n$  places, into the left-hand  $32-n$  bits of the result. And it sets the right-hand  $n$  bits of the result to 0. See [Figure 176](#).

You can use the LSL  $\#n$  operation to multiply the value in the register  $Rm$  by  $2^n$ , if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is `LSLS` or when `LSL #n`, with non-zero  $n$ , is used in *Operand2* with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to the last bit shifted out, `bit[32-n]`, of the register  $Rm$ . These instructions do not affect the carry flag when used with `LSL #0`.

Note

- If  $n$  is 32 or more, then all the bits in the result are cleared to 0.
- If  $n$  is 33 or more and the carry flag is updated, it is updated to 0.

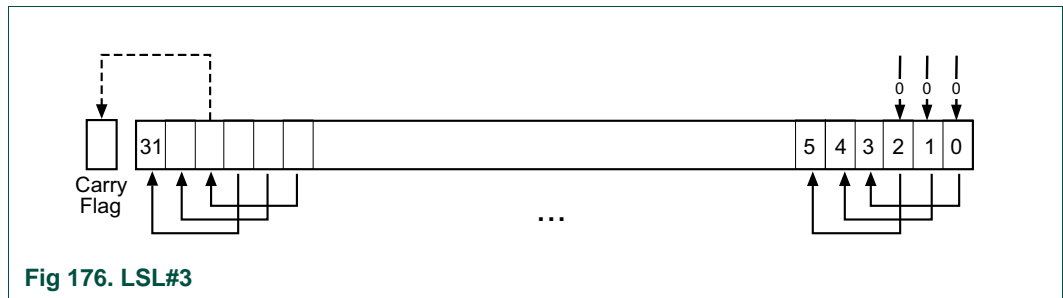


Fig 176. LSL#3

38.2.3.4.4 ROR

Rotate right by  $n$  bits moves the left-hand  $32-n$  bits of the register  $Rm$ , to the right by  $n$  places, into the right-hand  $32-n$  bits of the result. And it moves the right-hand  $n$  bits of the register into the left-hand  $n$  bits of the result. See [Figure 177](#).

When the instruction is `RORS` or when `ROR #n` is used in *Operand2* with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to the last bit rotation, `bit[n-1]`, of the register  $Rm$ .

Note

- If  $n$  is 32, then the value of the result is same as the value in  $Rm$ , and if the carry flag is updated, it is updated to bit[31] of  $Rm$ .
- ROR with shift length,  $n$ , more than 32 is the same as ROR with shift length  $n-32$ .

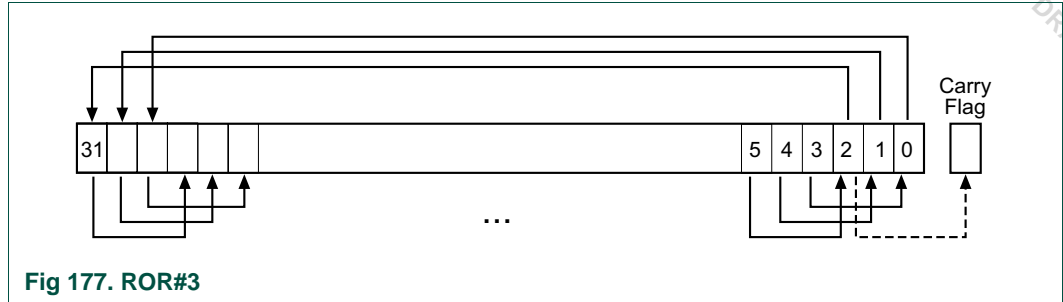


Fig 177. ROR#3

38.2.3.4.5 RRX

Rotate right with extend moves the bits of the register  $Rm$  to the right by one bit. And it copies the carry flag into bit[31] of the result. See [Figure 178](#).

When the instruction is `RRXS` or when `RRX` is used in *Operand2* with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to bit[0] of the register  $Rm$ .

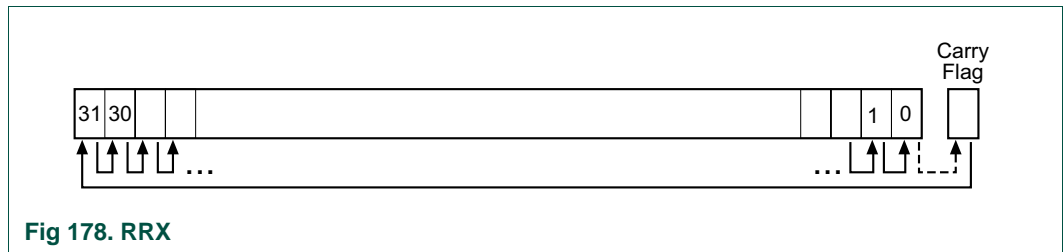


Fig 178. RRX

38.2.3.5 Address alignment

An aligned access is an operation where a word-aligned address is used for a word, dual word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

The Cortex-M3 processor supports unaligned access only for the following instructions:

- LDR, LDRT
- LDRH, LDRHT
- LDRSH, LDRSHT
- STR, STRT
- STRH, STRHT

All other load and store instructions generate a usage fault exception if they perform an unaligned access, and therefore their accesses must be address aligned. For more information about usage faults see [Section 38.3.4 “Fault handling”](#).

Unaligned accesses are usually slower than aligned accesses. In addition, some memory regions might not support unaligned accesses. Therefore, ARM recommends that programmers ensure that accesses are aligned. To avoid accidental generation of unaligned accesses, use the UNALIGN\_TRP bit in the Configuration and Control Register to trap all unaligned accesses, see [Section 38.4.3.8 “Configuration and Control Register”](#).

### 38.2.3.6 PC-relative expressions

A PC-relative expression or **label** is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset. The assembler calculates the required offset from the label and the address of the current instruction. If the offset is too big, the assembler produces an error.

#### Note

- For B, BL, CBNZ, and CBZ instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- For all other instructions that use labels, the value of the PC is the address of the current instruction plus 4 bytes, with bit[1] of the result cleared to 0 to make it word-aligned.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form [PC, #number].

### 38.2.3.7 Conditional execution

Most data processing instructions can optionally update the condition flags in the **Application Program Status Register (APSR)** according to the result of the operation, see [Section 38.3.1.3.5 “Program Status Register”](#). Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

You can execute an instruction conditionally, based on the condition flags set in another instruction, either:

- immediately after the instruction that updated the flags
- after any number of intervening instructions that have not updated the flags.

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See [Table 746](#) for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

- does not execute
- does not write any value to its destination register
- does not affect any of the flags
- does not generate any exception.

Conditional instructions, except for conditional branches, must be inside an If-Then instruction block. See [Section 38.2.9.3](#) for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if you have conditional instructions outside the IT block.

Use the `CBZ` and `CBNZ` instructions to compare the value of a register against zero and branch on the result.

This section describes:

- [Section 38.2.3.7.1 “The condition flags”](#)
- [Section 38.2.3.7.2 “Condition code suffixes”](#).

### 38.2.3.7.1 The condition flags

The APSR contains the following condition flags:

**N** — Set to 1 when the result of the operation was negative, cleared to 0 otherwise.

**Z** — Set to 1 when the result of the operation was zero, cleared to 0 otherwise.

**C** — Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.

**V** — Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR see [Section 38.3.1.3.5 “Program Status Register”](#).

A carry occurs:

- if the result of an addition is greater than or equal to  $2^{32}$
- if the result of a subtraction is positive or zero
- as the result of an inline barrel shifter operation in a move or logical instruction.

Overflow occurs if the result of an add, subtract, or compare is greater than or equal to  $2^{31}$ , or less than  $-2^{31}$ .

**Remark:** Most instructions update the status flags only if the `S` suffix is specified. See the instruction descriptions for more information.

### 38.2.3.7.2 Condition code suffixes

The instructions that can be conditional have an optional condition code, shown in syntax descriptions as `{cond}`. Conditional execution requires a preceding `IT` instruction. An instruction with a condition code is only executed if the condition code flags in the APSR meet the specified condition. [Table 746](#) shows the condition codes to use.

You can use conditional execution with the `IT` instruction to reduce the number of branch instructions in code.

[Table 746](#) also shows the relationship between condition code suffixes and the `N`, `Z`, `C`, and `V` flags.

**Table 746. Condition code suffixes**

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned $\geq$
CC or LO	C = 0	Lower, unsigned $<$
MI	N = 1	Negative

Table 746. Condition code suffixes

Suffix	Flags	Meaning
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

[Section](#) shows the use of a conditional instruction to find the absolute value of a number. `R0 = ABS(R1)`.

[Section](#) shows the use of conditional instructions to update the value of R4 if the signed values R0 is greater than R1 and R2 is greater than R3.

**Example: Absolute value:**

```

MOVSBMI R0, R1, #0 ; R0 = R1, setting flags
IT MI ; IT instruction for the negative condition

```

**Example: Compare and update value:**

```

CMP R0, R1 ; Compare R0 and R1,
setting flags
ITT GT ; IT instruction for the two GT conditions
CMPGT R2, R3 ; If 'greater than', compare R2 and R3, setting flags
MOVGT R4, R5 ; If still 'greater than', do R4 = R5

```

### 38.2.3.8 Instruction width selection

There are many instructions that can generate either a 16-bit encoding or a 32-bit encoding depending on the operands and destination register specified. For some of these instructions, you can force a specific instruction size by using an instruction width suffix. The `.W` suffix forces a 32-bit instruction encoding. The `.N` suffix forces a 16-bit instruction encoding.

If you specify an instruction width suffix and the assembler cannot generate an instruction encoding of the requested width, it generates an error.

**Remark:** In some cases it might be necessary to specify the `.W` suffix, for example if the operand is the label of an instruction or literal data, as in the case of branch instructions. This is because the assembler might not automatically generate the right size encoding.

To use an instruction width suffix, place it immediately after the instruction mnemonic and condition code, if any. [Section 38.2.3.8.1](#) shows instructions with the instruction width suffix.

### 38.2.3.8.1 Example: Instruction width selection

`BCS.W label` ; creates a 32-bit instruction even for a short branch

`ADDS.W R0, R0, R1` ; creates a 32-bit instruction even though the same  
; operation can be done by a 16-bit instruction



### 38.2.4 Memory access instructions

[Table 747](#) shows the memory access instructions:

**Table 747. Memory access instructions**

Mnemonic	Brief description	See
ADR	Load PC-relative address	<a href="#">Section 38.2.4.1</a>
CLREX	Clear Exclusive	<a href="#">Section 38.2.4.9</a>
LDM{mode}	Load Multiple registers	<a href="#">Section 38.2.4.6</a>
LDR{type}	Load Register using immediate offset	<a href="#">Section 38.2.4.2</a>
LDR{type}	Load Register using register offset	<a href="#">Section 38.2.4.3</a>
LDR{type}T	Load Register with unprivileged access	<a href="#">Section 38.2.4.4</a>
LDR	Load Register using PC-relative address	<a href="#">Section 38.2.4.5</a>
LDREX{type}	Load Register Exclusive	<a href="#">Section 38.2.4.8</a>
POP	Pop registers from stack	<a href="#">Section 38.2.4.7</a>
PUSH	Push registers onto stack	<a href="#">Section 38.2.4.7</a>
STM{mode}	Store Multiple registers	<a href="#">Section 38.2.4.6</a>
STR{type}	Store Register using immediate offset	<a href="#">Section 38.2.4.2</a>
STR{type}	Store Register using register offset	<a href="#">Section 38.2.4.3</a>
STR{type}T	Store Register with unprivileged access	<a href="#">Section 38.2.4.4</a>
STREX{type}	Store Register Exclusive	<a href="#">Section 38.2.4.8</a>

### 38.2.4.1 ADR

Load PC-relative address.

#### 38.2.4.1.1 Syntax

`ADR{cond} Rd, label`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*label* is a PC-relative expression. See [Section 38.2.3.6 “PC-relative expressions”](#).

#### 38.2.4.1.2 Operation

ADR determines the address by adding an immediate value to the PC, and writes the result to the destination register.

ADR produces position-independent code, because the address is PC-relative.

If you use ADR to generate a target address for a BX or BLX instruction, you must ensure that bit[0] of the address you generate is set to 1 for correct execution.

Values of *label* must be within the range of –4095 to +4095 from the address in the PC.

**Remark:** You might have to use the `.w` suffix to get the maximum offset range or to generate addresses that are not word-aligned. See [Section 38.2.3.8 “Instruction width selection”](#).

#### 38.2.4.1.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 38.2.4.1.4 Condition flags

This instruction does not change the flags.

#### 38.2.4.1.5 Examples

```
ADR    R1, TextMessage    ; Write address value of a location labelled as  
                        ; TextMessage to R1
```

### 38.2.4.2 LDR and STR, immediate offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

#### 38.2.4.2.1 Syntax

$op\{type\}\{cond\} Rt, [Rn \{, \#offset\}]$  ; immediate offset

$op\{type\}\{cond\} Rt, [Rn, \#offset]!$  ; pre-indexed

$op\{type\}\{cond\} Rt, [Rn], \#offset$  ; post-indexed

$opD\{cond\} Rt, Rt2, [Rn \{, \#offset\}]$  ; immediate offset, two words

$opD\{cond\} Rt, Rt2, [Rn, \#offset]!$  ; pre-indexed, two words

$opD\{cond\} Rt, Rt2, [Rn], \#offset$  ; post-indexed, two words

where:

$op$  is one of:

LDR: Load register.  
STR: Store register.

$type$  is one of:

B: unsigned byte, zero extend to 32 bits on loads.  
SB: signed byte, sign extend to 32 bits (LDR only).  
H: unsigned halfword, zero extend to 32 bits on loads.  
SH: signed halfword, sign extend to 32 bits (LDR only).  
—: omit, for word.

$cond$  is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

$Rt$  is the register to load or store.

$Rn$  is the register on which the memory address is based.

$offset$  is an offset from  $Rn$ . If  $offset$  is omitted, the address is the contents of  $Rn$ .

$Rt2$  is the additional register to load or store for two-word operations.

#### 38.2.4.2.2 Operation

LDR instructions load one or two registers with a value from memory.

STR instructions store one or two register values to memory.

Load and store instructions with immediate offset can use the following addressing modes:

- Offset addressing

The offset value is added to or subtracted from the address obtained from the register  $Rn$ . The result is used as the address for the memory access. The register  $Rn$  is unaltered. The assembly language syntax for this mode is:

[Rn, #offset]

- Pre-indexed addressing

The offset value is added to or subtracted from the address obtained from the register *Rn*. The result is used as the address for the memory access and written back into the register *Rn*. The assembly language syntax for this mode is:

[Rn, #offset]!

- Post-indexed addressing

The address obtained from the register *Rn* is used as the address for the memory access. The offset value is added to or subtracted from the address, and written back into the register *Rn*. The assembly language syntax for this mode is:

[Rn], #offset

The value to load or store can be a byte, halfword, word, or two words. Bytes and halfwords can either be signed or unsigned. See [Section 38.2.3.5 “Address alignment”](#).

[Table 748](#) shows the ranges of offset for immediate, pre-indexed and post-indexed forms.

**Table 748. Offset ranges**

Instruction type	Immediate offset	Pre-indexed	Post-indexed
Word, halfword, signed halfword, byte, or signed byte	-255 to 4095	-255 to 255	-255 to 255
Two words	multiple of 4 in the range -1020 to 1020	multiple of 4 in the range -1020 to 1020	multiple of 4 in the range -1020 to 1020

### 38.2.4.2.3 Restrictions

For load instructions:

- *Rt* can be SP or PC for word loads only
- *Rt* must be different from *Rt2* for two-word loads
- *Rn* must be different from *Rt* and *Rt2* in the pre-indexed or post-indexed forms.

When *Rt* is PC in a word load instruction:

- bit[0] of the loaded value must be 1 for correct execution
- a branch occurs to the address created by changing bit[0] of the loaded value to 0
- if the instruction is conditional, it must be the last instruction in the IT block.

For store instructions:

- *Rt* can be SP for word stores only
- *Rt* must not be PC
- *Rn* must not be PC
- *Rn* must be different from *Rt* and *Rt2* in the pre-indexed or post-indexed forms.

### 38.2.4.2.4 Condition flags

These instructions do not change the flags.

38.2.4.2.5 Examples

```

LDR    R8, [R10]           ; Loads R8 from the address in R10.
LDRNE  R2, [R5, #960]!    ; Loads (conditionally) R2 from a word
                           ; 960 bytes above the address in R5, and
                           ; increments R5 by 960.

STR    R2, [R9, #const-struct] ; const-struct is an expression evaluating
                           ; to a constant in the range 0-4095.

STRH   R3, [R4], #4       ; Store R3 as halfword data into address in
                           ; R4, then increment R4 by 4

LDRD   R8, R9, [R3, #0x20] ; Load R8 from a word 32 bytes above the
                           ; address in R3, and load R9 from a word 36
                           ; bytes above the address in R3

STRD   R0, R1, [R8], #-16  ; Store R0 to address in R8, and store R1 to
                           ; a word 4 bytes above the address in R8,
                           ; and then decrement R8 by 16.
    
```

### 38.2.4.3 LDR and STR, register offset

Load and Store with register offset.

#### 38.2.4.3.1 Syntax

$op\{type\}\{cond\} Rt, [Rn, Rm \{, LSL \#n\}]$

where:

$op$  is one of:

LDR: Load Register.

STR: Store Register.

$type$  is one of:

B: unsigned byte, zero extend to 32 bits on loads.

SB: signed byte, sign extend to 32 bits (LDR only).

H: unsigned halfword, zero extend to 32 bits on loads.

SH: signed halfword, sign extend to 32 bits (LDR only).

—: omit, for word.

$cond$  is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

$Rt$  is the register to load or store.

$Rn$  is the register on which the memory address is based.

$Rm$  is a register containing a value to be used as the offset.

LSL  $\#n$  is an optional shift, with  $n$  in the range 0 to 3.

#### 38.2.4.3.2 Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

The memory address to load from or store to is at an offset from the register  $Rn$ . The offset is specified by the register  $Rm$  and can be shifted left by up to 3 bits using LSL.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See [Section 38.2.3.5 “Address alignment”](#).

#### 38.2.4.3.3 Restrictions

In these instructions:

- $Rn$  must not be PC
- $Rm$  must not be SP and must not be PC
- $Rt$  can be SP only for word loads and word stores
- $Rt$  can be PC only for word loads.

When  $Rt$  is PC in a word load instruction:

- bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the IT block.

#### 38.2.4.3.4 Condition flags

These instructions do not change the flags.

#### 38.2.4.3.5 Examples

```
STR    R0, [R5, R1]           ; Store value of R0 into an address equal to
                                ; sum of R5 and R1
LDRSB  R0, [R5, R1, LSL #1]   ; Read byte value from an address equal to
                                ; sum of R5 and two times R1, sign extended it
                                ; to a word value and put it in R0
STR    R0, [R1, R2, LSL #2]   ; Stores R0 to an address equal to sum of R1
                                ; and four times R2
```

#### 38.2.4.4 LDR and STR, unprivileged

Load and Store with unprivileged access.

##### 38.2.4.4.1 Syntax

$op\{type\}T\{cond\} Rt, [Rn \{, \#offset\}]$  ; immediate offset

where:

*op* is one of:

LDR: Load Register.

STR: Store Register.

*type* is one of:

B: unsigned byte, zero extend to 32 bits on loads.

SB: signed byte, sign extend to 32 bits (LDR only).

H: unsigned halfword, zero extend to 32 bits on loads.

SH: signed halfword, sign extend to 32 bits (LDR only).

—: omit, for word.

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rt* is the register to load or store.

*Rn* is the register on which the memory address is based.

*offset* is an offset from *Rn* and can be 0 to 255. If *offset* is omitted, the address is the value in *Rn*.

##### 38.2.4.4.2 Operation

These load and store instructions perform the same function as the memory access instructions with immediate offset, see [Section 38.2.4.2](#). The difference is that these instructions have only unprivileged access even when used in privileged software.

When used in unprivileged software, these instructions behave in exactly the same way as normal memory access instructions with immediate offset.

##### 38.2.4.4.3 Restrictions

In these instructions:

- *Rn* must not be PC
- *Rt* must not be SP and must not be PC.

##### 38.2.4.4.4 Condition flags

These instructions do not change the flags.



#### 38.2.4.4.5 Examples

```
STRBTEQ R4, [R7] ; Conditionally store least significant byte in
                  ; R4 to an address in R7, with unprivileged access
LDRHT R2, [R2, #8] ; Load halfword value from an address equal to
                  ; sum of R2 and 8 into R2, with unprivileged access
```

**38.2.4.5 LDR, PC-relative**

Load register from memory.

**38.2.4.5.1 Syntax**

LDR{*type*}{*cond*} *Rt*, *label*

LDRD{*cond*} *Rt*, *Rt2*, *label* ; Load two words

*type* is one of:

- B: unsigned byte, zero extend to 32 bits on loads.
- SB: signed byte, sign extend to 32 bits (LDR only).
- H: unsigned halfword, zero extend to 32 bits on loads.
- SH: signed halfword, sign extend to 32 bits (LDR only).
- : omit, for word.

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rt* is the register to load or store.

*Rt2* is the second register to load or store.

*label* is a PC-relative expression. See [Section 38.2.3.6 “PC-relative expressions”](#).

**38.2.4.5.2 Operation**

LDR loads a register with a value from a PC-relative memory address. The memory address is specified by a label or by an offset from the PC.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See [Section 38.2.3.5 “Address alignment”](#).

*label* must be within a limited range of the current instruction. [Table 749](#) shows the possible offsets between *label* and the PC.

**Table 749. Offset ranges**

Instruction type	Offset range
Word, halfword, signed halfword, byte, signed byte	–4095 to 4095
Two words	–1020 to 1020

**Remark:** You might have to use the .W suffix to get the maximum offset range. See [Section 38.2.3.8 “Instruction width selection”](#).

**38.2.4.5.3 Restrictions**

In these instructions:

- *Rt* can be SP or PC only for word loads
- *Rt2* must not be SP and must not be PC
- *Rt* must be different from *Rt2*.

When *Rt* is PC in a word load instruction:

- bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the IT block.

#### 38.2.4.5.4 Condition flags

These instructions do not change the flags.

#### 38.2.4.5.5 Examples

```
LDR    R0, LookUpTable ; Load R0 with a word of data from an address
                          ; labelled as LookUpTable
LDRSB  R7, localdata   ; Load a byte value from an address labelled
                          ; as localdata, sign extend it to a word
                          ; value, and put it in R7
```

### 38.2.4.6 LDM and STM

Load and Store Multiple registers.

#### 38.2.4.6.1 Syntax

*op*{*addr\_mode*}{*cond*} *Rn*{!}, *reglist*

where:

*op* is one of:

LDM: Load Multiple registers.

STM: Store Multiple registers.

*addr\_mode* is any one of the following:

IA: Increment address After each access. This is the default.

DB: Decrement address Before each access.

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rn* is the register on which the memory addresses are based.

! is an optional writeback suffix. If ! is present the final address, that is loaded from or stored to, is written back into *Rn*.

*reglist* is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range, see [Section 38.2.4.6.5](#).

LDM and LDMFD are synonyms for LDMIA. LDMFD refers to its use for popping data from Full Descending stacks.

LDMEA is a synonym for LDMDB, and refers to its use for popping data from Empty Ascending stacks.

STM and STMEA are synonyms for STMIA. STMEA refers to its use for pushing data onto Empty Ascending stacks.

STMFD is a synonym for STMDB, and refers to its use for pushing data onto Full Descending stacks.

#### 38.2.4.6.2 Operation

LDM instructions load the registers in *reglist* with word values from memory addresses based on *Rn*.

STM instructions store the word values in the registers in *reglist* to memory addresses based on *Rn*.

For LDM, LDMIA, LDMFD, STM, STMIA, and STMEA the memory addresses used for the accesses are at 4-byte intervals ranging from  $Rn$  to  $Rn + 4 * (n-1)$ , where  $n$  is the number of registers in *reglist*. The accesses happens in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address. If the writeback suffix is specified, the value of  $Rn + 4 * (n-1)$  is written back to *Rn*.

For `LDMDB`, `LDMEA`, `STMDB`, and `STMF` the memory addresses used for the accesses are at 4-byte intervals ranging from  $Rn$  to  $Rn - 4 * (n-1)$ , where  $n$  is the number of registers in *reglist*. The accesses happen in order of decreasing register numbers, with the highest numbered register using the highest memory address and the lowest number register using the lowest memory address. If the writeback suffix is specified, the value of  $Rn - 4 * (n-1)$  is written back to  $Rn$ .

The `PUSH` and `POP` instructions can be expressed in this form. See [Section 38.2.4.7](#) for details.

### 38.2.4.6.3 Restrictions

In these instructions:

- $Rn$  must not be PC
- *reglist* must not contain SP
- in any `STM` instruction, *reglist* must not contain PC
- in any `LDM` instruction, *reglist* must not contain PC if it contains LR
- *reglist* must not contain  $Rn$  if you specify the writeback suffix.

When PC is in *reglist* in an `LDM` instruction:

- bit[0] of the value loaded to the PC must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the IT block.

### 38.2.4.6.4 Condition flags

These instructions do not change the flags.

### 38.2.4.6.5 Examples

```
LDM    R8, {R0,R2,R9}    ; LDMIA is a synonym for LDM
STMDB  R1!, {R3-R6,R11,R12}
```

### 38.2.4.6.6 Incorrect examples

```
STM    R5!, {R5,R4,R9} ; Value stored for R5 is unpredictable
LDM    R2, {}          ; There must be at least one register in the list
```

### 38.2.4.7 PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

#### 38.2.4.7.1 Syntax

`PUSH{cond} reglist`

`POP{cond} reglist`

where:

`cond` is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

`reglist` is a non-empty list of registers, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

`PUSH` and `POP` are synonyms for `STMDB` and `LDM` (or `LDMIA`) with the memory addresses for the access based on `SP`, and with the final address for the access written back to the `SP`. `PUSH` and `POP` are the preferred mnemonics in these cases.

#### 38.2.4.7.2 Operation

`PUSH` stores registers on the stack in order of decreasing the register numbers, with the highest numbered register using the highest memory address and the lowest numbered register using the lowest memory address.

`POP` loads registers from the stack in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

See [Section 38.2.4.6](#) for more information.

#### 38.2.4.7.3 Restrictions

In these instructions:

- `reglist` must not contain `SP`
- for the `PUSH` instruction, `reglist` must not contain `PC`
- for the `POP` instruction, `reglist` must not contain `PC` if it contains `LR`.

When `PC` is in `reglist` in a `POP` instruction:

- bit[0] of the value loaded to the `PC` must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the `IT` block.

#### 38.2.4.7.4 Condition flags

These instructions do not change the flags.

#### 38.2.4.7.5 Examples

```

PUSH    {R0, R4-R7}
PUSH    {R2, LR}
POP     {R0, R10, PC}

```

### 38.2.4.8 LDREX and STREX

Load and Store Register Exclusive.

#### 38.2.4.8.1 Syntax

LDREX{*cond*} *Rt*, [*Rn* {, #*offset*}]

STREX{*cond*} *Rd*, *Rt*, [*Rn* {, #*offset*}]

LDREXB{*cond*} *Rt*, [*Rn*]

STREXB{*cond*} *Rd*, *Rt*, [*Rn*]

LDREXH{*cond*} *Rt*, [*Rn*]

STREXH{*cond*} *Rd*, *Rt*, [*Rn*]

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register for the returned status.

*Rt* is the register to load or store.

*Rn* is the register on which the memory address is based.

*offset* is an optional offset applied to the value in *Rn*. If *offset* is omitted, the address is the value in *Rn*.

#### 38.2.4.8.2 Operation

LDREX, LDREXB, and LDREXH load a word, byte, and halfword respectively from a memory address.

STREX, STREXB, and STREXH attempt to store a word, byte, and halfword respectively to a memory address. The address used in any Store-Exclusive instruction must be the same as the address in the most recently executed Load-exclusive instruction. The value stored by the Store-Exclusive instruction must also have the same data size as the value loaded by the preceding Load-exclusive instruction. This means software must always use a Load-exclusive instruction and a matching Store-Exclusive instruction to perform a synchronization operation, see [Section 38.3.2.7 “Synchronization primitives”](#)

If an Store-Exclusive instruction performs the store, it writes 0 to its destination register. If it does not perform the store, it writes 1 to its destination register. If the Store-Exclusive instruction writes 0 to the destination register, it is guaranteed that no other process in the system has accessed the memory location between the Load-exclusive and Store-Exclusive instructions.

For reasons of performance, keep the number of instructions between corresponding Load-Exclusive and Store-Exclusive instruction to a minimum.

**Remark:** The result of executing a Store-Exclusive instruction to an address that is different from that used in the preceding Load-Exclusive instruction is unpredictable.

#### 38.2.4.8.3 Restrictions

In these instructions:

- do not use PC
- do not use SP for *Rd* and *Rt*
- for STREX, *Rd* must be different from both *Rt* and *Rn*
- the value of *offset* must be a multiple of four in the range 0-1020.

**38.2.4.8.4 Condition flags**

These instructions do not change the flags.

**38.2.4.8.5 Examples**

```

MOV    R1, #0x1           ; Initialize the 'lock taken' value
try
LDREX  R0, [LockAddr]    ; Load the lock value
CMP    R0, #0            ; Is the lock free?
ITT    EQ                ; IT instruction for STREXEQ and CMPEQ
STREXEQ R0, R1, [LockAddr] ; Try and claim the lock
CMPEQ  R0, #0            ; Did this succeed?
BNE    try               ; No - try again
....
; Yes - we have the lock
    
```



### 38.2.4.9 CLREX

Clear Exclusive.

#### 38.2.4.9.1 Syntax

CLREX{*cond*}

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.4.9.2 Operation

Use CLREX to make the next STREX, STREXB, or STREXH instruction write 1 to its destination register and fail to perform the store. It is useful in exception handler code to force the failure of the store exclusive if the exception occurs between a load exclusive instruction and the matching store exclusive instruction in a synchronization operation.

See [Section 38.3.2.7 “Synchronization primitives”](#) for more information.

#### 38.2.4.9.3 Condition flags

These instructions do not change the flags.

#### 38.2.4.9.4 Examples

```
CLREX
```

### 38.2.5 General data processing instructions

Table 750 shows the data processing instructions:

**Table 750. Data processing instructions**

Mnemonic	Brief description	See
ADC	Add with Carry	<a href="#">Section 38.2.5.1</a>
ADD	Add	<a href="#">Section 38.2.5.1</a>
ADDW	Add	<a href="#">Section 38.2.5.1</a>
AND	Logical AND	<a href="#">Section 38.2.5.2</a>
ASR	Arithmetic Shift Right	<a href="#">Section 38.2.5.3</a>
BIC	Bit Clear	<a href="#">Section 38.2.5.2</a>
CLZ	Count leading zeros	<a href="#">Section 38.2.5.4</a>
CMN	Compare Negative	<a href="#">Section 38.2.5.5</a>
CMP	Compare	<a href="#">Section 38.2.5.5</a>
EOR	Exclusive OR	<a href="#">Section 38.2.5.2</a>
LSL	Logical Shift Left	<a href="#">Section 38.2.5.3</a>
LSR	Logical Shift Right	<a href="#">Section 38.2.5.3</a>
MOV	Move	<a href="#">Section 38.2.5.6</a>
MOVT	Move Top	<a href="#">Section 38.2.5.7</a>
MOVW	Move 16-bit constant	<a href="#">Section 38.2.5.6</a>
MVN	Move NOT	<a href="#">Section 38.2.5.6</a>
ORN	Logical OR NOT	<a href="#">Section 38.2.5.2</a>
ORR	Logical OR	<a href="#">Section 38.2.5.2</a>
RBIT	Reverse Bits	<a href="#">Section 38.2.5.8</a>
REV	Reverse byte order in a word	<a href="#">Section 38.2.5.8</a>
REV16	Reverse byte order in each halfword	<a href="#">Section 38.2.5.8</a>
REVSH	Reverse byte order in bottom halfword and sign extend	<a href="#">Section 38.2.5.8</a>
ROR	Rotate Right	<a href="#">Section 38.2.5.3</a>
RRX	Rotate Right with Extend	<a href="#">Section 38.2.5.3</a>
RSB	Reverse Subtract	<a href="#">Section 38.2.5.1</a>
SBC	Subtract with Carry	<a href="#">Section 38.2.5.1</a>
SUB	Subtract	<a href="#">Section 38.2.5.1</a>
SUBW	Subtract	<a href="#">Section 38.2.5.1</a>
TEQ	Test Equivalence	<a href="#">Section 38.2.5.9</a>
TST	Test	<a href="#">Section 38.2.5.9</a>

### 38.2.5.1 ADD, ADC, SUB, SBC, and RSB

Add, Add with carry, Subtract, Subtract with carry, and Reverse Subtract.

#### 38.2.5.1.1 Syntax

$op\{S\}\{cond\} \{Rd,\} Rn, Operand2$

$op\{cond\} \{Rd,\} Rn, \#imm12$  ; ADD and SUB only

where:

*op* is one of:

- ADD: Add.
- ADC: Add with Carry.
- SUB: Subtract.
- RSB: Reverse Subtract.

*S*: is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Section 38.2.3.7 “Conditional execution”](#).

*cond*: is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register. If *Rd* is omitted, the destination register is *Rn*.

*Rn* is the register holding the first operand.

*Operand2* is a flexible second operand. See [Section 38.2.3.3](#) for details of the options.

*imm12* is any value in the range 0-4095.

#### 38.2.5.1.2 Operation

The **ADD** instruction adds the value of *Operand2* or *imm12* to the value in *Rn*.

The **ADC** instruction adds the values in *Rn* and *Operand2*, together with the carry flag.

The **SUB** instruction subtracts the value of *Operand2* or *imm12* from the value in *Rn*.

The **SBC** instruction subtracts the value of *Operand2* from the value in *Rn*. If the carry flag is clear, the result is reduced by one.

The **RSB** instruction subtracts the value in *Rn* from the value of *Operand2*. This is useful because of the wide range of options for *Operand2*.

Use **ADC** and **SBC** to synthesize multiword arithmetic, see [Section 38.2.5.1.6](#).

See also [Section 38.2.4.1](#).

**Remark:** **ADDW** is equivalent to the **ADD** syntax that uses the *imm12* operand. **SUBW** is equivalent to the **SUB** syntax that uses the *imm12* operand.

#### 38.2.5.1.3 Restrictions

- *Operand2* must not be SP and must not be PC
- *Rd* can be SP only in **ADD** and **SUB**, and only with the additional restrictions:

- *Rn* must also be SP
- any shift in *Operand2* must be limited to a maximum of 3 bits using LSL
- *Rn* can be SP only in ADD and SUB
- *Rd* can be PC only in the *cond* instruction where:
  - you must not specify the S suffix
  - *Rm* must not be PC and must not be SP
  - if the instruction is conditional, it must be the last instruction in the IT block
- with the exception of the *cond* instruction, *Rn* can be PC only in ADD and SUB, and only with the additional restrictions:
  - you must not specify the S suffix
  - the second operand must be a constant in the range 0 to 4095.

#### Note

- When using the PC for an addition or a subtraction, bits[1:0] of the PC are rounded to b00 before performing the calculation, making the base address for the calculation word-aligned.
- If you want to generate the address of an instruction, you have to adjust the constant based on the value of the PC. ARM recommends that you use the ADR instruction instead of ADD or SUB with *Rn* equal to the PC, because your assembler automatically calculates the correct constant for the ADR instruction.

When *Rd* is PC in the *cond* instruction:

- bit[0] of the value written to the PC is ignored
- a branch occurs to the address created by forcing bit[0] of that value to 0.

#### 38.2.5.1.4 Condition flags

If S is specified, these instructions update the N, Z, C and V flags according to the result.

#### 38.2.5.1.5 Examples

```

ADD    R2, R1, R3
SUBS   R8, R6, #240    ; Sets the flags on the result
RSB    R4, R4, #1280   ; Subtracts contents of R4 from 1280
ADCHI  R11, R0, R3    ; Only executed if C flag set and Z
                          ; flag clear

```

#### 38.2.5.1.6 Multiword arithmetic examples

[Section](#) shows two instructions that add a 64-bit integer contained in R2 and R3 to another 64-bit integer contained in R0 and R1, and place the result in R4 and R5.

Multiword values do not have to use consecutive registers. [Section](#) shows instructions that subtract a 96-bit integer contained in R9, R1, and R11 from another contained in R6, R2, and R8. The example stores the result in R6, R9, and R2.

#### 64-bit addition:

```

ADDS   R4, R0, R2    ; add the least significant words
ADC    R5, R1, R3    ; add the most significant words with carry

```

**96-bit subtraction:**

```
SUBS    R6, R6, R9    ; subtract the least significant words
SBCS    R9, R2, R1    ; subtract the middle words with carry
SBC     R2, R8, R11   ; subtract the most significant words with carry
```

**38.2.5.2 AND, ORR, EOR, BIC, and ORN**

Logical AND, OR, Exclusive OR, Bit Clear, and OR NOT.

**38.2.5.2.1 Syntax**

*op*{*S*}{*cond*} {*Rd*,} *Rn*, *Operand2*

where:

*op* is one of:

- AND: logical AND.
- ORR: logical OR, or bit set.
- EOR: logical Exclusive OR.
- BIC: logical AND NOT, or bit clear.
- ORN: logical OR NOT.

*S* is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Section 38.2.3.7 “Conditional execution”](#).

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*Rn* is the register holding the first operand.

*Operand2* is a flexible second operand. See [Section 38.2.3.3](#) for details of the options.

**38.2.5.2.2 Operation**

The AND, EOR, and ORR instructions perform bitwise AND, Exclusive OR, and OR operations on the values in *Rn* and *Operand2*.

The BIC instruction performs an AND operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

The ORN instruction performs an OR operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

**38.2.5.2.3 Restrictions**

Do not use SP and do not use PC.

**38.2.5.2.4 Condition flags**

If *S* is specified, these instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Section 38.2.3.3](#)
- do not affect the V flag.

**38.2.5.2.5 Examples**

```

AND      R9, R2, #0xFF00
ORREQ   R2, R0, R5
ANDS    R9, R8, #0x19
    
```



### 38.2.5.3 ASR, LSL, LSR, ROR, and RRX

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, Rotate Right, and Rotate Right with Extend.

#### 38.2.5.3.1 Syntax

$op\{S\}\{cond\} Rd, Rm, Rs$

$op\{S\}\{cond\} Rd, Rm, \#n$

$RRX\{S\}\{cond\} Rd, Rm$

where:

$op$  is one of:

ASR: Arithmetic Shift Right.

LSL: Logical Shift Left.

LSR: Logical Shift Right.

ROR: Rotate Right.

$S$  is an optional suffix. If  $S$  is specified, the condition code flags are updated on the result of the operation, see [Section 38.2.3.7 “Conditional execution”](#).

$Rd$  is the destination register.

$Rm$  is the register holding the shift length to apply to the value in  $Rm$ . Only the least significant byte is used and can be in the range 0 to 255.

$Rs$  is the register holding the shift length to apply to the value in  $Rm$ . Only the least significant byte is used and can be in the range 0 to 255.

$n$  is the shift length. The range of shift length depends on the instruction:

ASR: shift length from 1 to 32

LSL: shift length from 0 to 31

LSR: shift length from 1 to 32

ROR: shift length from 1 to 31

**Remark:**  $MOV\{S\}\{cond\} Rd, Rm$  is the preferred syntax for  $LSL\{S\}\{cond\} Rd, Rm, \#0$ .

#### 38.2.5.3.2 Operation

ASR, LSL, LSR, and ROR move the bits in the register  $Rm$  to the left or right by the number of places specified by constant  $n$  or register  $Rs$ .

RRX moves the bits in register  $Rm$  to the right by 1.

In all these instructions, the result is written to  $Rd$ , but the value in register  $Rm$  remains unchanged. For details on what result is generated by the different instructions, see [Section 38.2.3.4 “Shift Operations”](#).

#### 38.2.5.3.3 Restrictions

Do not use SP and do not use PC.



#### 38.2.5.3.4 Condition flags

If S is specified:

- these instructions update the N and Z flags according to the result
- the C flag is updated to the last bit shifted out, except when the shift length is 0, see [Section 38.2.3.4 "Shift Operations"](#).

#### 38.2.5.3.5 Examples

```
ASR   R7, R8, #9 ; Arithmetic shift right by 9 bits
LSLS  R1, R2, #3 ; Logical shift left by 3 bits with flag update
LSR   R4, R5, #6 ; Logical shift right by 6 bits
ROR   R4, R5, R6 ; Rotate right by the value in the bottom byte of R6
RRX   R4, R5     ; Rotate right with extend
```

#### 38.2.5.4 CLZ

Count Leading Zeros.

##### 38.2.5.4.1 Syntax

*CLZ*{*cond*} *Rd*, *Rm*

where:

*cond* is an optional condition code, see [Section 38.2.3.7](#).

*Rd* is the destination register.

*Rm* is the operand register.

##### 38.2.5.4.2 Operation

The *CLZ* instruction counts the number of leading zeros in the value in *Rm* and returns the result in *Rd*. The result value is 32 if no bits are set in the source register, and zero if bit[31] is set.

##### 38.2.5.4.3 Restrictions

Do not use SP and do not use PC.

##### 38.2.5.4.4 Condition flags

This instruction does not change the flags.

##### 38.2.5.4.5 Examples

```
CLZ      R4, R9
CLZNE   R2, R3
```

### 38.2.5.5 CMP and CMN

Compare and Compare Negative.

#### 38.2.5.5.1 Syntax

*CMP*{*cond*} *Rn*, *Operand2*

*CMN*{*cond*} *Rn*, *Operand2*

where:

*cond* is an optional condition code, see [Section 38.2.3.7](#).

*Rn* is the register holding the first operand.

*Operand2* is a flexible second operand. See Flexible second operand on page 3-10 for details of the options.

#### 38.2.5.5.2 Operation

These instructions compare the value in a register with *Operand2*. They update the condition flags on the result, but do not write the result to a register.

The *CMP* instruction subtracts the value of *Operand2* from the value in *Rn*. This is the same as a *SUBS* instruction, except that the result is discarded.

The *CMN* instruction adds the value of *Operand2* to the value in *Rn*. This is the same as an *ADDS* instruction, except that the result is discarded.

#### 38.2.5.5.3 Restrictions

In these instructions:

- do not use PC
- *Operand2* must not be SP.

#### 38.2.5.5.4 Condition flags

These instructions update the N, Z, C and V flags according to the result.

#### 38.2.5.5.5 Examples

```
CMP    R2, R9
CMN    R0, #6400
CMPGT  SP, R7, LSL #2
```

### 38.2.5.6 MOV and MVN

Move and Move NOT.

#### 38.2.5.6.1 Syntax

$MOV\{S\}\{cond\} Rd, Operand2$

$MOV\{cond\} Rd, \#imm16$

$MVN\{S\}\{cond\} Rd, Operand2$

where:

*S* is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Section 38.2.3.7](#).

*cond* is an optional condition code, see [Section 38.2.3.7](#).

*Rd* is the destination register.

*Operand2* is a flexible second operand. See Flexible second operand on page 3-10 for details of the options.

*imm16* is any value in the range 0-65535.

#### 38.2.5.6.2 Operation

The *MOV* instruction copies the value of *Operand2* into *Rd*.

When *Operand2* in a *MOV* instruction is a register with a shift other than *LSL #0*, the preferred syntax is the corresponding shift instruction:

- $ASR\{S\}\{cond\} Rd, Rm, \#n$  is the preferred syntax for  $MOV\{S\}\{cond\} Rd, Rm, ASR \#n$
- $LSL\{S\}\{cond\} Rd, Rm, \#n$  is the preferred syntax for  $MOV\{S\}\{cond\} Rd, Rm, LSL \#n$  if  $n \neq 0$
- $LSR\{S\}\{cond\} Rd, Rm, \#n$  is the preferred syntax for  $MOV\{S\}\{cond\} Rd, Rm, LSR \#n$
- $ROR\{S\}\{cond\} Rd, Rm, \#n$  is the preferred syntax for  $MOV\{S\}\{cond\} Rd, Rm, ROR \#n$
- $RRX\{S\}\{cond\} Rd, Rm$  is the preferred syntax for  $MOV\{S\}\{cond\} Rd, Rm, RRX$ .

Also, the *MOV* instruction permits additional forms of *Operand2* as synonyms for shift instructions:

- $MOV\{S\}\{cond\} Rd, Rm, ASR Rs$  is a synonym for  $ASR\{S\}\{cond\} Rd, Rm, Rs$
- $MOV\{S\}\{cond\} Rd, Rm, LSL Rs$  is a synonym for  $LSL\{S\}\{cond\} Rd, Rm, Rs$
- $MOV\{S\}\{cond\} Rd, Rm, LSR Rs$  is a synonym for  $LSR\{S\}\{cond\} Rd, Rm, Rs$
- $MOV\{S\}\{cond\} Rd, Rm, ROR Rs$  is a synonym for  $ROR\{S\}\{cond\} Rd, Rm, Rs$

See [Section 38.2.5.3](#).

The *MVN* instruction takes the value of *Operand2*, performs a bitwise logical NOT operation on the value, and places the result into *Rd*.

**Remark:** The *MOVW* instruction provides the same function as *MOV*, but is restricted to using the *imm16* operand.

### 38.2.5.6.3 Restrictions

You can use SP and PC only in the MOV instruction, with the following restrictions:

- the second operand must be a register without shift
- you must not specify the S suffix.

When *Rd* is PC in a MOV instruction:

- bit[0] of the value written to the PC is ignored
- a branch occurs to the address created by forcing bit[0] of that value to 0.

**Remark:** Though it is possible to use MOV as a branch instruction, ARM strongly recommends the use of a BX or BLX instruction to branch for software portability to the ARM instruction set.

### 38.2.5.6.4 Condition flags

If S is specified, these instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Section 38.2.3.3](#)
- do not affect the V flag.

### 38.2.5.6.5 Example

```
MOVS R11, #0x000B ; Write value of 0x000B to R11, flags get updated
MOV R1, #0xFA05 ; Write value of 0xFA05 to R1, flags are not updated
MOVS R10, R12 ; Write value in R12 to R10, flags get updated
MOV R3, #23 ; Write value of 23 to R3
MOV R8, SP ; Write value of stack pointer to R8
MVNS R2, #0xF ; Write value of 0xFFFFFFFF (bitwise inverse of 0xF)
; to the R2 and update flags
```

### 38.2.5.7 MOV<sub>T</sub>

Move Top.

#### 38.2.5.7.1 Syntax

*MOV<sub>T</sub>*{*cond*} *Rd*, #*imm16*

where:

*cond* is an optional condition code, see [Section 38.2.3.7](#).

*Rd* is the destination register.

*imm16* is a 16-bit immediate constant.

#### 38.2.5.7.2 Operation

*MOV<sub>T</sub>* writes a 16-bit immediate value, *imm16*, to the top halfword, *Rd*[31:16], of its destination register. The write does not affect *Rd*[15:0].

The *MOV*, *MOV<sub>T</sub>* instruction pair enables you to generate any 32-bit constant.

#### 38.2.5.7.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 38.2.5.7.4 Condition flags

This instruction does not change the flags.

#### 38.2.5.7.5 Examples

```
MOVT    R3, #0xF123 ; Write 0xF123 to upper halfword of R3, lower halfword
                ; and APSR are unchanged
```

**38.2.5.8 REV, REV16, REVSH, and RBIT**

Reverse bytes and Reverse bits.

**38.2.5.8.1 Syntax**

*op{cond} Rd, Rn*

where:

*op* is any of:

REV Reverse byte order in a word.

REV16 Reverse byte order in each halfword independently.

REVSH Reverse byte order in the bottom halfword, and sign extend to

RBIT Reverse the bit order in a 32-bit word.

*cond* is an optional condition code, see [Section 38.2.3.7](#).

*Rd* is the destination register.

*Rn* is the register holding the operand.

**38.2.5.8.2 Operation**

Use these instructions to change endianness of data:

REV: converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

REV16 converts 16-bit big-endian data into little-endian data or 16-bit little-endian data into big-endian data.

REVSH converts either:

16-bit signed big-endian data into 32-bit signed little-endian data

16-bit signed little-endian data into 32-bit signed big-endian data.

**38.2.5.8.3 Restrictions**

Do not use SP and do not use PC.

**38.2.5.8.4 Condition flags**

These instructions do not change the flags.

**38.2.5.8.5 Examples**

```
REV    R3, R7 ; Reverse byte order of value in R7 and write it to R3
REV16  R0, R0 ; Reverse byte order of each 16-bit halfword in R0
REVSH  R0, R5 ; Reverse Signed Halfword
REVSH  R3, R7 ; Reverse with Higher or Same condition
RBIT   R7, R8 ; Reverse bit order of value in R8 and write the result to R7
```

### 38.2.5.9 TST and TEQ

Test bits and Test Equivalence.

#### 38.2.5.9.1 Syntax

TST{*cond*} *Rn*, *Operand2*

TEQ{*cond*} *Rn*, *Operand2*

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rn* is the register holding the first operand.

*Operand2* is a flexible second operand. See [Section 38.2.3.3](#) for details of the options.

#### 38.2.5.9.2 Operation

These instructions test the value in a register against *Operand2*. They update the condition flags based on the result, but do not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

#### 38.2.5.9.3 Restrictions

Do not use SP and do not use PC.

#### 38.2.5.9.4 Condition flags

These instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Section 38.2.3.3](#)
- do not affect the V flag.

#### 38.2.5.9.5 Examples

```

TST    R0, #0x3F8 ; Perform bitwise AND of R0 value to 0x3F8,
                ; APSR is updated but result is discarded
TEQEQ  R10, R9   ; Conditionally test if value in R10 is equal to
                ; value in R9, APSR is updated but result is discarded

```



### 38.2.6 Multiply and divide instructions

[Table 751](#) shows the multiply and divide instructions:

**Table 751. Multiply and divide instructions**

Mnemonic	Brief description	See
MLA	Multiply with Accumulate, 32-bit result	<a href="#">Section 38.2.6.1</a>
MLS	Multiply and Subtract, 32-bit result	<a href="#">Section 38.2.6.1</a>
MUL	Multiply, 32-bit result	<a href="#">Section 38.2.6.1</a>
SDIV	Signed Divide	<a href="#">Section 38.2.6.3</a>
SMLAL	Signed Multiply with Accumulate (32x32+64), 64-bit result	<a href="#">Section 38.2.6.2</a>
SMULL	Signed Multiply (32x32), 64-bit result	<a href="#">Section 38.2.6.2</a>
UDIV	Unsigned Divide	<a href="#">Section 38.2.6.3</a>
UMLAL	Unsigned Multiply with Accumulate (32x32+64), 64-bit result	<a href="#">Section 38.2.6.2</a>
UMULL	Unsigned Multiply (32x32), 64-bit result	<a href="#">Section 38.2.6.2</a>

### 38.2.6.1 MUL, MLA, and MLS

Multiply, Multiply with Accumulate, and Multiply with Subtract, using 32-bit operands, and producing a 32-bit result.

#### 38.2.6.1.1 Syntax

`MUL`{S}{*cond*} {*Rd*,} *Rn*, *Rm* ; Multiply

`MLA`{*cond*} *Rd*, *Rn*, *Rm*, *Ra* ; Multiply with accumulate

`MLS`{*cond*} *Rd*, *Rn*, *Rm*, *Ra* ; Multiply with subtract

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register. If *Rd* is omitted, the destination register is *Rn*.

*Rn*, *Rm* are registers holding the values to be multiplied.

*Ra* is a register holding the value to be added or subtracted from.

#### 38.2.6.1.2 Operation

The `MUL` instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The `MLA` instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The `MLS` instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The results of these instructions do not depend on whether the operands are signed or unsigned.

#### 38.2.6.1.3 Restrictions

In these instructions, do not use SP and do not use PC.

If you use the S suffix with the `MUL` instruction:

- *Rd*, *Rn*, and *Rm* must all be in the range R0 to R7
- *Rd* must be the same as *Rm*
- you must not use the *cond* suffix.

#### 38.2.6.1.4 Condition flags

If S is specified, the `MUL` instruction:

- updates the N and Z flags according to the result
- does not affect the C and V flags.

### 38.2.6.1.5 Examples

```
MUL    R10, R2, R5      ; Multiply, R10 = R2 x R5
MLA    R10, R2, R1, R5  ; Multiply with accumulate, R10 = (R2 x R1) + R5
MULS   R0, R2, R2       ; Multiply with flag update, R0 = R2 x R2
MULLT  R2, R3, R2       ; Conditionally multiply, R2 = R3 x R2
MLS    R4, R5, R6, R7   ; Multiply with subtract, R4 = R7 - (R5 x R6)
```

### 38.2.6.2 UMULL, UMLAL, SMULL, and SMLAL

Signed and Unsigned Long Multiply, with optional Accumulate, using 32-bit operands and producing a 64-bit result.

#### 38.2.6.2.1 Syntax

*op{cond} RdLo, RdHi, Rn, Rm*

where:

*op* is one of:

- UMULL: Unsigned Long Multiply.
- UMLAL: Unsigned Long Multiply, with Accumulate.
- SMULL: Signed Long Multiply.
- SMLAL: Signed Long Multiply, with Accumulate.

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*RdHi*, *RdLo* are the destination registers. For UMLAL and SMLAL they also hold the accumulating value.

*Rn*, *Rm* are registers holding the operands.

#### 38.2.6.2.2 Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

#### 38.2.6.2.3 Restrictions

In these instructions:

- do not use SP and do not use PC
- *RdHi* and *RdLo* must be different registers.

#### 38.2.6.2.4 Condition flags

These instructions do not affect the condition code flags.

### 38.2.6.2.5 Examples

```
UMULL    R0, R4, R5, R6 ; Unsigned (R4,R0) = R5 x R6
SMLAL   R4, R5, R3, R8 ; Signed (R5,R4) = (R5,R4) + R3 x R8
```

### 38.2.6.3 SDIV and UDIV

Signed Divide and Unsigned Divide.

#### 38.2.6.3.1 Syntax

$\text{SDIV}\{\text{cond}\} \{Rd,\} Rn, Rm$

$\text{UDIV}\{\text{cond}\} \{Rd,\} Rn, Rm$

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register. If *Rd* is omitted, the destination register is *Rn*.

*Rn* is the register holding the value to be divided.

*Rm* is a register holding the divisor.

#### 38.2.6.3.2 Operation

**SDIV** performs a signed integer division of the value in *Rn* by the value in *Rm*.

**UDIV** performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

#### 38.2.6.3.3 Restrictions

Do not use SP and do not use PC.

#### 38.2.6.3.4 Condition flags

These instructions do not change the flags.

#### 38.2.6.3.5 Examples

```
SDIV R0, R2, R4 ; Signed divide, R0 = R2/R4
UDIV R8, R8, R1 ; Unsigned divide, R8 = R8/R1
```

## 38.2.7 Saturating instructions

This section describes the saturating instructions, `SSAT` and `USAT`.

### 38.2.7.1 SSAT and USAT

Signed Saturate and Unsigned Saturate to any bit position, with optional shift before saturating.

#### 38.2.7.1.1 Syntax

`op{cond} Rd, #n, Rm {, shift #s}`

where:

`op` is one of:

- `SSAT` Saturates a signed value to a signed range.
- `USAT` Saturates a signed value to an unsigned range.

`cond` is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

`Rd` is the destination register.

`n` specifies the bit position to saturate to:

- `n` ranges from 1 to 32 for `SSAT`.
- `n` ranges from 0 to 31 for `USAT`.

`Rm` is the register containing the value to saturate.

`shift #s` is an optional shift applied to `Rm` before saturating. It must be one of the following:

- `ASR #s`: where `s` is in the range 1 to 31
- `LSL #s`: where `s` is in the range 0 to 31.

#### 38.2.7.1.2 Operation

These instructions saturate to a signed or unsigned `n`-bit value.

The `SSAT` instruction applies the specified shift, then saturates to the signed range  $-2^{n-1} \leq x \leq 2^{n-1}-1$ .

The `USAT` instruction applies the specified shift, then saturates to the unsigned range  $0 \leq x \leq 2^n-1$ .

For signed `n`-bit saturation using `SSAT`, this means that:

- if the value to be saturated is less than  $-2^{n-1}$ , the result returned is  $-2^{n-1}$
- if the value to be saturated is greater than  $2^{n-1}-1$ , the result returned is  $2^{n-1}-1$
- otherwise, the result returned is the same as the value to be saturated.

For unsigned `n`-bit saturation using `USAT`, this means that:

- if the value to be saturated is less than 0, the result returned is 0
- if the value to be saturated is greater than  $2^n-1$ , the result returned is  $2^n-1$

- otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called **saturation**. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, you must use the MSR instruction, see [Section 38.2.10.7](#).

To read the state of the Q flag, use the MRS instruction, see [Section 38.2.10.6](#).

#### 38.2.7.1.3 Restrictions

Do not use SP and do not use PC.

#### 38.2.7.1.4 Condition flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

#### 38.2.7.1.5 Examples

```
SSAT    R7, #16, R7, LSL #4 ; Logical shift left value in R7 by 4, then
                                ; saturate it as a signed 16-bit value and
                                ; write it back to R7
USATNE  R0, #7, R5          ; Conditionally saturate value in R5 as an
                                ; unsigned 7 bit value and write it to R0
```



### 38.2.8 Bitfield instructions

[Table 752](#) shows the instructions that operate on adjacent sets of bits in registers or bitfields:

**Table 752. Packing and unpacking instructions**

Mnemonic	Brief description	See
BFC	Bit Field Clear	<a href="#">Section 38.2.8.1</a>
BFI	Bit Field Insert	<a href="#">Section 38.2.8.1</a>
SBFX	Signed Bit Field Extract	<a href="#">Section 38.2.8.2</a>
SXTB	Sign extend a byte	<a href="#">Section 38.2.8.3</a>
SXTH	Sign extend a halfword	<a href="#">Section 38.2.8.3</a>
UBFX	Unsigned Bit Field Extract	<a href="#">Section 38.2.8.2</a>
UXTB	Zero extend a byte	<a href="#">Section 38.2.8.3</a>
UXTH	Zero extend a halfword	<a href="#">Section 38.2.8.3</a>

**38.2.8.1 BFC and BFI**

Bit Field Clear and Bit Field Insert.

**38.2.8.1.1 Syntax**

BFC{*cond*} *Rd*, #*lsb*, #*width*

BFI{*cond*} *Rd*, *Rn*, #*lsb*, #*width*

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*Rn* is the source register.

*lsb* is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

*width* is the width of the bitfield and must be in the range 1 to 32–*lsb*.

**38.2.8.1.2 Operation**

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

**38.2.8.1.3 Restrictions**

Do not use SP and do not use PC.

**38.2.8.1.4 Condition flags**

These instructions do not affect the flags.

**38.2.8.1.5 Examples**

```
BFC  R4, #8, #12      ; Clear bit 8 to bit 19 (12 bits) of R4 to 0
BFI  R9, R2, #8, #12 ; Replace bit 8 to bit 19 (12 bits) of R9 with
                        ; bit 0 to bit 11 from R2
```

### 38.2.8.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

#### 38.2.8.2.1 Syntax

`SBFX{cond} Rd, Rn, #lsb, #width`

`UBFX{cond} Rd, Rn, #lsb, #width`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*Rn* is the source register.

*lsb* is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

*width* is the width of the bitfield and must be in the range 1 to 32–*lsb*.

#### 38.2.8.2.2 Operation

`SBFX` extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

`UBFX` extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

#### 38.2.8.2.3 Restrictions

Do not use SP and do not use PC.

#### 38.2.8.2.4 Condition flags

These instructions do not affect the flags.

#### 38.2.8.2.5 Examples

```
SBFX R0, R1, #20, #4 ; Extract bit 20 to bit 23 (4 bits) from R1 and sign
                        ; extend to 32 bits and then write the result to R0.
UBFX R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero
                        ; extend to 32 bits and then write the result to R8
```

### 38.2.8.3 SXT and UXT

Sign extend and Zero extend.

#### 38.2.8.3.1 Syntax

$SXT_{extend}\{cond\} \{Rd\}, Rm \{, ROR \#n\}$

$UXT_{extend}\{cond\} \{Rd\}, Rm \{, ROR \#n\}$

where:

*extend* is one of:

- B: Extends an 8-bit value to a 32-bit value.
- H: Extends a 16-bit value to a 32-bit value.

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*Rm* is the register holding the value to extend.

ROR *#n* is one of:

- ROR #8: Value from *Rm* is rotated right 8 bits.
- ROR #16: Value from *Rm* is rotated right 16 bits.
- ROR #24: Value from *Rm* is rotated right 24 bits.

If ROR *#n* is omitted, no rotation is performed.

#### 38.2.8.3.2 Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
  - SXTB extracts bits[7:0] and sign extends to 32 bits.
  - UXTB extracts bits[7:0] and zero extends to 32 bits.
  - SXTH extracts bits[15:0] and sign extends to 32 bits.
  - UXTH extracts bits[15:0] and zero extends to 32 bits.

#### 38.2.8.3.3 Restrictions

Do not use SP and do not use PC.

#### 38.2.8.3.4 Condition flags

These instructions do not affect the flags.

### 38.2.8.3.5 Examples

```
SXTH R4, R6, ROR #16 ; Rotate R6 right by 16 bits, then obtain the lower
                       ; halfword of the result and then sign extend to
                       ; 32 bits and write the result to R4.
UXTB R3, R10         ; Extract lowest byte of the value in R10 and zero
                       ; extend it, and write the result to R3
```

### 38.2.9 Branch and control instructions

[Table 753](#) shows the branch and control instructions:

**Table 753. Branch and control instructions**

Mnemonic	Brief description	See
B	Branch	<a href="#">Section 38.2.9.1</a>
BL	Branch with Link	<a href="#">Section 38.2.9.1</a>
BLX	Branch indirect with Link	<a href="#">Section 38.2.9.1</a>
BX	Branch indirect	<a href="#">Section 38.2.9.1</a>
CBNZ	Compare and Branch if Non Zero	<a href="#">Section 38.2.9.2</a>
CBZ	Compare and Branch if Non Zero	<a href="#">Section 38.2.9.2</a>
IT	If-Then	<a href="#">Section 38.2.9.3</a>
TBB	Table Branch Byte	<a href="#">Section 38.2.9.4</a>
TBH	Table Branch Halfword	<a href="#">Section 38.2.9.4</a>

**38.2.9.1 B, BL, BX, and BLX**

Branch instructions.

**38.2.9.1.1 Syntax**

*B{cond} label*

*BL{cond} label*

*BX{cond} Rm*

*BLX{cond} Rm*

where:

B is branch (immediate).

BL is branch with link (immediate).

BX is branch indirect (register).

BLX is branch indirect with link (register).

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*label* is a PC-relative expression. See [Section 38.2.3.6 “PC-relative expressions”](#).

*Rm* is a register that indicates an address to branch to. Bit[0] of the value in *Rm* must be 1, but the address to branch to is created by changing bit[0] to 0.

**38.2.9.1.2 Operation**

All these instructions cause a branch to *label*, or to the address indicated in *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR (the link register, R14).
- The BX and BLX instructions cause a UsageFault exception if bit[0] of *Rm* is 0.

*Bcond label* is the only conditional instruction that can be either inside or outside an IT block. All other branch instructions must be conditional inside an IT block, and must be unconditional outside the IT block, see [Section 38.2.9.3](#).

[Table 754](#) shows the ranges for the various branch instructions.

**Table 754. Branch ranges**

Instruction	Branch range
B <i>label</i>	–16 MB to +16 MB
<i>Bcond label</i> (outside IT block)	–1 MB to +1 MB
<i>Bcond label</i> (inside IT block)	–16 MB to +16 MB
BL{ <i>cond</i> } <i>label</i>	–16 MB to +16 MB
BX{ <i>cond</i> } <i>Rm</i>	Any value in register
BLX{ <i>cond</i> } <i>Rm</i>	Any value in register

**Remark:** You might have to use the .W suffix to get the maximum branch range. See [Section 38.2.3.8 “Instruction width selection”](#).

### 38.2.9.1.3 Restrictions

The restrictions are:

- do not use PC in the BLX instruction
- for BX and BLX, bit[0] of *Rm* must be 1 for correct execution but a branch occurs to the target address created by changing bit[0] to 0
- when any of these instructions is inside an IT block, it must be the last instruction of the IT block.

*Bcond* is the only conditional instruction that is not required to be inside an IT block. However, it has a longer branch range when it is inside an IT block.

### 38.2.9.1.4 Condition flags

These instructions do not change the flags.

### 38.2.9.1.5 Examples

```
B      loopA ; Branch to loopA
BLE   ng    ; Conditionally branch to label ng
B.W   target ; Branch to target within 16MB range
BEQ   target ; Conditionally branch to target
BEQ.W target ; Conditionally branch to target within 1MB
BL    funC  ; Branch with link (Call) to function funC, return address
      ; stored in LR
BX    LR    ; Return from function call
BXNE  R0    ; Conditionally branch to address stored in R0
BLX   R0    ; Branch with link and exchange (Call) to a address stored
      ; in R0
```



### 38.2.9.2 CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

#### 38.2.9.2.1 Syntax

CBZ *Rn*, *label*

CBNZ *Rn*, *label*

where:

*Rn* is the register holding the operand.

*label* is the branch destination.

#### 38.2.9.2.2 Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ *Rn*, *label* does not change condition flags but is otherwise equivalent to:

```
CMP    Rn, #0
      BEQ    label
```

CBNZ *Rn*, *label* does not change condition flags but is otherwise equivalent to:

```
CMP    Rn, #0
      BNE    label
```

#### 38.2.9.2.3 Restrictions

The restrictions are:

- *Rn* must be in the range of R0 to R7
- the branch destination must be within 4 to 130 bytes after the instruction
- these instructions must not be used inside an IT block.

#### 38.2.9.2.4 Condition flags

These instructions do not change the flags.

#### 38.2.9.2.5 Examples

```
CBZ    R5, target ; Forward branch if R5 is zero
CBNZ   R0, target ; Forward branch if R0 is not zero
```

### 38.2.9.3 IT

If-Then condition instruction.

#### 38.2.9.3.1 Syntax

`IT{x{y{z}}} cond`

where:

*x* specifies the condition switch for the second instruction in the IT block.

*y* specifies the condition switch for the third instruction in the IT block.

*z* specifies the condition switch for the fourth instruction in the IT block.

*cond* specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

**T**: Then. Applies the condition *cond* to the instruction.

**E**: Else. Applies the inverse condition of *cond* to the instruction.

**Remark:** It is possible to use **AL** (the **always** condition) for *cond* in an **IT** instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be **T** or omitted but not **E**.

#### 38.2.9.3.2 Operation

The **IT** instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the **IT** instruction form the **IT block**.

The instructions in the IT block, including any branches, must specify the condition in the *{cond}* part of their syntax.

**Remark:** Your assembler might be able to generate the required **IT** instructions for conditional instructions automatically, so that you do not need to write them yourself. See your assembler documentation for details.

A **BKPT** instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an **IT** instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

#### 38.2.9.3.3 Restrictions

The following instructions are not permitted in an IT block:

- **IT**
- **CBZ** and **CBNZ**

- CPSID and CPSIE.

Other restrictions when using an IT block are:

- a branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
  - ADD PC, PC, Rm
  - MOV PC, Rm
  - B, BL, BX, BLX
  - any LDM, LDR, or POP instruction that writes to the PC
  - TBB and TBH
- do not branch to any instruction inside an IT block, except when returning from an exception handler
- all conditional instructions except *Bcond* must be inside an IT block. *Bcond* can be either outside or inside an IT block but has a larger branch range if it is inside one
- each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

**Remark:** Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

#### 38.2.9.3.4 Condition flags

This instruction does not change the flags.

#### 38.2.9.3.5 Example

```

ITTE  NE           ; Next 3 instructions are conditional
ANDNE R0, R0, R1   ; ANDNE does not update condition flags
ADDSNE R2, R2, #1  ; ADDSNE updates condition flags
MOVEQ R2, R3       ; Conditional move

CMP   R0, #9       ; Convert R0 hex value (0 to 15) into ASCII
                        ; ('0'-'9', 'A'-'F')
ITE   GT           ; Next 2 instructions are conditional
ADDGT R1, R0, #55  ; Convert 0xA -> 'A'
ADDLE R1, R0, #48  ; Convert 0x0 -> '0'

IT    GT           ; IT block with only one conditional instruction

ADDGT R1, R1, #1   ; Increment R1 conditionally

ITTEE EQ           ; Next 4 instructions are conditional
MOVEQ R0, R1       ; Conditional move
ADDEQ R2, R2, #10  ; Conditional add
ANDNE R3, R3, #1   ; Conditional AND
BNE.W dloop        ; Branch instruction can only be used in the last
                        ; instruction of an IT block

```

```
IT    NE          ; Next instruction is conditional
ADD   R0, R0, R1  ; Syntax error: no condition code used in IT block
```

### 38.2.9.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

#### 38.2.9.4.1 Syntax

TBB [*Rn*, *Rm*]

TBH [*Rn*, *Rm*, LSL #1]

where:

*Rn* is the register containing the address of the table of branch lengths.

If *Rn* is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

*Rm* is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in *Rm* to form the right offset into the table.

#### 38.2.9.4.2 Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table. and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

#### 38.2.9.4.3 Restrictions

The restrictions are:

- *Rn* must not be SP
- *Rm* must not be SP and must not be PC
- when any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

#### 38.2.9.4.4 Condition flags

These instructions do not change the flags.

#### 38.2.9.4.5 Examples

```

ADR.W R0, BranchTable_Byte
TBB [R0, R1] ; R1 is the index, R0 is the base address of the
              ; branch table

Case1
; an instruction sequence follows
Case2
; an instruction sequence follows
Case3
; an instruction sequence follows
BranchTable_Byte
DCB 0 ; Case1 offset calculation
DCB ((Case2-Case1)/2) ; Case2 offset calculation
DCB ((Case3-Case1)/2) ; Case3 offset calculation

```

```
    TBH    [PC, R1, LSL #1] ; R1 is the index, PC is used as base of the
                                ; branch table

BranchTable_H
    DCI    ((CaseA - BranchTable_H)/2) ; CaseA offset calculation
    DCI    ((CaseB - BranchTable_H)/2) ; CaseB offset calculation
    DCI    ((CaseC - BranchTable_H)/2) ; CaseC offset calculation

CaseA
; an instruction sequence follows
CaseB
; an instruction sequence follows
CaseC
; an instruction sequence follows
```

**38.2.10 Miscellaneous instructions**

[Table 755](#) shows the remaining Cortex-M3 instructions:

**Table 755. Miscellaneous instructions**

Mnemonic	Brief description	See
BKPT	Breakpoint	<a href="#">Section 38.2.10.1</a>
CPSID	Change Processor State, Disable Interrupts	<a href="#">Section 38.2.10.2</a>
CPSIE	Change Processor State, Enable Interrupts	<a href="#">Section 38.2.10.2</a>
DMB	Data Memory Barrier	<a href="#">Section 38.2.10.3</a>
DSB	Data Synchronization Barrier	<a href="#">Section 38.2.10.4</a>
ISB	Instruction Synchronization Barrier	<a href="#">Section 38.2.10.5</a>
MRS	Move from special register to register	<a href="#">Section 38.2.10.6</a>
MSR	Move from register to special register	<a href="#">Section 38.2.10.7</a>
NOP	No Operation	<a href="#">Section 38.2.10.8</a>
SEV	Send Event	<a href="#">Section 38.2.10.9</a>
SVC	Supervisor Call	<a href="#">Section 38.2.10.10</a>
WFE	Wait For Event	<a href="#">Section 38.2.10.11</a>
WFI	Wait For Interrupt	<a href="#">Section 38.2.10.12</a>

### 38.2.10.1 BKPT

Breakpoint.

#### 38.2.10.1.1 Syntax

```
BKPT #imm
```

where:

*imm* is an expression evaluating to an integer in the range 0-255 (8-bit value).

#### 38.2.10.1.2 Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

*imm* is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

#### 38.2.10.1.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.1.4 Examples

```
BKPT 0xAB ; Breakpoint with immediate value set to 0xAB (debugger can  
; extract the immediate value by locating it using the PC)
```



**38.2.10.2 CPS**

Change Processor State.

**38.2.10.2.1 Syntax**

*CPSeffect iflags*

where:

*effect* is one of:

- IE Clears the special purpose register.
- ID Sets the special purpose register

*iflags* is a sequence of one or more flags:

- i Set or clear PRIMASK.
- f Set or clear FAULTMASK.

**38.2.10.2.2 Operation**

CPS changes the PRIMASK and FAULTMASK special register values. See [Section 38.3.1.3.6 “Exception mask registers”](#) for more information about these registers.

**38.2.10.2.3 Restrictions**

The restrictions are:

- use CPS only from privileged software, it has no effect if used in unprivileged software.
- CPS cannot be conditional and so must not be used inside an IT block.

**38.2.10.2.4 Condition flags**

This instruction does not change the condition flags.

**38.2.10.2.5 Examples**

```
CPSID i ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f ; Enable interrupts and fault handlers (clear FAULTMASK)
```

### 38.2.10.3 DMB

Data Memory Barrier.

#### 38.2.10.3.1 Syntax

`DMB{cond}`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.10.3.2 Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

#### 38.2.10.3.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.3.4 Examples

```
DMB ; Data Memory Barrier
```

#### 38.2.10.4 DSB

Data Synchronization Barrier.

##### 38.2.10.4.1 Syntax

`DSB{cond}`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

##### 38.2.10.4.2 Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

##### 38.2.10.4.3 Condition flags

This instruction does not change the flags.

##### 38.2.10.4.4 Examples

```
DSB ; Data Synchronisation Barrier
```

### 38.2.10.5 ISB

Instruction Synchronization Barrier.

#### 38.2.10.5.1 Syntax

ISB{*cond*}

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.10.5.2 Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

#### 38.2.10.5.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.5.4 Examples

```
ISB ; Instruction Synchronisation Barrier
```

### 38.2.10.6 MRS

Move the contents of a special register to a general-purpose register.

#### 38.2.10.6.1 Syntax

`MRS{cond} Rd, spec_reg`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rd* is the destination register.

*spec\_reg* can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI\_MAX, FAULTMASK, or CONTROL.

#### 38.2.10.6.2 Operation

Use `MRS` in combination with `MSR` as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

In process swap code, the programmers model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations use `MRS` in the state-saving instruction sequence and `MSR` in the state-restoring instruction sequence.

**Remark:** `BASEPRI_MAX` is an alias of `BASEPRI` when used with the `MRS` instruction.

See [Section 38.2.10.7](#).

#### 38.2.10.6.3 Restrictions

*Rd* must not be SP and must not be PC.

#### 38.2.10.6.4 Condition flags

This instruction does not change the flags.

#### 38.2.10.6.5 Examples

```
MRS R0, PRIMASK ; Read PRIMASK value and write it to R0
```

### 38.2.10.7 MSR

Move the contents of a general-purpose register into the specified special register.

#### 38.2.10.7.1 Syntax

`MSR{cond} spec_reg, Rn`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*Rn* is the source register.

*spec\_reg* can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI\_MAX, FAULTMASK, or CONTROL.

#### 38.2.10.7.2 Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR, see [Table 759 “APSR bit assignments”](#). Privileged software can access all special registers.

In unprivileged software writes to unallocated or execution state bits in the PSR are ignored.

#### Note

When you write to BASEPRI\_MAX, the instruction writes to BASEPRI only if either:

- *Rn* is non-zero and the current BASEPRI value is 0
- *Rn* is non-zero and less than the current BASEPRI value.

See [Section 38.2.10.6](#).

#### 38.2.10.7.3 Restrictions

*Rn* must not be SP and must not be PC.

#### 38.2.10.7.4 Condition flags

This instruction updates the flags explicitly based on the value in *Rn*.

#### 38.2.10.7.5 Examples

```
MSR CONTROL, R1 ; Read R1 value and write it to the CONTROL register
```

### 38.2.10.8 NOP

No Operation.

#### 38.2.10.8.1 Syntax

`NOP{cond}`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.10.8.2 Operation

`NOP` does nothing. `NOP` is not necessarily a time-consuming `NOP`. The processor might remove it from the pipeline before it reaches the execution stage.

Use `NOP` for padding, for example to place the following instruction on a 64-bit boundary.

#### 38.2.10.8.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.8.4 Examples

```
NOP ; No operation
```

### 38.2.10.9 SEV

Send Event.

#### 38.2.10.9.1 Syntax

SEV{*cond*}

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.10.9.2 Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see [Section 38.3.5 “Power management”](#).

#### 38.2.10.9.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.9.4 Examples

```
SEV ; Send Event
```



### 38.2.10.10 SVC

Supervisor Call.

#### 38.2.10.10.1 Syntax

`SVC{cond} #imm`

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

*imm* is an expression evaluating to an integer in the range 0-255 (8-bit value).

#### 38.2.10.10.2 Operation

The `SVC` instruction causes the `SVC` exception.

*imm* is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

#### 38.2.10.10.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.10.4 Examples

```
SVC 0x32 ; Supervisor Call (SVC handler can extract the immediate value
          ; by locating it via the stacked PC)
```

### 38.2.10.11 WFE

Wait For Event.

#### 38.2.10.11.1 Syntax

WFE{*cond*}

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#)

#### 38.2.10.11.2 Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- an exception, unless masked by the exception mask registers or the current priority level
- an exception enters the Pending state, if SEVONPEND in the System Control Register is set
- a Debug Entry request, if Debug is enabled
- an event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information see [Section 38.3.5 “Power management”](#).

#### 38.2.10.11.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.11.4 Examples

```
WFE ; Wait for event
```

### 38.2.10.12 WFI

Wait for Interrupt.

#### 38.2.10.12.1 Syntax

WFI{*cond*}

where:

*cond* is an optional condition code, see [Section 38.2.3.7 “Conditional execution”](#).

#### 38.2.10.12.2 Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- an exception
- a Debug Entry request, regardless of whether Debug is enabled.

#### 38.2.10.12.3 Condition flags

This instruction does not change the flags.

#### 38.2.10.12.4 Examples

```
WFI ; Wait for interrupt
```

## 38.3 ARM Cortex-M3 User Guide: Processor

### 38.3.1 Programmers model

This section describes the Cortex-M3 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

#### 38.3.1.1 Processor mode and privilege levels for software execution

The processor **modes** are:

- Thread mode  
Used to execute application software. The processor enters Thread mode when it comes out of reset.
- Handler mode  
Used to handle exceptions. The processor returns to Thread mode when it has finished exception processing.

The **privilege levels** for software execution are:

- Unprivileged  
The software:
  - has limited access to the `MSR` and `MRS` instructions, and cannot use the `CPS` instruction
  - cannot access the system timer, NVIC, or system control block
  - might have restricted access to memory or peripherals.**Unprivileged software** executes at the unprivileged level.
- Privileged  
The software can use all the instructions and has access to all resources.  
**Privileged software** executes at the privileged level.

In Thread mode, the `CONTROL` register controls whether software execution is privileged or unprivileged, see [Table 765](#). In Handler mode, software execution is always privileged.

Only privileged software can write to the `CONTROL` register to change the privilege level for software execution in Thread mode. Unprivileged software can use the `SVC` instruction to make a **supervisor call** to transfer control to privileged software.

#### 38.3.1.2 Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the **main stack** and the **process stack**, with independent copies of the stack pointer, see [Section 38.3.1.3.2](#).

In Thread mode, the `CONTROL` register controls whether the processor uses the main stack or the process stack, see [Table 765](#). In Handler mode, the processor always uses the main stack. The options for processor operations are:

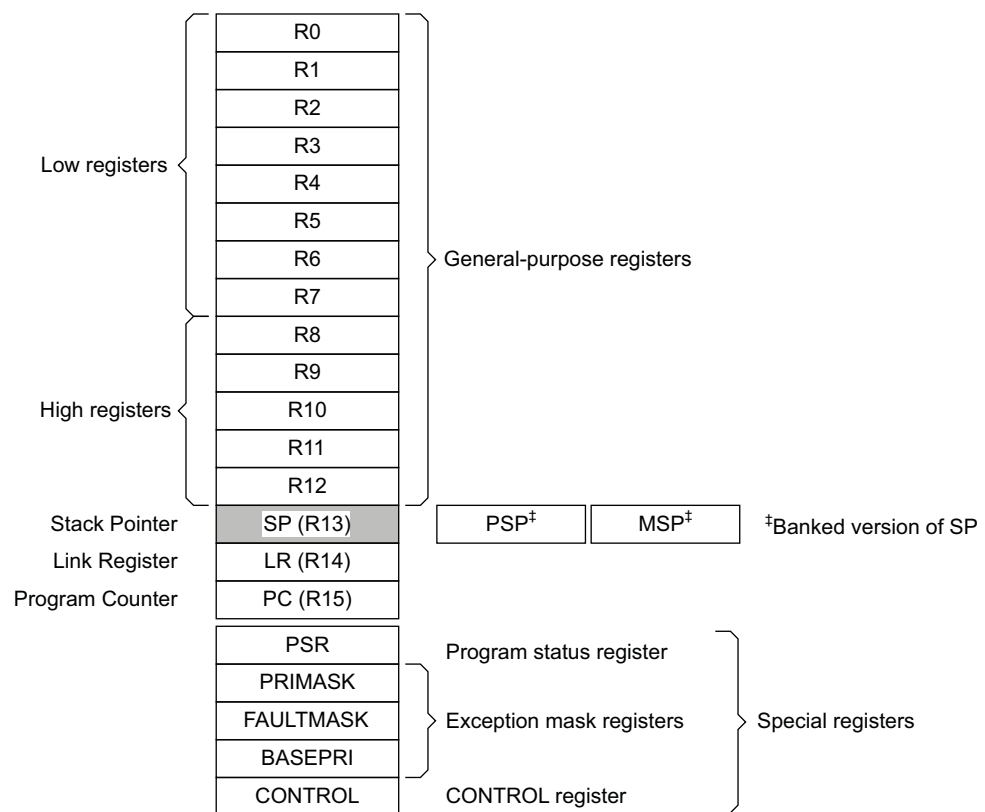
**Table 756. Summary of processor mode, execution privilege level, and stack use options**

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged <sup>[1]</sup>	Main stack or process stack <sup>[1]</sup>
Handler	Exception handlers	Always privileged	Main stack

[1] See [Table 765](#).

### 38.3.1.3 Core registers

The processor core registers are:



**Table 757. Core register set summary**

Name	Type <sup>[1]</sup>	Required privilege <sup>[2]</sup>	Reset value	Description
R0-R12	RW	Either	Undefined	<a href="#">Section 38.3.1.3.1</a>
MSP	RW	Privileged	See description	<a href="#">Section 38.3.1.3.2</a>
PSP	RW	Either	Undefined	<a href="#">Section 38.3.1.3.2</a>
LR	RW	Either	0xFFFFFFFF	<a href="#">Section 38.3.1.3.3</a>
PC	RW	Either	See description	<a href="#">Section 38.3.1.3.4</a>
PSR	RW	Privileged	0x01000000	<a href="#">Section 38.3.1.3.5</a>
ASPR	RW	Either	0x00000000	<a href="#">Table 759</a>
IPSR	RO	Privileged	0x00000000	<a href="#">Table 760</a>

**Table 757. Core register set summary**

Name	Type <sup>[1]</sup>	Required privilege <sup>[2]</sup>	Reset value	Description
EPSR	RO	Privileged	0x01000000	<a href="#">Table 761</a>
PRIMASK	RW	Privileged	0x00000000	<a href="#">Table 762</a>
FAULTMASK	RW	Privileged	0x00000000	<a href="#">Table 763</a>
BASEPRI	RW	Privileged	0x00000000	<a href="#">Table 764</a>
CONTROL	RW	Privileged	0x00000000	<a href="#">Table 765</a>

[1] Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

[2] An entry of Either means privileged and unprivileged software can access the register.

### 38.3.1.3.1 General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

### 38.3.1.3.2 Stack Pointer

The **Stack Pointer** (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

On reset, the processor loads the MSP with the value from address 0x00000000.

- 0 = **Main Stack Pointer** (MSP). This is the reset value.
- 1 = **Process Stack Pointer** (PSP).

### 38.3.1.3.3 Link Register

The **Link Register** (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor loads the LR value 0xFFFFFFFF.

### 38.3.1.3.4 Program Counter

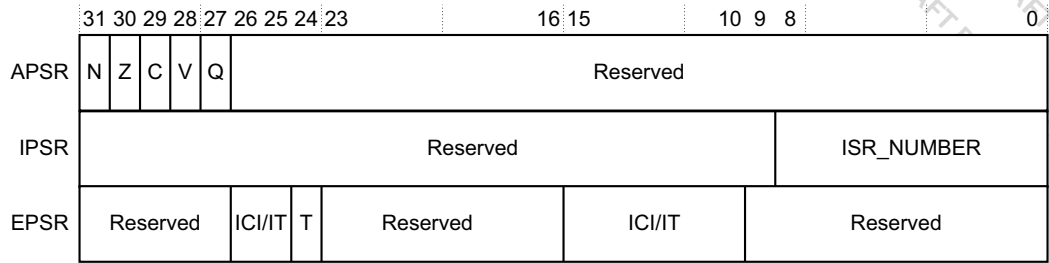
The **Program Counter** (PC) is register R15. It contains the current program address. Bit[0] is always 0 because instruction fetches must be halfword aligned. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004.

### 38.3.1.3.5 Program Status Register

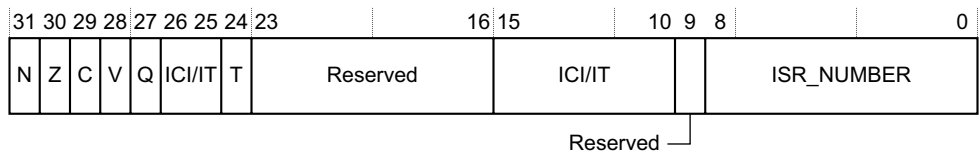
The **Program Status Register** (PSR) combines:

- **Application Program Status Register** (APSR)
- **Interrupt Program Status Register** (IPSR)
- **Execution Program Status Register** (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:



The PSR bit assignments are:



Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR using APSR with the MSR instruction.

The PSR combinations and attributes are:

**Table 758. PSR register combinations**

Register	Type	Combination
PSR	RW <sup>[1][2]</sup>	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	RW <sup>[1]</sup>	APSR and IPSR
EAPSR	RW <sup>[2]</sup>	APSR and EPSR

[1] The processor ignores writes to the IPSR bits.

[2] Reads of the EPSR bits return zero, and the processor ignores writes to the these bits

See the instruction descriptions [Section 38.2.10.6 “MRS”](#) and [Section 38.2.10.7 “MSR”](#) for more information about how to access the program status registers.

**Application Program Status Register:** The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in [Table 757](#) for its attributes. The bit assignments are:

**Table 759. APSR bit assignments**

Bits	Name	Function
[31]	N	Negative or less than flag: 0 = operation result was positive, zero, greater than, or equal 1 = operation result was negative or less than.
[30]	Z	Zero flag: 0 = operation result was not zero 1 = operation result was zero.
[29]	C	Carry or borrow flag: 0 = add operation did not result in a carry bit or subtract operation resulted in a borrow bit 1 = add operation resulted in a carry bit or subtract operation did not result in a borrow bit.
[28]	V	Overflow flag: 0 = operation did not result in an overflow 1 = operation resulted in an overflow.
[27]	Q	Sticky saturation flag: 0 = indicates that saturation has not occurred since reset or since the bit was last cleared to zero 1 = indicates when an <i>SSAT</i> or <i>USAT</i> instruction results in saturation. This bit is cleared to zero by software using an <i>MRS</i> instruction.
[26:0]	-	Reserved.

**Interrupt Program Status Register:** The IPSR contains the exception type number of the current **Interrupt Service Routine** (ISR). See the register summary in [Table 757](#) for its attributes. The bit assignments are:



**Table 760. IPSR bit assignments**

Bits	Name	Function
[31:9]	-	Reserved
[8:0]	ISR_NUMBER	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = Hard fault 4 = Memory management fault 5 = Bus fault 6 = Usage fault 7-10 = Reserved 11 = SVCcall 12 = Reserved for Debug 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 17 = IRQ1, first device specific interrupt . . 255 = IRQ243 (last implemented interrupt depends on device) see <a href="#">Section 38.3.3.2</a> for more information.

**Execution Program Status Register:** The EPSR contains the Thumb state bit, and the execution state bits for either the:

- **If-Then (IT)** instruction
- **Interruptible-Continuable Instruction (ICI)** field for an interrupted load multiple or store multiple instruction.

See the register summary in [Table 757](#) for the EPSR attributes. The bit assignments are:

**Table 761. EPSR bit assignments**

Bits	Name	Function
[31:27]	-	Reserved.
[26:25], [15:10]	ICI	Interruptible-continuable instruction bits, see <a href="#">Section .</a>
[26:25], [15:10]	IT	Indicates the execution state bits of the IT instruction, see <a href="#">Section 38.2.9.3 "IT"</a> .
[24]	T	Always set to 1.
[23:16]	-	Reserved.
[9:0]	-	Reserved.

Attempts to read the EPSR directly through application software using the `MSR` instruction always return zero. Attempts to write the EPSR using the `MSR` instruction in application software are ignored. Fault handlers can examine EPSR value in the stacked PSR to indicate the operation that is at fault. See [Section 38.3.3.7](#)

**Interruptible-continuable instructions:** When an interrupt occurs during the execution of an LDM or STM instruction, the processor:

After servicing the interrupt, the processor:

- stops the load multiple or store multiple instruction operation temporarily.
- stores the next register operand in the multiple operation to EPSR bits[15:12].
- returns to the register pointed to by bits[15:12].
- resumes execution of the multiple load or store instruction.

When the EPSR holds ICI execution state, bits[26:25,11:10] are zero.

**If-Then block:** The If-Then block contains up to four instructions following a 16-bit IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See [Section 38.2.9.3 “IT”](#) for more information.

### 38.3.1.3.6 Exception mask registers

The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks.

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK. See [Section 38.2.10.6 “MRS”](#), [Section 38.2.10.7 “MSR”](#), and [Section 38.2.10.2 “CPS”](#) for more information.

**Priority Mask Register:** The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in [Table 757](#) for its attributes. The bit assignments are shown in [Table 762](#).

**Table 762. PRIMASK register bit assignments**

Bits	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	0 = no effect 1 = prevents the activation of all exceptions with configurable priority.

**Fault Mask Register:** The FAULTMASK register prevents activation of all exceptions except for **Non-Maskable Interrupt (NMI)**. See the register summary in [Table 757](#) for its attributes. The bit assignments are shown in [Table 763](#).

**Table 763. FAULTMASK register bit assignments**

Bits	Name	Function
[31:1]	-	Reserved
[0]	FAULTMASK	0 = no effect 1 = prevents the activation of all exceptions except for NMI.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

**Base Priority Mask Register:** The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with same or lower priority level as the BASEPRI value. See the register summary in [Table 757](#) for its attributes. The bit assignments are shown in [Table 764](#).

**Table 764. BASEPRI register bit assignments**

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI <sup>[1]</sup>	Priority mask bits: 0x0000 = no effect Nonzero = defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

[1] This field is similar to the priority fields in the interrupt priority registers. The processor implements only bits[7:M] of this field, bits[M-1:0] read as zero and ignore writes. The value of M depends on the specific device. See [Section 38.4.2.7 "Interrupt Priority Registers"](#) for more information. Remember that higher priority field values correspond to lower exception priorities.

### 38.3.1.3.7 CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode. See the register summary in [Table 757](#) for its attributes. The bit assignments are shown in [Table 765](#).

**Table 765. CONTROL register bit assignments**

Bits	Name	Function
[31:2]	-	Reserved
[1]	Active stack pointer	Defines the current stack: 0 = MSP is the current stack pointer 1 = PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes.
[0]	Thread mode privilege level	Defines the Thread mode privilege level: 0 = privileged 1 = unprivileged.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms update the CONTROL register.

In an OS environment, ARM recommends that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, use the MSR instruction to set the Active stack pointer bit to 1, see [Section 38.2.10.7 "MSR"](#).

**Remark:** When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer. See [Section 38.2.10.5 "ISB"](#)

### 38.3.1.4 Exceptions and interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the **Nested Vectored Interrupt Controller** (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See [Section 38.3.3.7.1](#) and [Section 38.3.3.7.2](#) for more information.

The NVIC registers control interrupt handling. See [Section 38.4.2 “Nested Vectored Interrupt Controller”](#) for more information.

### 38.3.1.5 Data types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- supports 64-bit data transfer instructions.
- manages all data memory accesses as little-endian. See [Section 38.3.2.1](#) for more information.

### 38.3.1.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M3 microcontroller system, the **Cortex Microcontroller Software Interface Standard (CMSIS)** defines:

- a common way to:
  - access peripheral registers
  - define exception vectors
- the names of:
  - the registers of the core peripherals
  - the core exception vectors
- a device-independent interface for RTOS kernels, including a debug channel.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M3 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

CMSIS simplifies software development by enabling the reuse of template code and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

**Remark:** This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

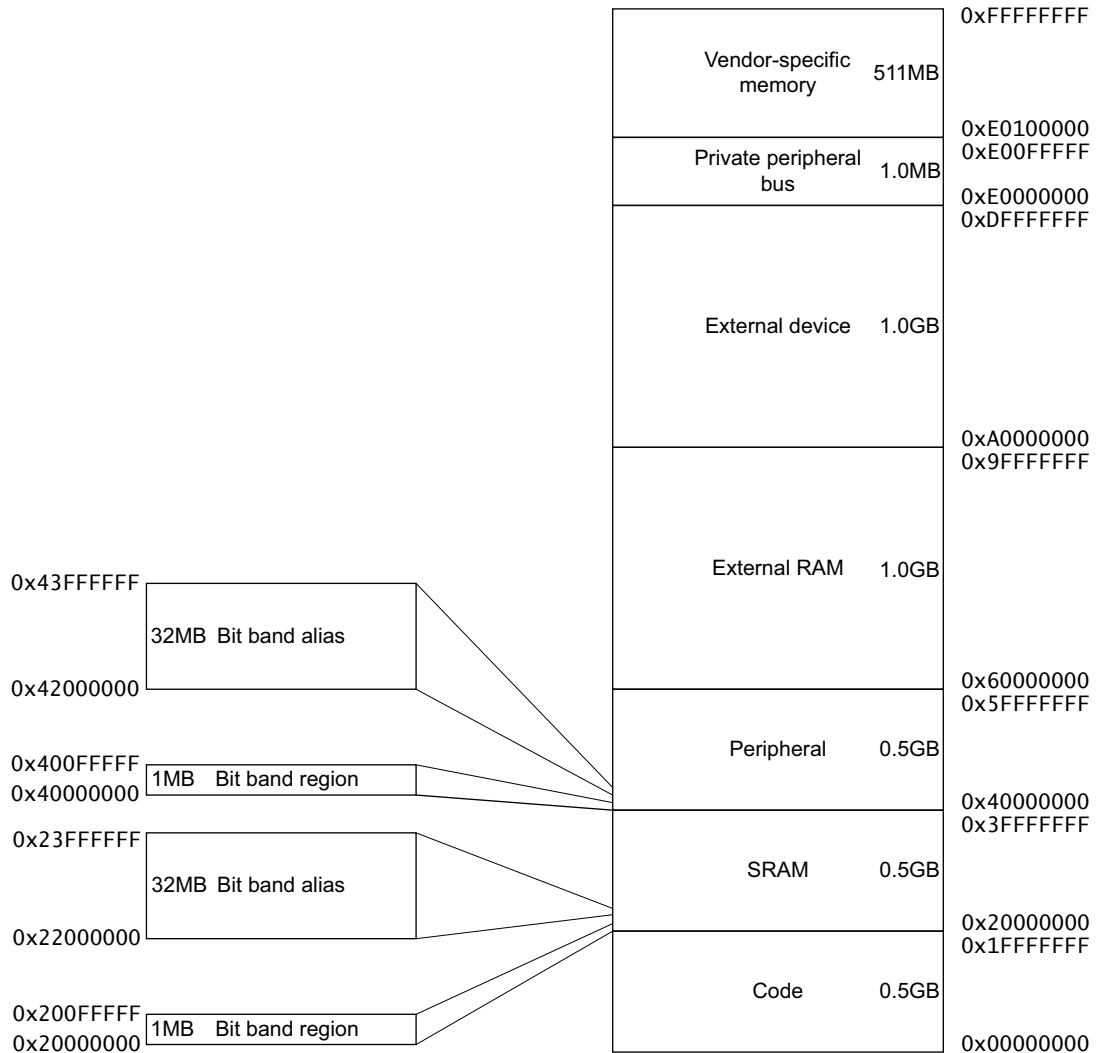
The following sections give more information about the CMSIS:

- [Section 38.3.5.4](#)
- [Section 38.2.2 “Intrinsic functions”](#)
- [Section 38.4.2.1 “The CMSIS mapping of the Cortex-M3 NVIC registers”](#)



### 38.3.2 Memory model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4GB of addressable memory. The memory map is:



The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data, see [Section 38.3.2.5](#).

The processor reserves regions of the **Private peripheral bus** (PPB) address range for core peripheral registers, see [Section 38.4.1 "About the Cortex-M3 peripherals"](#).

#### 38.3.2.1 Memory regions, types and attributes

The memory map and the programming of the MPU split the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

- Normal: The processor can re-order transactions for efficiency, or perform speculative reads.
- Device: The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.
- Strongly-ordered: The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include.

- **Shareable**

For a shareable memory region, the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller.

Strongly-ordered memory is always shareable.

If multiple bus masters can access a non-shareable memory region, software must ensure data coherency between the bus masters.

- **Execute Never (XN)**

Means the processor prevents instruction accesses. Any attempt to fetch an instruction from an XN region causes a memory management fault exception.

### 38.3.2.2 Memory system ordering of memory accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions, see [Section 38.3.2.4](#).

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses caused by two instructions is:

A1 \ A2	Normal access	Device access		Strongly-ordered access
		Non-shareable	Shareable	
Normal access	-	-	-	-
Device access, non-shareable	-	<	-	<
Device access, shareable	-	-	<	<
Strongly-ordered access	-	<	<	<

Where:

‘—’ means that the memory system does not guarantee the ordering of the accesses.

'<' means that accesses are observed in program order, that is, A1 is always observed before A2.

### 38.3.2.3 Behavior of memory accesses

The behavior of accesses to each region in the memory map is:

**Table 766. Memory access behavior**

Address range	Memory region	Memory type	XN	Description
0x00000000 - 0x1FFFFFFF	Code	Normal <sup>[1]</sup>	-	Executable region for program code. You can also put data here.
0x20000000 - 0x3FFFFFFF	SRAM	Normal <sup>[1]</sup>	-	Executable region for data. You can also put code here. This region includes bit band and bit band alias areas, see <a href="#">Table 767</a> .
0x40000000 - 0x5FFFFFFF	Peripheral	Device <sup>[1]</sup>	XN <sup>[1]</sup>	This region includes bit band and bit band alias areas, see <a href="#">Table 768</a> .
0x60000000 - 0x9FFFFFFF	External RAM	Normal <sup>[1]</sup>	-	Executable region for data.
0xA0000000 - 0xDFFFFFFF	External device	Device <sup>[1]</sup>	XN <sup>[1]</sup>	External Device memory
0xE0000000 - 0xE0FFFFFF	Private Peripheral Bus	Strongly-ordered <sup>[1]</sup>	XN <sup>[1]</sup>	This region includes the NVIC, System timer, and system control block.
0xE0100000 - 0xFFFFFFFF	Vendor-specific device	Device <sup>[1]</sup>	XN <sup>[1]</sup>	Not used for NXP devices.

[1] See [Section 38.3.2.1](#) for more information.

The Code, SRAM, and external RAM regions can hold programs. However, the most efficient access to programs is from the Code region. This is because the processor has separate buses that enable instruction fetches and data accesses to occur simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see [Section 38.4.5 "Memory protection unit"](#).

### 38.3.2.4 Software ordering of memory accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- the processor has multiple bus interfaces
- memory or devices in the memory map have different wait states
- some memory accesses are buffered or speculative.

[Section 38.3.2.2](#) describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:



- DMB  
The **Data Memory Barrier** (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See [Section 38.2.10.3 “DMB”](#).
- DSB  
The **Data Synchronization Barrier** (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See [Section 38.2.10.4 “DSB”](#).
- ISB  
The **Instruction Synchronization Barrier** (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See [Section 38.2.10.5 “ISB”](#).

Use memory barrier instructions in, for example:

Memory accesses to Strongly-ordered memory, such as the system control block, do not require the use of DMB instructions.

- MPU programming:
  - Use a DSB instruction to ensure the effect of the MPU takes place immediately at the end of context switching.
  - Use an ISB instruction to ensure the new MPU setting takes effect immediately after programming the MPU region or regions, if the MPU configuration code was accessed using a branch or call. If the MPU configuration code is entered using exception mechanisms, then an ISB instruction is not required.
- Vector table. If the program changes an entry in the vector table, and then enables the corresponding exception, use a DMB instruction between the operations. This ensures that if the exception is taken immediately after being enabled the processor uses the new exception vector.
- Self-modifying code. If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. This ensures subsequent instruction execution uses the updated program.
- Memory map switching. If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. This ensures subsequent instruction execution uses the updated memory map.
- Dynamic exception priority change. When an exception priority has to change when the exception is pending or active, use DSB instructions after the change. This ensures the change takes effect on completion of the DSB instruction.
- Using a semaphore in multi-master system. If the system contains more than one bus master, for example, if another processor is present in the system, each processor must use a DMB instruction after any semaphore instructions, to ensure other bus masters see the memory transactions in the order in which they were executed.

### 38.3.2.5 Bit-banding

A bit-band region maps each word in a **bit-band alias** region to a single bit in the **bit-band region**. The bit-band regions occupy the lowest 1MB of the SRAM and peripheral memory regions.

The memory map has two 32MB alias regions that map to two 1MB bit-band regions:

- accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region, as shown in [Table 767](#)
- accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region, as shown in [Table 768](#).

**Table 767. SRAM memory bit-banding regions**

Address range	Memory region	Instruction and data accesses
0x20000000 - 0x200FFFFFF	SRAM bit-band region	Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias.
0x22000000 - 0x23FFFFFF0	SRAM bit-band alias	Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped.

**Table 768. Peripheral memory bit-banding regions**

Address range	Memory region	Instruction and data accesses
0x40000000 - 0x400FFFFFF	Peripheral bit-band alias	Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias.
0x42000000 - 0x44FFFFFF	Peripheral bit-band region	Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted.

**Remark:** A word access to the SRAM or peripheral bit-band alias regions map to a single bit in the SRAM or peripheral bit-band region.

The following formula shows how the alias region maps onto the bit-band region:

$$\text{bit\_word\_offset} = (\text{byte\_offset} \times 32) + (\text{bit\_number} \times 4)$$

$$\text{bit\_word\_addr} = \text{bit\_band\_base} + \text{bit\_word\_offset}$$

where:

- `Bit_word_offset` is the position of the target bit in the bit-band memory region.
- `Bit_word_addr` is the address of the word in the alias memory region that maps to the targeted bit.
- `Bit_band_base` is the starting address of the alias region.
- `Byte_offset` is the number of the byte in the bit-band region that contains the targeted bit.
- `Bit_number` is the bit position, 0-7, of the targeted bit.

[Figure 179](#) shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at `0x23FFFFFF0` maps to bit[0] of the bit-band byte at `0x200FFFFFF`:  

$$0x23FFFFFF0 = 0x22000000 + (0xFFFF * 32) + (0 * 4).$$

- The alias word at 0x23FFFFFFC maps to bit[7] of the bit-band byte at 0x200FFFFFF:  $0x23FFFFFFC = 0x22000000 + (0xFFFF * 32) + (7 * 4)$ .
- The alias word at 0x22000000 maps to bit[0] of the bit-band byte at 0x20000000:  $0x22000000 = 0x22000000 + (0 * 32) + (0 * 4)$ .
- The alias word at 0x2200001C maps to bit[7] of the bit-band byte at 0x20000000:  $0x2200001C = 0x22000000 + (0 * 32) + (7 * 4)$ .

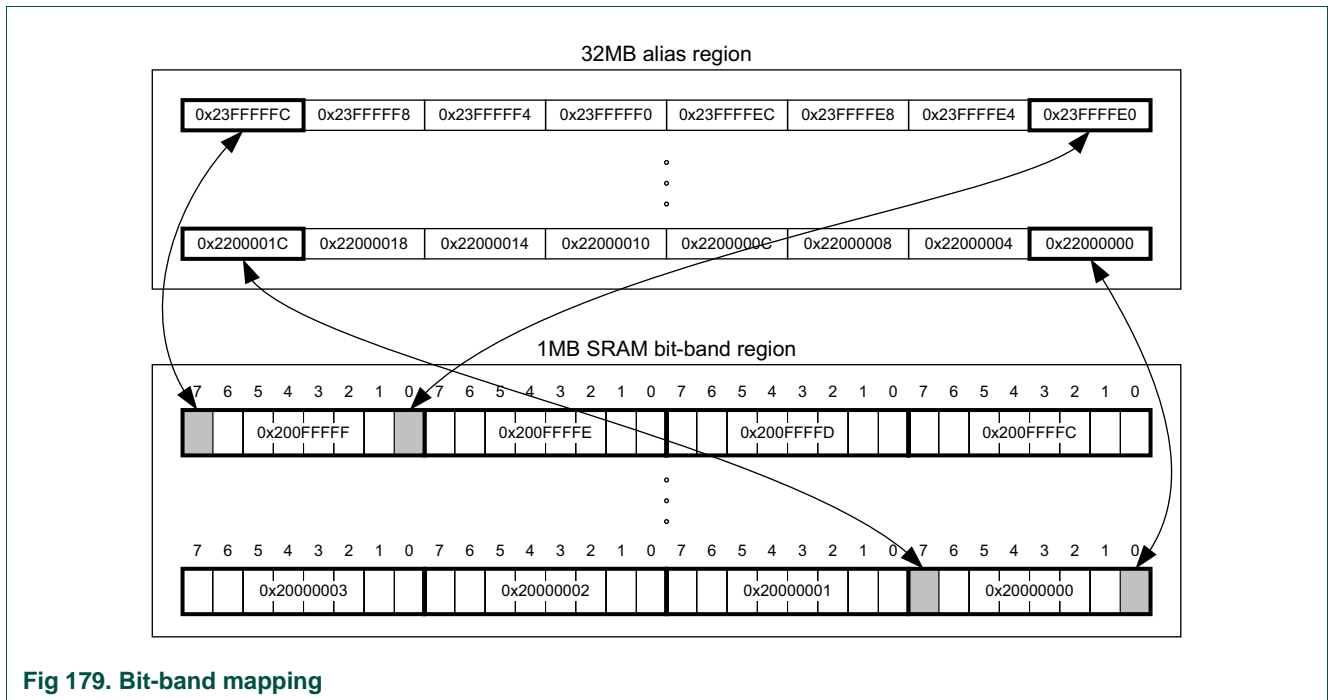


Fig 179. Bit-band mapping

### 38.3.2.5.1 Directly accessing an alias region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit[0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit[0] set to 1 writes a 1 to the bit-band bit, and writing a value with bit[0] set to 0 writes a 0 to the bit-band bit.

Bits[31:1] of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

Reading a word in the alias region:

- 0x00000000 indicates that the targeted bit in the bit-band region is set to zero
- 0x00000001 indicates that the targeted bit in the bit-band region is set to 1

### 38.3.2.5.2 Directly accessing a bit-band region

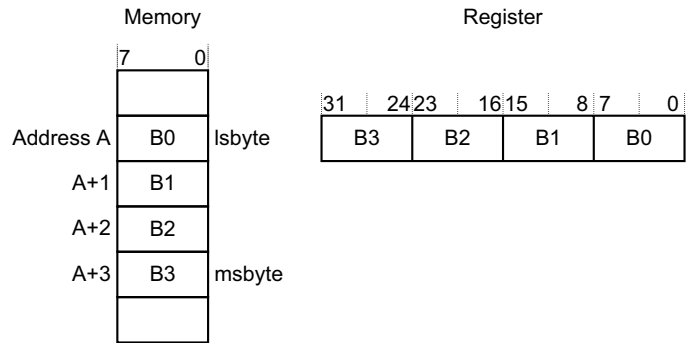
[Section 38.3.2.3](#) describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

**38.3.2.6 Memory endianness**

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. [Section 38.3.2.6.1](#) describes how words of data are stored in memory.

**38.3.2.6.1 Little-endian format**

In little-endian format, the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte. For example:



**38.3.2.7 Synchronization primitives**

The Cortex-M3 instruction set includes pairs of **synchronization primitives**. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

- A Load-Exclusive instruction  
Used to read the value of a memory location, requesting exclusive access to that location.
- A Store-Exclusive instruction  
Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:
  - 0: it indicates that the thread or process gained exclusive access to the memory, and the write succeeds,
  - 1: it indicates that the thread or process did not gain exclusive access to the memory, and no write is performed,

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- the word instructions `LDREX` and `STREX`
- the halfword instructions `LDREXH` and `STREXH`
- the byte instructions `LDREXB` and `STREXB`.

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform a guaranteed read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Update the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location, and tests the returned status bit. If this bit is:
  - 0: The read-modify-write completed successfully,
  - 1: No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the read-modify-write sequence,

Software can use the synchronization primitives to implement a semaphores as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from step 2 indicates that the Store-Exclusive succeeded then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M3 includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction.

The processor removes its exclusive access tag if:

- It executes a CLREX instruction
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs. This means the processor can resolve semaphore conflicts between different threads.

For more information about the synchronization primitive instructions, see [Section 38.2.4.8 “LDREX and STREX”](#) and [Section 38.2.4.9](#).

### 38.3.2.8 Programming hints for the synchronization primitives

ANSI C cannot directly generate the exclusive access instructions. Some C compilers provide intrinsic functions for generation of these instructions:

**Table 769. C compiler intrinsic functions for exclusive access instructions**

Instruction	Intrinsic function
LDREX, LDREXH, or LDREXB	<code>unsigned int __ldrex(volatile void *ptr)</code>
STREX, STREXH, or STREXB	<code>int __strex(unsigned int val, volatile void *ptr)</code>
CLREX	<code>void __clrex(void)</code>

The actual exclusive access instruction generated depends on the data type of the pointer passed to the intrinsic function. For example, the following C code generates the require LDREXB operation:

```
__ldrex((volatile char *) 0xFF);
```

### 38.3.3 Exception model

This section describes the exception model.

#### 38.3.3.1 Exception states

Each exception is in one of the following states:

- Inactive  
The exception is not active and not pending.
- Pending  
The exception is waiting to be serviced by the processor.  
An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- Active  
An exception that is being serviced by the processor but has not completed.  
**Remark:** An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.
- Active and pending  
The exception is being serviced by the processor and there is a pending exception from the same source.

#### 38.3.3.2 Exception types

The exception types are:

- Reset  
Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.
- NMI  
A **NonMaskable Interrupt** (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be:
  - masked or prevented from activation by any other exception
  - preempted by any exception other than Reset.
- Hard fault  
A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard faults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
- Memory management fault

A memory management fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to **Execute Never** (XN) memory regions, even if the MPU is disabled.

- Bus fault

A bus fault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

- Usage fault

A usage fault is an exception that occurs because of a fault related to instruction execution. This includes:

- an undefined instruction
- an illegal unaligned access
- invalid state on instruction execution
- an error on exception return.

The following can cause a usage fault when the core is configured to report them:

- an unaligned address on word and halfword memory access
- division by zero.

- SVCall

A **supervisor call** (SVC) is an exception that is triggered by the *svc* instruction. In an OS environment, applications can use *svc* instructions to access OS kernel functions and device drivers.

- PendSV

PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

- SysTick

A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

- Interrupt (IRQ)

An interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

**Table 770. Properties of the different exception types**

Exception number <sup>[1]</sup>	IRQ number <sup>[1]</sup>	Exception type	Priority	Vector address or offset <sup>[2]</sup>	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable <sup>[3]</sup>	0x00000010	Synchronous

**Table 770. Properties of the different exception types**

Exception number <sup>[1]</sup>	IRQ number <sup>[1]</sup>	Exception type	Priority	Vector address or offset <sup>[2]</sup>	Activation
5	-11	Bus fault	Configurable <sup>[3]</sup>	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable <sup>[3]</sup>	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCcall	Configurable <sup>[3]</sup>	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable <sup>[3]</sup>	0x00000038	Asynchronous
15	-1	SysTick	Configurable <sup>[3]</sup>	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable <sup>[4]</sup>	0x00000040 and above <sup>[5]</sup>	Asynchronous

[1] To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see [Table 760](#).

[2] See [Section 38.3.3.4](#) for more information.

[3] See [Section 38.4.3.9 “System Handler Priority Registers”](#).

[4] See [Section 38.4.2.7 “Interrupt Priority Registers”](#).

[5] Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that [Table 770](#) shows as having configurable priority, see:

- [Section 38.4.3.10 “System Handler Control and State Register”](#)
- [Section 38.4.2.3 “Interrupt Clear-enable Registers”](#).

For more information about hard faults, memory management faults, bus faults, and usage faults, see [Section 38.3.4](#).

### 38.3.3.3 Exception handlers

The processor handles exceptions using:

- Interrupt Service Routines (ISRs)  
Interrupts IRQ0 and up are the exceptions handled by ISRs.
- Fault handlers  
Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers.
- System handlers  
NMI, PendSV, SVCcall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.



38.3.3.4 Vector table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. [Figure 180](#) shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code. Note that the upper limit of the IRQ number is device dependent.

Exception number	IRQ number	Offset	Vector
127	111	0x1FC	IRQ111
.	.	.	.
.	.	.	.
.	.	.	.
18		0x004C	IRQ2
17	2	0x0048	IRQ1
16	1	0x0044	IRQ0
15	0	0x0040	Systick
14	-1	0x003C	PendSV
13	-2	0x0038	Reserved
12			Reserved for debug
11			SVCall
10	-5	0x002C	Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value.

**Fig 180. Vector table**

On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000080 to 0x3FFFFFF80, see [Section 38.4.3.5 “Vector Table Offset Register”](#).

38.3.3.5 Exception priorities

As [Table 770](#) shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, Hard fault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see:

- [Section 38.4.3.9 “System Handler Priority Registers”](#)
- [Section 38.4.2.7 “Interrupt Priority Registers”](#).

**Remark:** Configurable priority values are in the range 0 to 31. This means that the Reset, Hard fault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

### 38.3.3.6 Interrupt priority grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler,

- an upper field that defines the **group priority**
- a lower field that defines a **subpriority** within the group.

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see [Section 38.4.3.6 “Application Interrupt and Reset Control Register”](#).

### 38.3.3.7 Exception entry and return

Descriptions of exception handling use the following terms:

- **Preemption**  
When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See [Section 38.3.3.6](#) for more information about preemption by an interrupt. When one exception preempts another, the exceptions are called nested exceptions. See [Section 38.3.3.7.1](#) more information.
- **Return**  
This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See [Section 38.3.3.7.2](#) for more information.

- Tail-chaining

This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.

- Late-arriving

This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

#### 38.3.3.7.1 Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers, see [Section 38.3.1.3.6](#). An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred as **stacking** and the structure of eight data words is referred as **stack frame**. The stack frame contains the following information:

- R0-R3, R12
- Return address
- PSR
- LR.

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. Unless stack alignment is disabled, the stack frame is aligned to a double-word address. If the STKALIGN bit of the **Configuration Control Register (CCR)** is set to 1, stack align adjustment is performed during stacking.

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC\_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the was processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

### 38.3.3.7.2 Exception return

Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC\_RETURN value into the PC:

- a POP instruction that includes the PC
- a BX instruction with any register.
- an LDR or LDM instruction with the PC as the destination.

EXC\_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest four bits of this value provide information on the return stack and processor mode. [Table 771](#) shows the EXC\_RETURN[3:0] values with a description of the exception return behavior.

The processor sets EXC\_RETURN bits[31:4] to 0xFFFFFFFF. When this value is loaded into the PC it indicates to the processor that the exception is complete, and the processor initiates the exception return sequence.

**Table 771. Exception return behavior**

EXC_RETURN[3:0]	Description
bXXX0	Reserved.
b0001	Return to Handler mode. Exception return gets state from MSP. Execution uses MSP after return.
b0011	Reserved.
b01X1	Reserved.

**Table 771. Exception return behavior**

<b>EXC_RETURN[3:0]</b>	<b>Description</b>
b1001	Return to Thread mode. Exception return gets state from MSP. Execution uses MSP after return.
b1101	Return to Thread mode. Exception return gets state from PSP. Execution uses PSP after return.
b1X11	Reserved.

### 38.3.4 Fault handling

Faults are a subset of the exceptions, see [Section 38.3.3](#). The following generate a fault:

- a bus error on:
  - an instruction fetch or vector table load
  - a data access
- an internally-detected error such as an undefined instruction or an attempt to change state with a BX instruction
- attempting to execute an instruction from a memory region marked as **Non-Executable** (XN)
- an MPU fault because of a privilege violation or an attempt to access an unmanaged region

#### 38.3.4.1 Fault types

[Table 772](#) shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See [Section 38.4.3.11 “Configurable Fault Status Register”](#) for more information about the fault status registers.

**Table 772. Faults**

Fault	Handler	Bit name	Fault status register
Bus error on a vector read	Hard fault	VECTTBL	<a href="#">Section 38.4.3.12 “Hard Fault Status Register”</a>
Fault escalated to a hard fault		FORCED	
MPU mismatch:	Memory management fault	-	-
on instruction access		IACCVIOL <sup>[1]</sup>	<a href="#">Section 38.4.3.13 “Memory Management Fault Address Register”</a>
on data access		DACCVIOL	
during exception stacking		MSTKERR	
during exception unstacking	MUNSKERR		
Bus error:	Bus fault	-	-
during exception stacking		STKERR	<a href="#">Section 38.4.3.14 “Bus Fault Address Register”</a>
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	Usage fault	NOCP	<a href="#">Section 38.4.3.11.3 “Usage Fault Status Register”</a>
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state <sup>[2]</sup>		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

[1] Occurs on an access to an XN region even if the MPU is disabled.

[2] Attempting to use an instruction set other than the Thumb instruction set.

### 38.3.4.2 Fault escalation and hard faults

All faults exceptions except for hard fault have configurable exception priority, see [Section 38.4.3.9 “System Handler Priority Registers”](#). Software can disable execution of the handlers for these faults, see [Section 38.4.3.10 “System Handler Control and State Register”](#).

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler. as described in [Section 38.3.3](#).

In some situations, a fault with configurable priority is treated as a hard fault. This is called **priority escalation**, and the fault is described as **escalated to hard fault**. Escalation to hard fault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to hard fault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a bus fault occurs during a stack push when entering a bus fault handler, the bus fault does not escalate to a hard fault. This means that if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

**Remark:** Only Reset and NMI can preempt the fixed priority hard fault. A hard fault can preempt any exception other than Reset, NMI, or another hard fault.

### 38.3.4.3 Fault status registers and fault address registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in [Table 773](#).

**Table 773. Fault status and fault address registers**

Handler	Status register name	Address register name	Register description
Hard fault	HFSR	-	<a href="#">Section 38.4.3.12 “Hard Fault Status Register”</a>
Memory management fault	MMFSR	MMFAR	<a href="#">Section 38.4.3.13 “Memory Management Fault Address Register”</a> <a href="#">Section 38.4.3.11.1 “Memory Management Fault Status Register”</a>
Bus fault	BFSR	BFAR	<a href="#">Section 38.4.3.11.2 “Bus Fault Status Register”</a> <a href="#">Section 38.4.3.14 “Bus Fault Address Register”</a>
Usage fault	UFSR	-	<a href="#">Section 38.4.3.11.3 “Usage Fault Status Register”</a>

#### 38.3.4.4 Lockup

The processor enters a lockup state if a hard fault occurs when executing the NMI or hard fault handlers. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until either:

- it is reset
- an NMI occurs.

**Remark:** If lockup state occurs from the NMI handler a subsequent NMI does not cause the processor to leave lockup state.



### 38.3.5 Power management

**Note:** NXP devices based on the Cortex-M3 processor, including the LPC178x/177x, support additional reduced power modes. See [Section 4.7 “Power control”](#) for information on all available reduced power modes.

The Cortex-M3 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock
- Deep sleep mode stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see [Section 38.4.3.7 “System Control Register”](#). For more information about the behavior of the sleep modes see [Section 4.7 “Power control”](#).

This section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

#### 38.3.5.1 Entering sleep mode

This section describes the mechanisms software can use to put the processor into sleep mode.

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

##### 38.3.5.1.1 Wait for interrupt

The **wait for interrupt** instruction, `WFI`, causes immediate entry to sleep mode. When the processor executes a `WFI` instruction it stops executing instructions and enters sleep mode. See [Section 38.2.10.12 “WFI”](#) for more information.

##### 38.3.5.1.2 Wait for event

**Note:** LPC178x/177x devices based on the Cortex-M3 processor do not implement external events.

The **wait for event** instruction, `WFE`, causes entry to sleep mode conditional on the value of an one-bit event register. When the processor executes a `WFE` instruction, it checks this register:

- if the register is 0 the processor stops executing instructions and enters sleep mode
- if the register is 1 the processor clears the register to 0 and continues executing instructions without entering sleep mode.

See [Section 38.2.10.11 “WFE”](#) for more information.

If the event register is 1, this indicate that the processor must not enter sleep mode on execution of a `WFE` instruction. Typically, this is because an external event signal is asserted, or a processor in the system has executed an `SEV` instruction, see [Section 38.2.10.9 “SEV”](#). Software cannot access this register directly.

### 38.3.5.1.3 Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of an exception handler it returns to Thread mode and immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

### 38.3.5.2 Wakeup from sleep mode

The conditions for the processor to wakeup depend on the mechanism that cause it to enter sleep mode.

#### 38.3.5.2.1 Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK see [Section 38.3.1.3.6](#).

#### 38.3.5.2.2 Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see [Section 38.4.3.7 "System Control Register"](#).

### 38.3.5.3 The Wakeup Interrupt Controller

The **Wakeup Interrupt Controller** (WIC) is a peripheral that can detect an interrupt and wake the processor from deep sleep mode, Power-down, or Deep Power-down modes. The WIC is enabled only when the DEEPSLEEP bit in the SCR is set to 1, see [Section 38.4.3.7 "System Control Register"](#).

**Remark:** NXP microcontrollers extend the number of reduced power modes beyond what is directly supported by the Cortex-M3. Details all available reduced power modes and wakeup possibilities on the LPC178x/177x can be found in [Section 4.7 "Power control"](#).

The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

When the WIC is enabled and the processor enters deep sleep mode or Power-down mode, the power management unit in the system can power down most of the Cortex-M3 processor. This has the side effect of stopping the SysTick timer. When the WIC receives an interrupt, it takes a number of clock cycles to wakeup the processor and restore its state, before it can process the interrupt. This means interrupt latency is increased in deep sleep mode. Wakeup from Power-down mode requires startup of many other portions of the device, and takes longer. Wakeup from Deep Power-down mode adds time to re-establish the on-chip regulator voltage as well.

**Remark:** If the processor detects a connection to a debugger it disables the WIC.

#### 38.3.5.4 Power management programming hints

ANSI C cannot directly generate the `WFI` and `WFE` instructions. The CMSIS provides the following intrinsic functions for these instructions:

```
void __WFE(void) // Wait for Event
```

```
void __WFI(void) // Wait for Interrupt
```

## 38.4 ARM Cortex-M3 User Guide: Peripherals

### 38.4.1 About the Cortex-M3 peripherals

The address map of the **Private peripheral bus** (PPB) is:

**Table 774. Core peripheral register regions**

Address	Core peripheral	Description
0xE000E008 - 0xE000E00F	System control block	<a href="#">Table 785 “Summary of the system control block registers”</a>
0xE000E010 - 0xE000E01F	System timer	<a href="#">Table 805 “System timer registers summary”</a>
0xE000E100 - 0xE000E4EF	Nested Vectored Interrupt Controller	<a href="#">Table 775 “NVIC register summary”</a>
0xE000ED00 - 0xE000ED3F	System control block	<a href="#">Table 785 “Summary of the system control block registers”</a>
0xE000ED90 - 0xE000EDB8	Memory Protection Unit	<a href="#">Table 811 “MPU registers summary”</a>
0xE000EF00 - 0xE000EF03	Nested Vectored Interrupt Controller	<a href="#">Table 775 “NVIC register summary”</a>

In register descriptions:

- the register **type** is described as follows:
  - **RW**: Read and write.
  - **RO**: Read-only.
  - **WO**: Write-only.
- the **required privilege** gives the privilege level required to access the register, as follows:
  - **Privileged**: Only privileged software can access the register
  - **Unprivileged**: Both unprivileged and privileged software can access the register.

### 38.4.2 Nested Vectored Interrupt Controller

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- Up to 112 interrupts. The number of interrupts implemented is device dependent.
- A programmable priority level of 0 to 31 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external **Non-maskable interrupt** (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

**Table 775. NVIC register summary**

Address	Name	Type	Required privilege	Reset value	Description
0xE000E100 - 0xE000E10C	ISER0 - ISER3	RW	Privileged	0x00000000	<a href="#">Table 777</a>
0xE000E180 - 0xE000E18C	ICER0 - ICER3	RW	Privileged	0x00000000	<a href="#">Table 778</a>
0xE000E200 - 0xE000E20C	ISPR0 - ISPR3	RW	Privileged	0x00000000	<a href="#">Table 779</a>
0xE000E280 - 0xE000E28C	ICPR0 - ICPR3	RW	Privileged	0x00000000	<a href="#">Table 780</a>
0xE000E300 - 0xE000E30C	IABR0 - IABR3	RO	Privileged	0x00000000	<a href="#">Table 781</a>
0xE000E400 - 0xE000E46C	IPR0 - IPR27	RW	Privileged	0x00000000	<a href="#">Table 782</a>
0xE000EF00	STIR	WO	Configurable <sup>[1]</sup>	0x00000000	<a href="#">Table 783</a>

[1] Each array element corresponds to a single NVIC register, for example the element ICER[1] corresponds to the ICER1 register.

#### 38.4.2.1 The CMSIS mapping of the Cortex-M3 NVIC registers

To improve software efficiency, the CMSIS simplifies the NVIC register presentation. In the CMSIS:

- the Set-enable, Clear-enable, Set-pending, Clear-pending and Active Bit registers map to arrays of 32-bit integers, so that:
  - the array ISER[0] to ISER[3] corresponds to the registers ISER0 - ISER3
  - the array ICER[0] to ICER[3] corresponds to the registers ICER0 - ICER3
  - the array ISPR[0] to ISPR[3] corresponds to the registers ISPR0 - ISPR3
  - the array ICPR[0] to ICPR[3] corresponds to the registers ICPR0 - ICPR3
  - the array IABR[0] to IABR[3] corresponds to the registers IABR0 - IABR3.
- the 8-bit fields of the Interrupt Priority Registers map to an array of 8-bit integers, so that the array IP[0] to IP[112] corresponds to the registers IPR0 - IPR59, and the array entry IP[n] holds the interrupt priority for interrupt n.

The CMSIS provides thread-safe code that gives atomic access to the Interrupt Priority Registers. For more information see the description of the `NVIC_SetPriority` function in [Section 38.4.2.10.1 “NVIC programming hints”](#). [Table 776](#) shows how the interrupts, or IRQ numbers, map onto the interrupt registers and corresponding CMSIS variables that have one bit per interrupt.

**Table 776. Mapping of interrupts to the interrupt variables**

Interrupts	CMSIS array elements <sup>[1]</sup>				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]
64-95	ISER[2]	ICER[2]	ISPR[2]	ICPR[2]	IABR[2]
96-127	ISER[3]	ICER[3]	ISPR[3]	ICPR[3]	IABR[3]

[1] Each array element corresponds to a single NVIC register, for example the element `ICER[1]` corresponds to the `ICER1` register.

### 38.4.2.2 Interrupt Set-enable Registers

The `ISER0-ISER3` registers enable interrupts, and show which interrupts are enabled. See:

- the register summary in [Table 775](#) for the register attributes
- [Table 776](#) for which interrupts are controlled by each register.

The bit assignments are shown in [Table 777](#).

**Table 777. ISER bit assignments**

Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0 = no effect 1 = enable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

### 38.4.2.3 Interrupt Clear-enable Registers

The `ICER0-ICER3` registers disable interrupts, and show which interrupts are enabled. See:

- the register summary in [Table 775](#) for the register attributes
- [Table 776](#) for which interrupts are controlled by each register.

The bit assignments are shown in [Table 778](#).

**Table 778. ICER bit assignments**

Bits	Name	Function
[31:0]	CLRENA	Interrupt clear-enable bits. Write: 0 = no effect 1 = disable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

#### 38.4.2.4 Interrupt Set-pending Registers

The ISPR0-ISPR3 registers force interrupts into the pending state, and show which interrupts are pending. See:

- the register summary in [Table 775](#) for the register attributes
- [Table 776](#) for which interrupts are controlled by each register.

The bit assignments are shown in [Table 779](#).

**Table 779. ISPR bit assignments**

Bits	Name	Function
[31:0]	SETPEND	Interrupt set-pending bits. Write: 0 = no effect 1 = changes interrupt state to pending. Read: 0 = interrupt is not pending 1 = interrupt is pending.

**Remark:** Writing 1 to the ISPR bit corresponding to:

- an interrupt that is pending has no effect
- a disabled interrupt sets the state of that interrupt to pending.

#### 38.4.2.5 Interrupt Clear-pending Registers

The ICPR0-ICPR3 registers remove the pending state from interrupts, and show which interrupts are pending. See:

- the register summary in [Table 775](#) for the register attributes
- [Table 776](#) for which interrupts are controlled by each register.

The bit assignments are shown in [Table 780](#).

**Table 780. ICPR bit assignments**

Bits	Name	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. Write: 0 = no effect 1 = removes pending state an interrupt. Read: 0 = interrupt is not pending 1 = interrupt is pending.

**Remark:** Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

### 38.4.2.6 Interrupt Active Bit Registers

The IABR0-IABR3 registers indicate which interrupts are active. See:

- the register summary in [Table 775](#) for the register attributes
- [Table 776](#) for which interrupts are controlled by each register.

The bit assignments are shown in [Table 781](#).

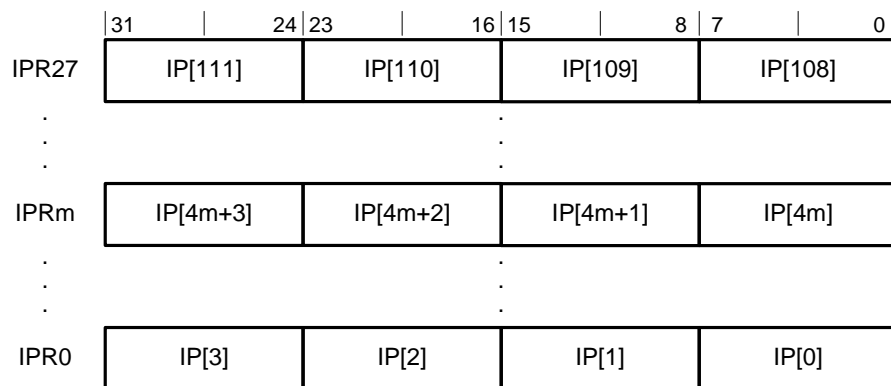
**Table 781. IABR bit assignments**

Bits	Name	Function
[31:0]	ACTIVE	Interrupt active flags: 0 = interrupt not active 1 = interrupt active.

A bit reads as one if the status of the corresponding interrupt is active or active and pending.

### 38.4.2.7 Interrupt Priority Registers

The IPR0-IPR27 registers provide a 5-bit priority field for each interrupt. These registers are byte-accessible. See the register summary in [Table 775](#) for their attributes. Each register holds four priority fields, that map to four elements in the CMSIS interrupt priority array IP[0] to IP[111], as shown:





**Table 782. IPR bit assignments**

Bits	Name	Function
[31:24]	Priority, byte offset 3	Each priority field holds a priority value, 0-31. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:3] of each field, bits[2:0] read as zero and ignore writes.
[23:16]	Priority, byte offset 2	
[15:8]	Priority, byte offset 1	
[7:0]	Priority, byte offset 0	

See [Table 776](#) for more information about the IP[0] to IP[111] interrupt priority array, that provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt **N** as follows:

- the corresponding IPR number, **M**, is given by  $M = N \text{ DIV } 4$
- the byte offset of the required Priority field in this register is  $N \text{ MOD } 4$ , where:
  - byte offset 0 refers to register bits[7:0]
  - byte offset 1 refers to register bits[15:8]
  - byte offset 2 refers to register bits[23:16]
  - byte offset 3 refers to register bits[31:24].

### 38.4.2.8 Software Trigger Interrupt Register

Write to the STIR to generate a **Software Generated Interrupt** (SGI). See the register summary in [Table 775](#) for the STIR attributes.

When the USERSETMPEND bit in the CCR is set to 1, unprivileged software can access the STIR, see [Section 38.4.3.8 “Configuration and Control Register”](#).

**Remark:** Only privileged software can enable unprivileged access to the STIR.

The bit assignments are shown in [Table 783](#).

**Table 783. STIR bit assignments**

Bits	Field	Function
[31:9]	-	Reserved.
[8:0]	INTID	Interrupt ID of the required SGI, in the range 0-111. For example, a value of b000000011 specifies interrupt IRQ3.

### 38.4.2.9 Level-sensitive and pulse interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see [Section 38.4.2.9.1](#). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

#### 38.4.2.9.1 Hardware and software control of interrupts

The Cortex-M3 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see [Table 779](#), or to the STIR to make an SGI pending, see [Table 783](#).

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.  
If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.  
For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.  
For a pulse interrupt, state of the interrupt changes to:
  - inactive, if the state was pending
  - active, if the state was active and pending.

#### 38.4.2.10 NVIC design hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter pending state even it is disabled.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers, NMI and all enabled exception like interrupts. For more information see [Table 789](#).

**38.4.2.10.1 NVIC programming hints**

Software uses the CPSIE I and CPSID I instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

**Table 784. CMSIS functions for NVIC control**

CMSIS interrupt control function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

For more information about these functions see the CMSIS documentation.

### 38.4.3 System control block

The **System control block** (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

**Table 785. Summary of the system control block registers**

Address	Name	Type	Required privilege	Reset value	Description
0xE000E008	ACTLR	RW	Privileged	0x00000000	<a href="#">Table 786</a>
0xE000ED00	CPUID	RO	Privileged	0x412FC230	<a href="#">Table 787</a>
0xE000ED04	ICSR	RW <sup>[1]</sup>	Privileged	0x00000000	<a href="#">Table 788</a>
0xE000ED08	VTOR	RW	Privileged	0x00000000	<a href="#">Table 789</a>
0xE000ED0C	AIRCR	RW <sup>[1]</sup>	Privileged	0xFA050000	<a href="#">Table 790</a>
0xE000ED10	SCR	RW	Privileged	0x00000000	<a href="#">Table 792</a>
0xE000ED14	CCR	RW	Privileged	0x00000200	<a href="#">Table 793</a>
0xE000ED18	SHPR1	RW	Privileged	0x00000000	<a href="#">Table 795</a>
0xE000ED1C	SHPR2	RW	Privileged	0x00000000	<a href="#">Table 796</a>
0xE000ED20	SHPR3	RW	Privileged	0x00000000	<a href="#">Table 797</a>
0xE000ED24	SHCRS	RW	Privileged	0x00000000	<a href="#">Table 798</a>
0xE000ED28	CFSR	RW	Privileged	0x00000000	<a href="#">Section 38.4.3.11</a>
0xE000ED28	MMSR <sup>[2]</sup>	RW	Privileged	0x00	<a href="#">Table 799</a>
0xE000ED29	BFSR <sup>[2]</sup>	RW	Privileged	0x00	<a href="#">Table 800</a>
0xE000ED2A	UFSR <sup>[2]</sup>	RW	Privileged	0x0000	<a href="#">Table 801</a>
0xE000ED2C	HFSR	RW	Privileged	0x00000000	<a href="#">Table 802</a>
0xE000ED34	MMFAR	RW	Privileged	Undefined	<a href="#">Table 803</a>
0xE000ED38	BFAR	RW	Privileged	Undefined	<a href="#">Table 804</a>

[1] See the register description for more information.

[2] A subregister of the CFSR.

#### 38.4.3.1 The CMSIS mapping of the Cortex-M3 SCB registers

To improve software efficiency, the CMSIS simplifies the SCB register presentation. In the CMSIS, the byte array `SHP[0]` to `SHP[12]` corresponds to the registers SHPR1-SHPR3.

#### 38.4.3.2 Auxiliary Control Register

The ACTLR provides disable bits for the following processor functions:

- IT folding
- write buffer use for accesses to the default memory map
- interruption of multi-cycle instructions.

See the register summary in [Table 785](#) for the ACTLR attributes. The bit assignments are shown in [Table 786](#).

Table 786. ACTLR bit assignments

Bits	Name	Function
[31:3]	-	Reserved
[2]	DISFOLD	When set to 1, disables IT folding. see <a href="#">Section 38.4.3.2.1</a> for more information.
[1]	DISDEFWBUF	When set to 1, disables write buffer use during default memory map accesses. This causes all bus faults to be precise bus faults but decreases performance because any store to memory must complete before the processor can execute the next instruction. <b>Remark:</b> This bit only affects write buffers implemented in the Cortex-M3 processor.
[0]	DISMCYCINT	When set to 1, disables interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.

### 38.4.3.2.1 About IT folding

In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and improves performance. However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable IT folding.

### 38.4.3.3 CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in [Table 785](#) for its attributes. The bit assignments are shown in [Table 787](#).

Table 787. CPUID register bit assignments

Bits	Name	Function
[31:24]	Implementer	Implementer code: 0x41 = ARM
[23:20]	Variant	Variant number, the r value in the rnpn product revision identifier: 0x2 = r2p0
[19:16]	Constant	Reads as 0xF
[15:4]	PartNo	Part number of the processor: 0xC23 = Cortex-M3
[3:0]	Revision	Revision number, the p value in the rnpn product revision identifier: 0x0 = r2p0

### 38.4.3.4 Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the **Non-Maskable Interrupt** (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed

- whether there are preempted active exceptions
- the exception number of the highest priority pending exception
- whether any interrupts are pending.

See the register summary in [Table 785](#), and the Type descriptions in [Table 788](#), for the ICSR attributes. The bit assignments are shown in [Table 788](#).

**Table 788. ICSR bit assignments**

Bits	Name	Type	Function
[31]	NMIPENDSET	RW	<p>NMI set-pending bit.</p> <p>Write:</p> <p>0 = no effect</p> <p>1 = changes NMI exception state to pending.</p> <p>Read:</p> <p>0 = NMI exception is not pending</p> <p>1 = NMI exception is pending.</p> <p>Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.</p>
[30:29]	-	-	Reserved.
[28]	PENDSVSET	RW	<p>PendSV set-pending bit.</p> <p>Write:</p> <p>0 = no effect</p> <p>1 = changes PendSV exception state to pending.</p> <p>Read:</p> <p>0 = PendSV exception is not pending</p> <p>1 = PendSV exception is pending.</p> <p>Writing 1 to this bit is the only way to set the PendSV exception state to pending.</p>
[27]	PENDSVCLR	WO	<p>PendSV clear-pending bit.</p> <p>Write:</p> <p>0 = no effect</p> <p>1 = removes the pending state from the PendSV exception.</p>
[26]	PENDSTSET	RW	<p>SysTick exception set-pending bit.</p> <p>Write:</p> <p>0 = no effect</p> <p>1 = changes SysTick exception state to pending.</p> <p>Read:</p> <p>0 = SysTick exception is not pending</p> <p>1 = SysTick exception is pending.</p>

Table 788. ICSR bit assignments

Bits	Name	Type	Function
[25]	PENDSTCLR	WO	SysTick exception clear-pending bit. Write: 0 = no effect 1 = removes the pending state from the SysTick exception. This bit is WO. On a register read its value is Unknown.
[24]	-	-	Reserved.
[23]	Reserved for Debug use	RO	This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
[22]	ISR_PENDING	RO	Interrupt pending flag, excluding NMI and Faults: 0 = interrupt not pending 1 = interrupt pending.
[21:18]	-	-	Reserved.
[17:12]	VECT_PENDING	RO	Indicates the exception number of the highest priority pending enabled exception: 0 = no pending exceptions Nonzero = the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.
[11]	RETTOBASE	RO	Indicates whether there are preempted active exceptions: 0 = there are preempted active exceptions to execute 1 = there are no active exceptions, or the currently-executing exception is the only active exception.
[10:9]	-	-	Reserved.
[8:0]	VECT_ACTIVE <sup>[1]</sup>	RO	Contains the active exception number: 0 = Thread mode Nonzero = The exception number <sup>[1]</sup> of the currently active exception. <b>Remark:</b> Subtract 16 from this value to obtain the IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see <a href="#">Table 760 "IPSR bit assignments"</a> .

[1] This is the same value as IPSR bits[8:0], see [Table 760 "IPSR bit assignments"](#).

When you write to the ICSR, the effect is unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

### 38.4.3.5 Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000. See the register summary in [Table 785](#) for its attributes.

The bit assignments are shown in [Table 789](#).

**Table 789. VTOR bit assignments**

Bits	Name	Function
[31:30]	-	Reserved.
[29:8]	TBLOFF	<p>Vector table base offset field. It contains bits[29:8] of the offset of the table base from the bottom of the memory map.</p> <p><b>Remark:</b> Bit[29] determines whether the vector table is in the code or SRAM memory region:</p> <p>Bit[29] is sometimes called the TBLBASE bit.</p> <ul style="list-style-type: none"> <li>• 0 = code</li> <li>• 1 = SRAM.</li> </ul>
[7:0]	-	Reserved.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. The recommended alignment is 256 words, allowing for 128 interrupts.

**Remark:** Table alignment requirements mean that bits[7:0] of the table offset are always zero.

### 38.4.3.6 Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. See the register summary in [Table 785](#) and [Table 790](#) for its attributes.

To write to this register, you must write 0x5VA to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are shown in [Table 790](#).

**Table 790. AIRCR bit assignments**

Bits	Name	Type	Function
[31:16]	Write: VECTKEYSTAT Read: VECTKEY	RW	<p>Register key: Reads as 0x05FA</p> <p>On writes, write 0x5FA to VECTKEY, otherwise the write is ignored.</p>
[15]	ENDIANESS	RO	Data endianness bit: 0 = Little-endian.
[14:11]	-	-	Reserved
[10:8]	PRIGROUP	R/W	Interrupt priority grouping field. This field determines the split of group priority from subpriority, see <a href="#">Section 38.4.3.6.1</a> .
[7:3]	-	-	Reserved.



Table 790. AIRCR bit assignments

Bits	Name	Type	Function
[2]	SYSRESETREQ	WO	System reset request: 0 = no system reset request 1 = asserts a signal to the outer system that requests a reset. This is intended to force a large system reset of all major components except for debug. <b>Note: SYSRESETREQ is supported in LPC178x/177x devices.</b> This bit reads as 0.
[1]	VECTCLRACTIVE	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
[0]	VECTRESET	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

38.4.3.6.1 Binary point

The PRIGROUP field indicates the position of the binary point that splits the PRI\_n fields in the Interrupt Priority Registers into separate **group priority** and **subpriority** fields. [Table 791](#) shows how the PRIGROUP value controls this split.

Table 791. Priority grouping

PRIGROUP	Interrupt priority level value, PRI_N[7:0]			Number of	
	Binary point <sup>[1]</sup>	Group priority bits	Subpriority bits	Group priorities	Subpriorities
b010	bxxxxx.000	[7:3]	none	32	1
b011	bxxxx.y000	[7:4]	[3]	16	2
b100	bxxx.yy000	[7:5]	[4:3]	8	4
b101	bxx.yyy000	[7:6]	[5:3]	4	8
b110	bx.yyyy000	[7]	[6:3]	2	16
b111	b.yyyyy000	None	[7:3]	1	32

[1] PRI\_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit. Bits [2:0] are not used in LPC178x/177x devices.

**Remark:** Determining preemption of an exception uses only the group priority field, see [Section 38.3.3.6 “Interrupt priority grouping”](#).

38.4.3.7 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in [Table 785](#) for its attributes. The bit assignments are shown in [Table 792](#).

**Table 792. SCR bit assignments**

Bits	Name	Function
[31:5]	-	Reserved.
[4]	SEVONPEND	Send Event on Pending bit: 0 = only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded 1 = enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.
[3]	-	Reserved.
[2]	SLEEPDEEP	Controls whether the processor uses sleep or deep sleep as its low power mode: 0 = sleep 1 = deep sleep.
[1]	SLEEPONEXIT	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0 = do not sleep when returning to Thread mode. 1 = enter sleep, or deep sleep, on return from an ISR. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
[0]	-	Reserved.

### 38.4.3.8 Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore bus faults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see [Table 783](#).

See the register summary in [Table 785](#) for the CCR attributes.

The bit assignments are shown in [Table 793](#).

**Table 793. CCR bit assignments**

Bits	Name	Function
[31:10]	-	Reserved.
[9]	STKALIGN	<p>Indicates stack alignment on exception entry:</p> <p>0 = 4-byte aligned 1 = 8-byte aligned.</p> <p>On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment.</p>
[8]	BFHFNMIEN	<p>Enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. This applies to the hard fault, NMI, and FAULTMASK escalated handlers:</p> <p>0 = data bus faults caused by load and store instructions cause a lock-up 1 = handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.</p> <p>Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.</p>
[7:5]	-	Reserved.
[4]	DIV_0_TRP	<p>Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0:</p> <p>0 = do not trap divide by 0 1 = trap divide by 0.</p> <p>When this bit is set to 0, a divide by zero returns a quotient of 0.</p>
[3]	UNALIGN_TRP	<p>Enables unaligned access traps:</p> <p>0 = do not trap unaligned halfword and word accesses 1 = trap unaligned halfword and word accesses.</p> <p>If this bit is set to 1, an unaligned access generates a usage fault. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.</p>
[2]	-	Reserved.
[1]	USERSETMPEND	<p>Enables unprivileged software access to the STIR, see <a href="#">Table 783</a>:</p> <p>0 = disable 1 = enable.</p>
[0]	NONEBASE_THRDENA	<p>Indicates how the processor enters Thread mode:</p> <p>0 = processor can enter Thread mode only when no exception is active. 1 = processor can enter Thread mode from any level under the control of an EXC_RETURN value, see <a href="#">Section 38.3.3.7.2 "Exception return"</a>.</p>

### 38.4.3.9 System Handler Priority Registers

The SHPR1-SHPR3 registers set the priority level, 0 to 31 of the exception handlers that have configurable priority.

SHPR1-SHPR3 are byte accessible. See the register summary in [Table 785](#) for their attributes.

The system fault handlers and the priority field and register for each handler are:

**Table 794. System fault handler priority fields**

Handler	Field	Register description
Memory management fault	PRI_4	<a href="#">Table 795</a>
Bus fault	PRI_5	
Usage fault	PRI_6	
SVCcall	PRI_11	<a href="#">Table 796</a>
PendSV	PRI_14	<a href="#">Table 797</a>
SysTick	PRI_15	

Each PRI\_N field is 8 bits wide, but the processor implements only bits[7:3] of each field, and bits[2:0] read as zero and ignore writes.

### 38.4.3.9.1 System Handler Priority Register 1

The bit assignments are shown in [Table 795](#).

**Table 795. SHPR1 register bit assignments**

Bits	Name	Function
[31:24]	PRI_7	Reserved
[23:16]	PRI_6	Priority of system handler 6, usage fault
[15:8]	PRI_5	Priority of system handler 5, bus fault
[7:0]	PRI_4	Priority of system handler 4, memory management fault

### 38.4.3.9.2 System Handler Priority Register 2

The bit assignments are shown in [Table 796](#).

**Table 796. SHPR2 register bit assignments**

Bits	Name	Function
[31:24]	PRI_11	Priority of system handler 11, SVCcall
[23:0]	-	Reserved

### 38.4.3.9.3 System Handler Priority Register 3

The bit assignments are shown in [Table 797](#).

**Table 797. SHPR3 register bit assignments**

Bits	Name	Function
[31:24]	PRI_15	Priority of system handler 15, SysTick exception
[23:16]	PRI_14	Priority of system handler 14, PendSV
[15:0]	-	Reserved

### 38.4.3.10 System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

- the pending status of the bus fault, memory management fault, and SVC exceptions
- the active status of the system handlers.

See the register summary in [Table 785](#) for the SHCSR attributes. The bit assignments are shown in [Table 798](#).

**Table 798. SHCSR bit assignments**

Bits	Name	Function
[31:19]	-	Reserved
[18]	USGFAULTENA	Usage fault enable bit, set to 1 to enable <sup>[1]</sup>
[17]	BUSFAULTENA	Bus fault enable bit, set to 1 to enable <sup>[1]</sup>
[16]	MEMFAULTENA	Memory management fault enable bit, set to 1 to enable <sup>[1]</sup>
[15]	SVCALLPENDEd	SVC call pending bit, reads as 1 if exception is pending <sup>[2]</sup>
[14]	BUSFAULTPENDEd	Bus fault exception pending bit, reads as 1 if exception is pending <sup>[2]</sup>
[13]	MEMFAULTPENDEd	Memory management fault exception pending bit, reads as 1 if exception is pending <sup>[2]</sup>
[12]	USGFAULTPENDEd	Usage fault exception pending bit, reads as 1 if exception is pending <sup>[2]</sup>
[11]	SYSTICKACT	SysTick exception active bit, reads as 1 if exception is active <sup>[3]</sup>
[10]	PENDSVACT	PendSV exception active bit, reads as 1 if exception is active
[9]	-	Reserved
[8]	MONITORACT	Debug monitor active bit, reads as 1 if Debug monitor is active
[7]	SVCALLACT	SVC call active bit, reads as 1 if SVC call is active
[6:4]	-	Reserved
[3]	USGFAULTACT	Usage fault exception active bit, reads as 1 if exception is active
[2]	-	Reserved
[1]	BUSFAULTACT	Bus fault exception active bit, reads as 1 if exception is active
[0]	MEMFAULTACT	Memory management fault exception active bit, reads as 1 if exception is active

[1] Enable bits, set to 1 to enable the exception, or set to 0 to disable the exception.

[2] Pending bits, read as 1 if the exception is pending, or as 0 if it is not pending. You can write to these bits to change the pending status of the exceptions.

[3] Active bits, read as 1 if the exception is active, or as 0 if it is not active. You can write to these bits to change the active status of the exceptions, but see the Caution in this section.

If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

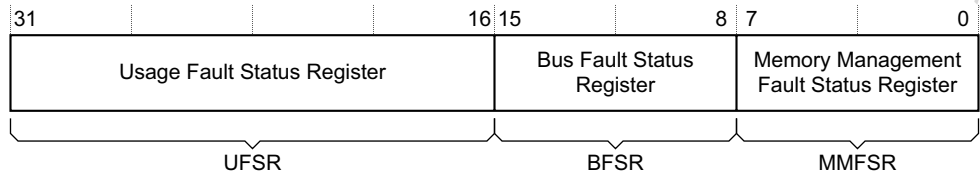
You can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

**Caution**

- Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.
- After you have enabled the system handlers, if you have to change the value of a bit in this register you must use a read-modify-write procedure to ensure that you change only the required bit.

### 38.4.3.11 Configurable Fault Status Register

The CFSR indicates the cause of a memory management fault, bus fault, or usage fault. See the register summary in [Table 785](#) for its attributes. The bit assignments are:



The following subsections describe the subregisters that make up the CFSR:

- [Table 799 “MMFSR bit assignments”](#)
- [Table 804 “BFAR bit assignments”](#)
- [Table 801 “UFSR bit assignments”](#).

The CFSR is byte accessible. You can access the CFSR or its subregisters as follows:

- access the complete CFSR with a word access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR and BFSR with a halfword access to 0xE000ED28
- access the BFSR with a byte access to 0xE000ED29
- access the UFSR with a halfword access to 0xE000ED2A.

#### 38.4.3.11.1 Memory Management Fault Status Register

The flags in the MMFSR indicate the cause of memory access faults. The bit assignments are shown in [Table 799](#).

**Table 799. MMFSR bit assignments**

Bits	Name	Function
[7]	MMARVALID	<p><b>Memory Management Fault Address Register (MMAR) valid flag:</b>                      0 = value in MMAR is not a valid fault address                      1 = MMAR holds a valid fault address.</p> <p>If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems on return to a stacked active memory management fault handler whose MMAR value has been overwritten.</p>
[6:5]	-	Reserved.
[4]	MSTKERR	<p>Memory manager fault on stacking for exception entry:                      0 = no stacking fault                      1 = stacking for an exception entry has caused one or more access violations.</p> <p>When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to the MMAR.</p>

Table 799. MMFSR bit assignments

Bits	Name	Function
[3]	MUNSTKERR	<p>Memory manager fault on unstacking for a return from exception:</p> <p>0 = no unstacking fault</p> <p>1 = unstack for an exception return has caused one or more access violations.</p> <p>This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the MMAR.</p>
[2]	-	Reserved
[1]	DACCVIOL	<p>Data access violation flag:</p> <p>0 = no data access violation fault</p> <p>1 = the processor attempted a load or store at a location that does not permit the operation.</p> <p>When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the MMAR with the address of the attempted access.</p>
[0]	IACCVIOL	<p>Instruction access violation flag:</p> <p>0 = no instruction access violation fault</p> <p>1 = the processor attempted an instruction fetch from a location that does not permit execution.</p> <p>This fault occurs on any access to an XN region, even when the MPU is disabled.</p> <p>When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the MMAR.</p>

### 38.4.3.11.2 Bus Fault Status Register

The flags in the BFSR indicate the cause of a bus access fault. The bit assignments are shown in [Table 800](#).

Table 800. BFSR bit assignments

Bits	Name	Function
[7]	BFARVALID	<p><b>Bus Fault Address Register (BFAR) valid flag:</b></p> <p>0 = value in BFAR is not a valid fault address</p> <p>1 = BFAR holds a valid fault address.</p> <p>The processor sets this bit to 1 after a bus fault where the address is known. Other faults can set this bit to 0, such as a memory management fault occurring later.</p> <p>If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active bus fault handler whose BFAR value has been overwritten.</p>
[6:5]	-	Reserved.

**Table 800. BFSR bit assignments**

Bits	Name	Function
[4]	STKERR	<p>Bus fault on stacking for exception entry:</p> <p>0 = no stacking fault</p> <p>1 = stacking for an exception entry has caused one or more bus faults.</p> <p>When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the BFAR.</p>
[3]	UNSTKERR	<p>Bus fault on unstacking for a return from exception:</p> <p>0 = no unstacking fault</p> <p>1 = unstack for an exception return has caused one or more bus faults.</p> <p>This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.</p>
[2]	IMPRECISERR	<p>Imprecise data bus error:</p> <p>0 = no imprecise data bus error</p> <p>1 = a data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.</p> <p>When the processor sets this bit to 1, it does not write a fault address to the BFAR.</p> <p>This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects both IMPRECISERR set to 1 and one of the precise fault status bits set to 1.</p>
[1]	PRECISERR	<p>Precise data bus error:</p> <p>0 = no precise data bus error</p> <p>1 = a data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.</p> <p>When the processor sets this bit is 1, it writes the faulting address to the BFAR.</p>
[0]	IBUSERR	<p>Instruction bus error:</p> <p>0 = no instruction bus error</p> <p>1 = instruction bus error.</p> <p>The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.</p> <p>When the processor sets this bit is 1, it does not write a fault address to the BFAR.</p>

**38.4.3.11.3 Usage Fault Status Register**

The UFSR indicates the cause of a usage fault. The bit assignments are shown in [Table 801](#).



**Table 801. UFSR bit assignments**

Bits	Name	Function
[15:10]	-	Reserved.
[9]	DIVBYZERO	<p>Divide by zero usage fault:</p> <p>0 = no divide by zero fault, or divide by zero trapping not enabled</p> <p>1 = the processor has executed an <code>SDIV</code> or <code>UDIV</code> instruction with a divisor of 0.</p> <p>When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero.</p> <p>Enable trapping of divide by zero by setting the <code>DIV_0_TRP</code> bit in the CCR to 1, see <a href="#">Table 793</a>.</p>
[8]	UNALIGNED	<p>Unaligned access usage fault:</p> <p>0 = no unaligned access fault, or unaligned access trapping not enabled</p> <p>1 = the processor has made an unaligned memory access.</p> <p>Enable trapping of unaligned accesses by setting the <code>UNALIGN_TRP</code> bit in the CCR to 1, see <a href="#">Table 793</a>.</p> <p>Unaligned <code>LDM</code>, <code>STM</code>, <code>LDRD</code>, and <code>STRD</code> instructions always fault irrespective of the setting of <code>UNALIGN_TRP</code>.</p>
[7:4]	-	Reserved.
[3]	NOCP	<p>No coprocessor usage fault. The processor does not support coprocessor instructions:</p> <p>0 = no usage fault caused by attempting to access a coprocessor</p> <p>1 = the processor has attempted to access a coprocessor.</p>
[2]	INVPC	<p>Invalid PC load usage fault, caused by an invalid PC load by <code>EXC_RETURN</code>:</p> <p>0 = no invalid PC load usage fault</p> <p>1 = the processor has attempted an illegal load of <code>EXC_RETURN</code> to the PC, as a result of an invalid context, or an invalid <code>EXC_RETURN</code> value.</p> <p>When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.</p>
[1]	INVSTATE	<p>Invalid state usage fault:</p> <p>0 = no invalid state usage fault</p> <p>1 = the processor has attempted to execute an instruction that makes illegal use of the EPSR.</p> <p>When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR.</p> <p>This bit is not set to 1 if an undefined instruction uses the EPSR.</p>
[0]	UNDEFINSTR	<p>Undefined instruction usage fault:</p> <p>0 = no undefined instruction usage fault</p> <p>1 = the processor has attempted to execute an undefined instruction.</p> <p>When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction.</p> <p>An undefined instruction is an instruction that the processor cannot decode.</p>

**Remark:** The UFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

### 38.4.3.12 Hard Fault Status Register

The HFSR gives information about events that activate the hard fault handler. See the register summary in [Table 785](#) for its attributes.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are shown in [Table 802](#).

**Table 802. HFSR bit assignments**

Bits	Name	Function
[31]	DEBUGEVT	Reserved for Debug use. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
[30]	FORCED	Indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled: 0 = no forced hard fault 1 = forced hard fault. When this bit is set to 1, the hard fault handler must read the other fault status registers to find the cause of the fault.
[29:2]	-	Reserved.
[1]	VECTTBL	Indicates a bus fault on a vector table read during exception processing: 0 = no bus fault on vector table read 1 = bus fault on vector table read. This error is always handled by the hard fault handler. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.
[0]	-	Reserved.

**Remark:** The HFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

### 38.4.3.13 Memory Management Fault Address Register

The MMFAR contains the address of the location that generated a memory management fault. See the register summary in [Table 785](#) for its attributes. The bit assignments are:

**Table 803. MMFAR bit assignments**

Bits	Name	Function
[31:0]	ADDRESS	When the MMARVALID bit of the MMFSR is set to 1, this field holds the address of the location that generated the memory management fault

When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See [Table 799](#).

### 38.4.3.14 Bus Fault Address Register

The BFAR contains the address of the location that generated a bus fault. See the register summary in [Table 785](#) for its attributes. The bit assignments are:

**Table 804. BFAR bit assignments**

Bits	Name	Function
[31:0]	ADDRESS	When the BFARVALID bit of the BFSR is set to 1, this field holds the address of the location that generated the bus fault

When an unaligned access faults the address in the BFAR is the one requested by the instruction, even if it is not the address of the fault.

Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See [Table 800](#).

**38.4.3.15 System control block design hints and tips**

Ensure software uses aligned accesses of the correct size to access the system control block registers:

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses
- for the CFSR and SHPR1-SHPR3 it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or BFAR value.
2. Read the MMARVALID bit in the MMFSR, or the BFARVALID bit in the BFSR. The MMFAR or BFAR address is valid only if this bit is 1.

Software must follow this sequence because another higher priority exception might change the MMFAR or BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the MMFAR or BFAR value.

### 38.4.4 System timer, SysTick

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the LOAD register on the next clock edge, then counts down on subsequent clocks.

**Note:** refer to the separate chapter in the LPC178x/177x User Manual [Section 25.1](#) for device specific information on the System Timer.

**Remark:** when the processor is halted for debugging the counter does not decrement.

The system timer registers are:

**Table 805. System timer registers summary**

Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	CTRL	RW	Privileged	0x00000004	<a href="#">Table 806</a>
0xE000E014	LOAD	RW	Privileged	0x00000000	<a href="#">Table 807</a>
0xE000E018	VAL	RW	Privileged	0x00000000	<a href="#">Table 808</a>
0xE000E01C	CALIB	RO	Privileged	0x000F423F <sup>[1]</sup>	<a href="#">Table 809</a>

[1] SysTick calibration value. This value is specific to LPC178x/177x devices.

#### 38.4.4.1 SysTick Control and Status Register

The SysTick CTRL register enables the SysTick features. See the register summary in [Table 805](#) for its attributes. The bit assignments are shown in [Table 806](#).

**Table 806. SysTick CTRL register bit assignments**

Bits	Name	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read.
[15:3]	-	Reserved.
[2]	CLKSOURCE	Indicates the clock source: 0 = external clock 1 = processor clock.
[1]	TICKINT	Enables SysTick exception request: 0 = counting down to zero does not assert the SysTick exception request 1 = counting down to zero to asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0 = counter disabled 1 = counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the LOAD register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

### 38.4.4.2 SysTick Reload Value Register

The LOAD register specifies the start value to load into the VAL register. See the register summary in [Table 805](#) for its attributes. The bit assignments are shown in [Table 807](#).

**Table 807. LOAD register bit assignments**

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	RELOAD	Value to load into the VAL register when the counter is enabled and when it reaches 0, see <a href="#">Section 38.4.4.2.1</a> .

#### 38.4.4.2.1 Calculating the RELOAD value

The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use:

- To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.
- To deliver a single SysTick interrupt after a delay of N processor clock cycles, use a RELOAD of value N. For example, if a SysTick interrupt is required after 400 clock pulses, set RELOAD to 400.

### 38.4.4.3 SysTick Current Value Register

The VAL register contains the current value of the SysTick counter. See the register summary in [Table 805](#) for its attributes. The bit assignments are shown in [Table 808](#).

**Table 808. VAL register bit assignments**

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	CURRENT	Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SysTick CTRL.COUNTFLAG bit to 0.

### 38.4.4.4 SysTick Calibration Value Register

The CALIB register indicates the SysTick calibration properties. See the register summary in [Table 805](#) for its attributes. The bit assignments are shown in [Table 809](#).

**Table 809. CALIB register bit assignments**

Bits	Name	Function
[31]	NOREF	Indicates whether a separate reference clock is available. This value is factory preset as described in <a href="#">Section 25.1</a> .
[30]	SKEW	Indicates whether the value of TENMS is precise. This can affect the suitability of SysTick as a software real time clock. This value is factory preset as described in <a href="#">Section 25.1</a> .
[29:24]	-	Reserved.
[23:0]	TENMS	This value is factory preset as described in <a href="#">Section 25.1</a> .

If a different frequency is used than that intended by the factory preset value, calculate the calibration value required from the frequency of the processor clock or external clock.

#### 38.4.4.5 SysTick design hints and tips

The SysTick counter runs on the processor clock. If this clock signal is stopped for low power mode while the SysTick counter is running from it, the SysTick counter will stop.

Ensure software uses aligned word accesses to access the SysTick registers.

### 38.4.5 Memory protection unit

This section describes the **Memory protection unit** (MPU).

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- independent attribute settings for each region
- overlapping regions
- export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M3 MPU defines:

- eight separate memory regions, 0-7
- a background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M3 MPU memory map is unified. This means instruction accesses and data accesses have same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault. This causes a fault exception, and might cause termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types, see [Section 38.3.2.1 “Memory regions, types and attributes”](#).

[Table 810](#) shows the possible MPU region attributes. These include Shareability and cache behavior attributes that are not relevant to most microcontroller implementations. See [Table 821](#) for guidelines for programming such an implementation.

**Table 810. Memory attributes summary**

Memory type	Shareability	Other attributes	Description
Strongly- ordered	-	-	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	-	Memory-mapped peripherals that several processors share.

**Table 810. Memory attributes summary**

Memory type	Shareability	Other attributes	Description
	Non-shared	-	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that only a single processor uses.

Use the MPU registers to define the MPU regions and their attributes. The MPU registers are:

**Table 811. MPU registers summary**

Address	Name	Type	Required privilege	Reset value	Description
0xE000ED90	TYPE	RO	Privileged	0x00000800	<a href="#">Table 812</a>
0xE000ED94	CTRL	RW	Privileged	0x00000000	<a href="#">Table 813</a>
0xE000ED98	RNR	RW	Privileged	0x00000000	<a href="#">Table 814</a>
0xE000ED9C	RBAR	RW	Privileged	0x00000000	<a href="#">Table 815</a>
0xE000EDA0	RASR	RW	Privileged	0x00000000	<a href="#">Table 816</a>
0xE000EDA4	RBAR_A1	RW	Privileged	0x00000000	Alias of RBAR, see <a href="#">Table 815</a>
0xE000EDA8	RASR_A1	RW	Privileged	0x00000000	Alias of RASR, see <a href="#">Table 816</a>
0xE000EDAC	RBAR_A2	RW	Privileged	0x00000000	Alias of RBAR, see <a href="#">Table 815</a>
0xE000EDB0	RASR_A2	RW	Privileged	0x00000000	Alias of RASR, see <a href="#">Table 816</a>
0xE000EDB4	RBAR_A3	RW	Privileged	0x00000000	Alias of RBAR, see <a href="#">Table 815</a>
0xE000EDB8	RASR_A3	RW	Privileged	0x00000000	Alias of RASR, see <a href="#">Table 816</a>

### 38.4.5.1 MPU Type Register

The TYPE register indicates whether the MPU is present, and if so, how many regions it supports. See the register summary in [Table 811](#) for its attributes. The bit assignments are shown in [Table 812](#).

**Table 812. TYPE register bit assignments**

Bits	Name	Function
[31:24]	-	Reserved.
[23:16]	IREGION	Indicates the number of supported MPU instruction regions.  Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.



**Table 812. TYPE register bit assignments**

Bits	Name	Function
[15:8]	DREGION	Indicates the number of supported MPU data regions: 0x08 = Eight MPU regions.
[7:0]	-	Reserved.
[0]	SEPARATE	Indicates support for unified or separate instruction and data memory maps: 0 = unified.

### 38.4.5.2 MPU Control Register

The MPU CTRL register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the hard fault, **Non-maskable Interrupt (NMI)**, and FAULTMASK escalated handlers.

See the register summary in [Table 811](#) for the MPU CTRL attributes. The bit assignments are shown in [Table 813](#).

**Table 813. MPU CTRL register bit assignments**

Bits	Name	Function
[31:3]	-	Reserved.
[2]	PRIVDEFENA	Enables privileged software access to the default memory map: 0 = If the MPU is enabled, disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault. 1 = If the MPU is enabled, enables use of the default memory map as a background region for privileged software accesses. When enabled, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map. If the MPU is disabled, the processor ignores this bit.
[1]	HFNMIENA	Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers. When the MPU is enabled: 0 = MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit 1 = the MPU is enabled during hard fault, NMI, and FAULTMASK handlers. When the MPU is disabled, if this bit is set to 1 the behavior is Unpredictable.
[0]	ENABLE	Enables the MPU: 0 = MPU disabled 1 = MPU enabled.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the **default memory map** is as described in [Section 38.3.2 “Memory model”](#). Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented, see [Table 766 “Memory access behavior”](#). The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

### 38.4.5.3 MPU Region Number Register

The RNR selects which memory region is referenced by the RBAR and RASR registers. See the register summary in [Table 811](#) for its attributes. The bit assignments are shown in [Table 814](#).

**Table 814. RNR bit assignments**

Bits	Name	Function
[31:8]	-	Reserved.
[7:0]	REGION	Indicates the MPU region referenced by the RBAR and RASR registers. The MPU supports 8 memory regions, so the permitted values of this field are 0-7.

Normally, you write the required region number to this register before accessing the RBAR or RASR. However you can change the region number by writing to the RBAR with the VALID bit set to 1, see [Table 815](#). This write updates the value of the REGION field.

### 38.4.5.4 MPU Region Base Address Register

The RBAR defines the base address of the MPU region selected by the RNR, and can update the value of the RNR. See the register summary in [Table 811](#) for its attributes.

Write RBAR with the VALID bit set to 1 to change the current region number and update the RNR. The bit assignments are shown in [Table 815](#).

**Table 815. RBAR bit assignments**

[31:N]	ADDR	Region base address field. The value of N depends on the region size. For more information see <a href="#">Section 38.4.5.4.1</a> .
[(N-1):5]	-	Reserved.
[4]	VALID	MPU Region Number valid bit: Write: 0 = RNR not changed, and the processor: <ul style="list-style-type: none"> <li>updates the base address for the region specified in the RNR</li> <li>ignores the value of the REGION field</li> </ul> 1 = the processor: <ul style="list-style-type: none"> <li>updates the value of the RNR to the value of the REGION field</li> <li>updates the base address for the region specified in the REGION field.</li> </ul> Always reads as zero.
[3:0]	REGION	MPU region field: For the behavior on writes, see the description of the VALID field. On reads, returns the current region number, as specified by the RNR.

**38.4.5.4.1 The ADDR field**

The ADDR field is bits[31:N] of the RBAR. The region size, as specified by the SIZE field in the RASR, defines the value of N:

$$N = \text{Log}_2(\text{Region size in bytes}),$$

If the region size is configured to 4GB, in the RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64KB region must be aligned on a multiple of 64KB, for example, at 0x00010000 or 0x00020000.

**38.4.5.5 MPU Region Attribute and Size Register**

The RASR defines the region size and memory attributes of the MPU region specified by the RNR, and enables that region and any subregions. See the register summary in [Table 811](#) for its attributes.

RASR is accessible using word or halfword accesses:

The bit assignments are shown in [Table 816](#).

- the most significant halfword holds the region attributes
- the least significant halfword holds the region size and the region and subregion enable bits.

**Table 816. RASR bit assignments**

Bits	Name	Function
[31:29]	-	Reserved.
[28]	XN	Instruction access disable bit: 0 = instruction fetches enabled 1 = instruction fetches disabled.
[27]	-	Reserved.
[26:24]	AP	Access permission field, see <a href="#">Table 820</a> .
[23:22]	-	Reserved.
[21:19, 17, 16]	TEX, C, B	Memory access attributes, see <a href="#">Table 818</a> .
[18]	S	Shareable bit, see <a href="#">Table 818</a> .
[15:8]	SRD	Subregion disable bits. For each bit in this field: 0 = corresponding sub-region is enabled 1 = corresponding sub-region is disabled See <a href="#">Section 38.4.5.8.3</a> for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.
[7:6]	-	Reserved.
[5:1]	SIZE	Specifies the size of the MPU protection region. The minimum permitted value is 3 (b00010), see <a href="#">Section 38.4.5.5.1</a> for more information.
[0]	ENABLE	Region enable bit.

For information about access permission, see [Section 38.4.5.6](#).

**38.4.5.5.1 SIZE field values**

The SIZE field defines the size of the MPU memory region specified by the RNR. as follows:

$$(\text{Region size in bytes}) = 2^{(\text{SIZE}+1)}$$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. [Table 817](#) gives example SIZE values, with the corresponding region size and value of N in the RBAR.

**Table 817. Example SIZE field values**

SIZE value	Region size	Value of N <sup>[1]</sup>	Note
b00100 (4)	32B	5	Minimum permitted size
b01001 (9)	1KB	10	-
b10011 (19)	1MB	20	-
b11101 (29)	1GB	30	-
b11111 (31)	4GB	b01100	Maximum possible size

[1] In the RBAR, see [Table 815](#).

38.4.5.6 MPU access permission attributes

This section describes the MPU access permission attributes. The access permission bits, TEX, C, B, S, AP, and XN, of the RASR, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

[Table 818](#) shows the encodings for the TEX, C, B, and S access permission bits.

**Table 818. TEX, C, B, and S encoding**

TEX	C	B	S	Memory type	Shareability	Other attributes
b000	0	0	x <sup>[1]</sup>	Strongly-ordered	Shareable	-
		1	x <sup>[1]</sup>	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1	Normal	Shareable	Outer and inner write-back. No write allocate.
b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
	1	0	x <sup>[1]</sup>	Reserved encoding		-
			x <sup>[1]</sup>	Implementation defined attributes.		-
b010	0	0	x <sup>[1]</sup>	Device	Not shareable	Nonshared Device.
			1	x <sup>[1]</sup>	Reserved encoding	
	1	x <sup>[1]</sup>	x <sup>[1]</sup>	Reserved encoding		-
b1BB	A	A	0	Normal	Not shareable	Cached memory <sup>[2]</sup> , BB = outer policy, AA = inner policy.
			1		Shareable	

[1] The MPU ignores the value of this bit.

[2] See [Table 819](#) for the encoding of the AA and BB bits.

[Table 819](#) shows the cache policy for memory attribute encodings with a TEX value is in the range 4-7.

**Table 819. Cache policy for memory attribute encoding**

Encoding, AA or BB	Corresponding cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

[Table 820](#) shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 820. AP encoding

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from privileged software only
010	RW	RO	Writes by unprivileged software generate a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Reads by privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

### 38.4.5.7 MPU mismatch

When an access violates the MPU permissions, the processor generates a memory management fault, see [Section 38.3.1.4 “Exceptions and interrupts”](#). The MMFSR indicates the cause of the fault. See [Table 799](#) for more information.

### 38.4.5.8 Updating an MPU region

To update the attributes for an MPU region, update the RNR, RBAR and RASR registers. You can program each register separately, or use a multiple-word write to program all of these registers. You can use the RBAR and RASR aliases to program up to four regions simultaneously using an *STM* instruction.

#### 38.4.5.8.1 Updating an MPU region using separate words

Simple code to configure one region:

```

; R1 = region number

; R2 = size/enable

; R3 = attributes

; R4 = address

LDR R0,MPU_RNR      ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R4, [R0, #0x4]  ; Region Base Address
STRH R2, [R0, #0x8] ; Region Size and Enable
STRH R3, [R0, #0xA] ; Region Attribute

```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```

; R1 = region number

; R2 = size/enable

```

```

; R3 = attributes

; R4 = address

LDR R0,=MPU_RNR      ; 0xE000ED98, MPU region number register

STR R1, [R0, #0x0]   ; Region Number

BIC R2, R2, #1       ; Disable

STRH R2, [R0, #0x8]  ; Region Size and Enable

STR R4, [R0, #0x4]   ; Region Base Address

STRH R3, [R0, #0xA]  ; Region Attribute

ORR R2, #1           ; Enable

STRH R2, [R0, #0x8]  ; Region Size and Enable

```

Software must use memory barrier instructions:

- before MPU setup if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- after MPU setup if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if you want all of the memory access behavior to take effect immediately after the programming sequence, use a `DSB` instruction and an `ISB` instruction. A `DSB` is required after changing MPU settings, such as at the end of context switch. An `ISB` is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then you do not require an `ISB`.

#### 38.4.5.8.2 Updating an MPU region using multi-word writes

You can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```

; R1 = region number

; R2 = address

; R3 = size, attributes in one

LDR R0, =MPU_RNR      ; 0xE000ED98, MPU region number register

STR R1, [R0, #0x0]   ; Region Number

STR R2, [R0, #0x4]   ; Region Base Address

STR R3, [R0, #0x8]   ; Region Attribute, Size and Enable

```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one

LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register

STM R0, {R1-R3}    ; Region Number, address, attribute, size and enable
```

You can do this in two words for pre-packed information. This means that the RBAR contains the required region number and had the VALID bit set to 1, see [Table 815](#). Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one

LDR R0, =MPU_RBAR    ; 0xE000ED9C, MPU Region Base register

STR R1, [R0, #0x0]   ; Region base address and
                    ; region number combined with VALID (bit 4) set to 1

STR R2, [R0, #0x4]   ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one

LDR R0, =MPU_RBAR    ; 0xE000ED9C, MPU Region Base register

STM R0, {R1-R2}    ; Region base address, region number and VALID bit,
                    ; and Region Attribute, Size and Enable
```

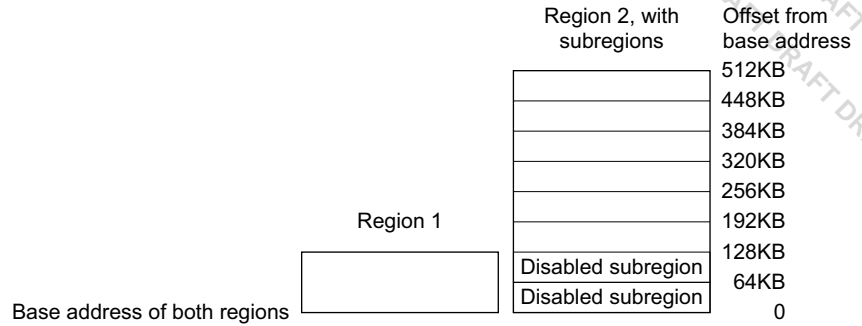
### 38.4.5.8.3 Subregions

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the RASR to disable a subregion, see [Table 816](#). The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, you must set the SRD field to 0x00, otherwise the MPU behavior is Unpredictable.

**Example of SRD use:** Two regions with the same base address overlap. Region one is 128KB, and region two is 512KB. To ensure the attributes from region one apply to the first 128KB region, set the SRD field for region two to b00000011 to disable the first two subregions, as the figure shows.





### 38.4.5.9 MPU design hints and tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- except for the RASR, it must use aligned word accesses
- for the RASR it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

#### 38.4.5.9.1 MPU configuration for a microcontroller

Usually, a microcontroller system has only a single processor and no caches. In such a system, program the MPU as follows:

**Table 821. Memory region attributes for a microcontroller**

Memory region	TEX	C	B	S	Memory type and attributes
Flash memory	b000	1	0	0	Normal memory, Non-shareable, write-through
Internal SRAM	b000	1	0	1	Normal memory, Shareable, write-through
External SRAM	b000	1	1	1	Normal memory, Shareable, write-back, write-allocate
Peripherals	b000	0	1	1	Device memory, Shareable

In most microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations. In special systems, such as multiprocessor designs or designs with a separate DMA engine, the shareability attribute might be important. In these cases refer to the recommendations of the memory device manufacturer.

## 38.5 ARM Cortex-M3 User Guide: Glossary

**Abort** — A mechanism that indicates to a processor that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory.

**Aligned** — A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

**Banked register** — A register that has multiple physical copies, where the state of the processor determines which copy is used. The Stack Pointer, SP (R13) is a banked register.

**Base register** — In instruction descriptions, a register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory. See also *Index register*.

**Big-endian (BE)** — Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory. See also *Byte-invariant*, *Endianness*, *Little-endian*.

**Big-endian memory** — Memory in which:

- a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

See also *Little-endian memory*.

**Breakpoint** — A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.

**Byte-invariant** — In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access.

An ARM byte-invariant implementation also supports unaligned halfword and word memory accesses. It expects multi-word accesses to be word-aligned.

**Cache** — A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions, data, or instructions and data. This is done to greatly increase the average speed of memory accesses and so improve processor performance.

**Condition field** — A four-bit field in an instruction that specifies a condition under which the instruction can execute.

**Context** — The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the physical address range that it can access in memory and the associated memory access permissions.

**Coprocessor** — A processor that supplements the main processor. Cortex-M3 does not support any coprocessors.

**Debugger** — A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

**Direct Memory Access (DMA)** — An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

**Doubleword** — A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

**Doubleword-aligned** — A data item having a memory address that is divisible by eight.

**Endianness** — Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping. See also *Little-endian* and *Big-endian*.

**Exception** — An event that interrupts program execution. When an exception occurs, the processor suspends the normal program flow and starts execution at the address indicated by the corresponding exception vector. The indicated address contains the first instruction of the handler for the exception.

An exception can be an interrupt request, a fault, or a software-generated system exception. Faults include attempting an invalid memory access, attempting to execute an instruction in an invalid processor state, and attempting to execute an undefined instruction.

**Exception service routine** — See also *Interrupt handler*.

**Exception vector** — See also *Interrupt vector*.

**Flat address mapping** — A system of organizing memory in which each physical address in the memory space is the same as the corresponding virtual address.

**Halfword** — A 16-bit data item.

**Illegal instruction** — An instruction that is architecturally Undefined.

**Implementation-defined** — The behavior is not architecturally defined, but is defined and documented by individual implementations.

**Implementation-specific** — The behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

**Index register** — In some load and store instruction descriptions, the value of this register is used as an offset to be added to or subtracted from the base register value to form the address that is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction. See also *Base register*.

**Instruction cycle count** — The number of cycles that an instruction occupies the Execute stage of the pipeline.

**Interrupt handler** — A program that control of the processor is passed to when an interrupt occurs.

**Interrupt vector** — One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

**Little-endian (LE)** — Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory. See also *Big-endian*, *Byte-invariant*, *Endianness*.

**Little-endian memory** — Memory in which:

- a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

See also *Big-endian memory*.

**Load/store architecture** — A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

**Memory Protection Unit (MPU)** — Hardware that controls access permissions to blocks of memory. An MPU does not perform any address translation.

**Prefetching** — In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction has to be executed.

**Read** — Reads are defined as memory operations that have the semantics of a load. Reads include the Thumb instructions `LDM`, `LDR`, `LDRSH`, `LDRH`, `LDRSB`, `LDRB`, and `POP`.

**Region** — A partition of memory space.

**Reserved** — A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

**Should Be One (SBO)** — Write as 1, or all 1s for bit fields, by software. Writing as 0 produces Unpredictable results.

**Should Be Zero (SBZ)** — Write as 0, or all 0s for bit fields, by software. Writing as 1 produces Unpredictable results.

**Should Be Zero or Preserved (SBZP)** — Write as 0, or all 0s for bit fields, by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

**Thread-safe** — In a multi-tasking environment, thread-safe functions use safeguard mechanisms when accessing shared resources, to ensure correct operation without the risk of shared access conflicts.

**Thumb instruction** — One or two halfwords that specify an operation for a processor to perform. Thumb instructions must be halfword-aligned.

**Unaligned** — A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

**Unpredictable (UNP)** — You cannot rely on the behavior. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.

**Warm reset** — Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

**WA** — See *Write-allocate*.

**WB** — See *Write-back*.

**Word** — A 32-bit data item.

**Write** — Writes are defined as operations that have the semantics of a store. Writes include the Thumb instructions `STM`, `STR`, `STRH`, `STRB`, and `PUSH`.

**Write-allocate (WA)** — In a write-allocate cache, a cache miss on storing data causes a cache line to be allocated into the cache.

**Write-back (WB)** — In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. This is also known as copyback.

**Write buffer** — A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.

**Write-through (WT)** — In a write-through cache, data is written to main memory at the same time as the cache is updated.

### 39.1 Abbreviations

Table 822. Abbreviations

Acronym	Description
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
BOD	BrownOut Detection
CAN	Controller Area Network
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
EOP	End Of Packet
ETM	Embedded Trace Macrocell
GPIO	General Purpose Input/Output
I2C	Inter-IC control bus
I2S	Inter-IC sound bus
IrDA	Infrared Data Association
JTAG	Joint Test Action Group
MII	Media Independent Interface (Ethernet related)
MIIM	Media Independent Interface Management (Ethernet related)
PHY	Physical layer interface
PLL	Phase-Locked Loop
PWM	Pulse Width Modulator
QEI	Quadrature Encoder Interface
RMI	Reduced Media Independent Interface (Ethernet related)
SE0	Single Ended Zero (USB related)
SPI	Serial Peripheral Interface
SSI	Serial Synchronous Interface
SSP	Synchronous Serial Port
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

## 39.2 Legal information

### 39.2.1 Definitions

**Draft** — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

### 39.2.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product

design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

**Evaluation products** — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out of the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US\$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

### 39.2.3 Trademarks

**Notice:** All referenced brands, product names, service names and trademarks are the property of their respective owners.

**I<sup>2</sup>C-bus** — logo is a trademark of NXP B.V.

39.3 Tables

Table 1. Ordering information . . . . .	8	address 0x400F C1A8) bit description . . . . .	52
Table 2. Ordering options for LPC178x/177x parts . . . . .	9	Table 34. Power Control registers . . . . .	55
Table 3. LPC178x/177x memory usage and details . . . . .	14	Table 35. Power Mode Control register (PCON - address 0x400F C0C0) bit description . . . . .	56
Table 4. AHB peripherals and base addresses . . . . .	17	Table 36. Encoding of reduced power modes . . . . .	57
Table 5. APB0 peripherals and base addresses . . . . .	17	Table 37. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description . . . . .	58
Table 6. APB1 peripherals and base addresses . . . . .	18	Table 38. Clock Output Configuration register (CLKOUTCFG - 0x400F C1C8) bit description . . . . .	62
Table 7. Matrix Arbitration register (Matrix_Arb - 0x400F C188) bit description . . . . .	19	Table 39. Summary of flash accelerator registers . . . . .	64
Table 8. Pin summary . . . . .	20	Table 40. Flash Accelerator Configuration register (FLASHCFG - address 0x400F C000) bit description . . . . .	65
Table 9. Summary of system control registers . . . . .	21	Table 41. Connection of interrupt sources to the Vectored Interrupt Controller . . . . .	68
Table 10. Reset Source Identification register (RSID - address 0x400F C180) bit description . . . . .	24	Table 42. NVIC register map . . . . .	71
Table 11. Reset control register 0 (RSTCON0 - 0x400F C1CC) bit description . . . . .	25	Table 43. Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100) . . . . .	72
Table 12. Reset control register 1 (RSTCON1 - 0x400F C1D0) bit description . . . . .	26	Table 44. Interrupt Set-Enable Register 1 register (ISER1 - 0xE000 E104) . . . . .	73
Table 13. External Interrupt registers . . . . .	29	Table 45. Interrupt Clear-Enable Register 0 (ICER0 - 0xE000 E180) . . . . .	74
Table 14. External Interrupt Flag register (EXTINT - address 0x400F C140) bit description . . . . .	30	Table 46. Interrupt Clear-Enable Register 1 register (ICER1 - 0xE000 E184) . . . . .	75
Table 15. External Interrupt Mode register (EXTMODE - address 0x400F C148) bit description . . . . .	30	Table 47. Interrupt Set-Pending Register 0 register (ISPR0 - 0xE000 E200) . . . . .	76
Table 16. External Interrupt Polarity register (EXTPOLAR - address 0x400F C14C) bit description . . . . .	31	Table 48. Interrupt Set-Pending Register 1 register (ISPR1 - 0xE000 E204) . . . . .	77
Table 17. System Controls and Status register (SCS - address 0x400F C1A0) bit description . . . . .	32	Table 49. Interrupt Clear-Pending Register 0 register (ICPR0 - 0xE000 E280) . . . . .	78
Table 18. Summary of system control registers . . . . .	36	Table 50. Interrupt Clear-Pending Register 1 register (ICPR1 - 0xE000 E284) . . . . .	79
Table 19. Recommended values for C <sub>X1/X2</sub> in oscillation mode (crystal and external components parameters) low frequency mode (OSCRANGE = 0, see <a href="#">Table 17</a> ) . . . . .	38	Table 51. Interrupt Active Bit Register 0 (IABR0 - 0xE000 E300) . . . . .	80
Table 20. Recommended values for C <sub>X1/X2</sub> in oscillation mode (crystal and external components parameters) high frequency mode (OSCRANGE = 1, see <a href="#">Table 17</a> ) . . . . .	38	Table 52. Interrupt Active Bit Register 1 (IABR1 - 0xE000 E304) . . . . .	81
Table 21. Clock Source Selection register (CLKSRCSEL - 0x400F C10C) bit description . . . . .	40	Table 53. Interrupt Priority Register 0 (IPR0 - 0xE000 E400) . . . . .	82
Table 22. PLL1 registers . . . . .	42	Table 54. Interrupt Priority Register 1 (IPR1 - 0xE000 E404) . . . . .	82
Table 23. PLL Control registers (PLL0CON - 0x400F C080, PLL1CON - 0x400F C0A0) bit description . . . . .	43	Table 55. Interrupt Priority Register 2 (IPR2 - 0xE000 E408) . . . . .	82
Table 24. PLL Configuration registers (PLL0CFG - 0x400F C084, PLL1CFG - 0x400F C0A4) bit description . . . . .	43	Table 56. Interrupt Priority Register 3 (IPR3 - 0xE000 E40C) . . . . .	83
Table 25. PLL Status registers (PLL0STAT - 0x400F C088, PLL1STAT - 0x400F C0A8) bit description . . . . .	43	Table 57. Interrupt Priority Register 4 (IPR4 - 0xE000 E410) . . . . .	83
Table 26. PLL Feed registers (PLL0FEED - 0x400F C08C, PLL1FEED - 0x400F C0AC) bit description . . . . .	44	Table 58. Interrupt Priority Register 5 (IPR5 - 0xE000 E414) . . . . .	83
Table 27. Elements determining PLL frequency . . . . .	46	Table 59. Interrupt Priority Register 6 (IPR6 - 0xE000 E418) . . . . .	84
Table 28. PLL Multiplier values . . . . .	47	Table 60. Interrupt Priority Register 7 (IPR7 - 0xE000 E41C) . . . . .	84
Table 29. PLL Divider values . . . . .	48	Table 61. Interrupt Priority Register 8 (IPR8 - 0xE000 E420) . . . . .	84
Table 30. CPU Clock Selection register (CCLKSEL - address 0x400F C104) bit description . . . . .	50	Table 62. Interrupt Priority Register 9 (IPR9 - 0xE000 E424) . . . . .	85
Table 31. USB Clock Selection register (USBCLKSEL - address 0x400F C108) bit description . . . . .	51		
Table 32. EMC Clock Selection register (EMCCLKSEL - address 0x400F C100) bit description . . . . .	51		
Table 33. Peripheral Clock Selection register (PCLKSEL -			



Table 63. Interrupt Priority Register 10 (IPR10 - 0xE000 E428) . . . . .	85	Table 97. GPIO overall Interrupt Status register (IOIntStatus - address 0x4002 8080) bit description . . . . .	140
Table 64. Software Trigger Interrupt Register (STIR - 0xE000 EF00) . . . . .	85	Table 98. GPIO Interrupt Enable for port 0 Rising Edge (IOIntEnR - 0x4002 8090) bit description. . . . .	141
Table 65. LPC178x/177x pin description . . . . .	86	Table 99. GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR - 0x4002 80B0) bit description . . . . .	142
Table 66. Boot control on pins P3[15]/D15 and P3[14]/D14. 106		Table 100. GPIO Interrupt Enable for port 0 Falling Edge (IOIntEnF - address 0x4002 8094) bit description 143	
Table 67. Summary of I/O pin configuration registers . . . . .	107	Table 101. GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF - 0x4002 80B4) bit description. . . . .	144
Table 68. I/O Control registers for port 0 . . . . .	111	Table 102. GPIO Interrupt Status for port 0 Rising Edge Interrupt (IOIntStatR - 0x4002 8084) bit description . . . . .	145
Table 69. I/O Control registers for port 1 . . . . .	112	Table 103. GPIO Interrupt Status for port 2 Rising Edge Interrupt (IO2IntStatR - 0x4002 80A4) bit description . . . . .	146
Table 70. I/O Control registers for port 2 . . . . .	113	Table 104. GPIO Interrupt Status for port 0 Falling Edge Interrupt (IOIntStatF - 0x4002 8088) bit description . . . . .	147
Table 71. I/O Control registers for port 3 . . . . .	114	Table 105. GPIO Interrupt Status for port 2 Falling Edge Interrupt (IO2IntStatF - 0x4002 80A8) bit description . . . . .	148
Table 72. I/O Control registers for port 4 . . . . .	115	Table 106. GPIO Interrupt Clear register for port 0 (IOIntClr - 0x4002 808C) bit description . . . . .	149
Table 73. I/O Control registers for port 5 . . . . .	116	Table 107. GPIO Interrupt Clear register for port 0 (IO2IntClr - 0x4002 80AC) bit description. . . . .	150
Table 74. Type D IOCON registers bit description . . . . .	118	Table 108. EMC configuration. . . . .	152
Table 75. Type D I/O Control registers: FUNC values and pin functions . . . . .	119	Table 109. Memory bank selection . . . . .	159
Table 76. Type A IOCON registers bit description . . . . .	124	Table 110. Pad interface and control signal descriptions 161	
Table 77. Type A I/O Control registers: FUNC values and pin functions . . . . .	125	Table 111. Summary of EMC registers . . . . .	162
Table 78. Type U IOCON registers bit description . . . . .	126	Table 112. EMC Control register (EMCControl - address 0x2009 C000) bit description . . . . .	165
Table 79. Type U I/O Control registers: FUNC values and pin functions . . . . .	126	Table 113. EMC Status register (EMCStatus - address 0x2009 C008) bit description . . . . .	165
Table 80. Type I IOCON registers bit description . . . . .	127	Table 114. EMC Configuration register (EMCConfig - address 0x2009 C008) bit description . . . . .	166
Table 81. Type I I/O Control registers: FUNC values and pin functions . . . . .	127	Table 115. Dynamic Control register (EMCDynamicControl - address 0x2009 C020) bit description . . . . .	166
Table 82. Type W IOCON registers bit description. . . . .	128	Table 116. Dynamic Memory Refresh Timer register (EMCDynamicRefresh - address 0x2009 C024) bit description. . . . .	168
Table 83. Type W I/O Control registers: FUNC values and pin functions . . . . .	128	Table 117. Dynamic Memory Read Configuration register (EMCDynamicReadConfig - address 0x2009 C028) bit description . . . . .	169
Table 84. GPIO pin description . . . . .	130	Table 118. Dynamic Memory Precharge Command Period register (EMCDynamicRP - address 0x2009 C030) bit description . . . . .	169
Table 85. GPIO register map (local bus accessible registers - enhanced GPIO features) . . . . .	131	Table 119. Dynamic Memory Active to Precharge Command Period register (EMCDynamicRAS - address 0x2009 C034) bit description . . . . .	170
Table 86. GPIO interrupt register map . . . . .	132	Table 120. Dynamic Memory Self-refresh Exit Time register (EMCDynamicSREX - address 0x2009 C038) bit description . . . . .	170
Table 87. Fast GPIO port Direction register FIO0DIR to FIO5DIR - addresses 0x2009 8000 to 0x2009 80A0) bit description . . . . .	132	Table 121. Dynamic Memory Last Data Out to Active Time register (EMCDynamicAPR - address 0x2009 C03C) bit description. . . . .	171
Table 88. Fast GPIO port Direction control byte and half-word accessible register description . . . . .	133	Table 122. Dynamic Memory Data-in to Active Command	
Table 89. Fast GPIO port output Set register (FIO0SET to FIO4SET - addresses 0x2009 8018 to 0x2009 8098) bit description . . . . .	134		
Table 90. Fast GPIO port output Set byte and half-word accessible register description . . . . .	134		
Table 91. Fast GPIO port output Clear register (FIO0CLR to FIO4CLR- addresses 0x2009 801C to 0x2009 809C) bit description . . . . .	135		
Table 92. Fast GPIO port output Clear byte and half-word accessible register description . . . . .	136		
Table 93. Fast GPIO port Pin value register (FIO0PIN to FIO4PIN- addresses 0x2009 8014 to 0x2009 8094) bit description . . . . .	137		
Table 94. Fast GPIO port Pin value byte and half-word accessible register description . . . . .	137		
Table 95. Fast GPIO port Mask register (FIO0MASK to FIO4MASK - addresses 0x2009 8010 to 0x2009 8090) bit description . . . . .	138		
Table 96. Fast GPIO port Mask byte and half-word accessible register description . . . . .	139		

Time register (EMCDynamicDAL - address 0x2009 C040) bit description . . . . .	171	0x2009 C238, 0x2009 C258, 0x2009 C278) bit description . . . . .	182
Table 123. Dynamic Memory Write recover Time register (EMCDynamicWR - address 0x2009 C044) bit description . . . . .	172	Table 140. Delay Control register (EMCDLYCTL - 0x400F C1DC) bit description. . . . .	183
Table 124. Dynamic Memory Active to Active Command Period register (EMCDynamicRC - address 0x2009 C048) bit description . . . . .	172	Table 141. EMC Calibration register (EMCCAL - 0x400F C1E0) bit description . . . . .	184
Table 125. Dynamic Memory Auto-refresh Period register (EMCDynamicRFC - address 0x2009 C04C) bit description . . . . .	173	Table 142. Ethernet acronyms, abbreviations, and definitions	189
Table 126. Dynamic Memory Exit Self-refresh register (EMCDynamicXSR - address 0x2009 C050) bit description . . . . .	173	Table 143. Example PHY Devices . . . . .	195
Table 127. Dynamic Memory Active Bank A to Active Bank B Time register (EMCDynamicRRD - address 0x2009 C054) bit description . . . . .	174	Table 144. Ethernet MII pin descriptions. . . . .	196
Table 128. Dynamic Memory Load Mode register to Active Command Time (EMCDynamicMRD - address 0x2009 C058) bit description . . . . .	174	Table 145. Ethernet RMI pin descriptions . . . . .	196
Table 129. Static Memory Extended Wait register (EMCStaticExtendedWait - address 0x2009 C080) bit description . . . . .	174	Table 146. Ethernet MIIM pin descriptions . . . . .	196
Table 130. Dynamic Memory Configuration registers (EMCDynamicConfig0-3 - address 0x2009 C100, 0x2009 C120, 0x2009 C140, 0x2009 C160) bit description . . . . .	175	Table 147. Ethernet register definitions . . . . .	197
Table 131. Address mapping. . . . .	175	Table 148. MAC Configuration register 1 (MAC1 - address 0x2008 4000) bit description . . . . .	199
Table 132. Dynamic Memory RAS & CAS Delay registers (EMCDynamicRasCas0-3 - address 0x2009 C104, 0x2009 C124, 0x2009 C144, 0x2009 C164) bit description . . . . .	177	Table 149. MAC Configuration register 2 (MAC2 - address 0x2008 4004) bit description . . . . .	199
Table 133. Static Memory Configuration registers (EMCStaticConfig0-3 - address 0x2009 C200, 0x2009 C220, 0x2009 C240, 0x2009 C260) bit description . . . . .	178	Table 150. Pad operation. . . . .	200
Table 134. Static Memory Write Enable Delay registers (EMCStaticWaitWen0-3 - address 0x2009 C204, 0x2009 C224, 0x2009 C244, 0x2009 C264) bit description . . . . .	180	Table 151. Back-to-back Inter-packet-gap register (IPGT - address 0x2008 4008) bit description . . . . .	201
Table 135. Static Memory Output Enable delay registers (EMCStaticWaitOen03 - address 0x2009 C208, 0x2009 C228, 0x2009 C248, 0x2009 C268) bit description . . . . .	180	Table 152. Non Back-to-back Inter-packet-gap register (IPGR - address 0x2008 400C) bit description . . . . .	201
Table 136. Static Memory Read Delay registers (EMCStaticWaitRd0-3 - address 0x2009 C20C, 0x2009 C22C, 0x2009 C24C, 0x2009 C26C) bit description . . . . .	181	Table 153. Collision Window / Retry register (CLRT - address 0x2008 4010) bit description . . . . .	202
Table 137. Static Memory Page Mode Read Delay registers0-3 (EMCStaticWaitPage0-3 - address 0x2009 C210, 0x2009 C230, 0x2009 C250, 0x2009 C270) bit description . . . . .	181	Table 154. Maximum Frame register (MAXF - address 0x2008 4014) bit description . . . . .	202
Table 138. Static Memory Write Delay registers0-3 (EMCStaticWaitWr - address 0x2009 C214, 0x2009 C234, 0x2009 C254, 0x2009 C274) bit description . . . . .	182	Table 155. PHY Support register (SUPP - address 0x2008 4018) bit description . . . . .	202
Table 139. Static Memory Turn Round Delay registers0-3 (EMCStaticWaitTurn0-3 - address 0x2009 C218,		Table 156. Test register (TEST - address 0x2008 401C) bit description . . . . .	203
		Table 157. MII Mgmt Configuration register (MCFG - address 0x2008 4020) bit description . . . . .	203
		Table 158. Clock select encoding . . . . .	203
		Table 159. MII Mgmt Command register (MCMD - address 0x2008 4024) bit description . . . . .	204
		Table 160. MII Mgmt Address register (MADR - address 0x2008 4028) bit description . . . . .	204
		Table 161. MII Mgmt Write Data register (MWTD - address 0x2008 402C) bit description . . . . .	204
		Table 162. MII Mgmt Read Data register (MRDD - address 0x2008 4030) bit description . . . . .	205
		Table 163. MII Mgmt Indicators register (MIND - address 0x2008 4034) bit description . . . . .	205
		Table 164. Station Address register (SA0 - address 0x2008 4040) bit description . . . . .	206
		Table 165. Station Address register (SA1 - address 0x2008 4044) bit description . . . . .	206
		Table 166. Station Address register (SA2 - address 0x2008 4048) bit description . . . . .	206
		Table 167. Command register (Command - address 0x2008 4100) bit description . . . . .	207
		Table 168. Status register (Status - address 0x2008 4104) bit description . . . . .	207
		Table 169. Receive Descriptor Base Address register (RxDescriptor - address 0x2008 4108) bit description . . . . .	208
		Table 170. Receive Status Base Address register (RxStatus	

- address 0x2008 410C) bit description . . . . .	208	Table 195. Receive Descriptor Control Word . . . . .	221
Table 171. Receive Number of Descriptors register (RxDescriptor - address 0x2008 4110) bit description . . . . .	208	Table 196. Receive Status Fields . . . . .	221
Table 172. Receive Produce Index register (RxProduceIndex - address 0x2008 4114) bit description . . . . .	209	Table 197. Receive Status HashCRC Word . . . . .	222
Table 173. Receive Consume Index register (RxConsumeIndex - address 0x2008 4118) bit description . . . . .	209	Table 198. Receive status information word . . . . .	222
Table 174. Transmit Descriptor Base Address register (TxDescriptor - address 0x2008 411C) bit description . . . . .	209	Table 199. Transmit descriptor fields . . . . .	224
Table 175. Transmit Status Base Address register (TxStatus - address 0x2008 4120) bit description . . . . .	210	Table 200. Transmit descriptor control word . . . . .	224
Table 176. Transmit Number of Descriptors register (TxDescriptorNumber - address 0x2008 4124) bit description . . . . .	210	Table 201. Transmit status fields . . . . .	224
Table 177. Transmit Produce Index register (TxProduceIndex - address 0x2008 4128) bit description . . . . .	210	Table 202. Transmit status information word . . . . .	225
Table 178. Transmit Consume Index register (TxConsumeIndex - address 0x2008 412C) bit description . . . . .	211	Table 203. LCD controller pins . . . . .	267
Table 179. Transmit Status Vector 0 register (TSV0 - address 0x2008 4158) bit description. . . . .	211	Table 204. Pins used for single panel STN displays. . . . .	267
Table 180. Transmit Status Vector 1 register (TSV1 - address 0x2008 415C) bit description . . . . .	212	Table 205. Pins used for dual panel STN displays . . . . .	268
Table 181. Receive Status Vector register (RSV - address 0x2008 4160) bit description . . . . .	212	Table 206. Pins used for TFT displays . . . . .	268
Table 182. Flow Control Counter register (FlowControlCounter - address 0x2008 4170) bit description . . . . .	213	Table 207. FIFO bits for Little-endian Byte, Little-endian Pixel order. . . . .	272
Table 183. Flow Control Status register (FlowControlStatus - address 0x2008 4174) bit description. . . . .	214	Table 208. FIFO bits for Big-endian Byte, Big-endian Pixel order. . . . .	273
Table 184. Receive Filter Control register (RxFilterCtrl - address 0x2008 4200) bit description. . . . .	215	Table 209. FIFO bits for Little-endian Byte, Big-endian Pixel order. . . . .	274
Table 185. Receive Filter WoL Status register (RxFilterWoLStatus - address 0x2008 4204) bit description . . . . .	215	Table 210. RGB mode data formats . . . . .	275
Table 186. Receive Filter WoL Clear register (RxFilterWoLClear - address 0x2008 4208) bit description . . . . .	216	Table 211. Palette data storage for TFT modes. . . . .	276
Table 187. Hash Filter Table LSBs register (HashFilterL - address 0x2008 4210) bit description. . . . .	216	Table 212. Palette data storage for STN color modes. . . . .	276
Table 188. Hash Filter MSBs register (HashFilterH - address 0x2008 4214) bit description . . . . .	216	Table 213. Palette data storage for STN monochrome mode. 277	
Table 189. Interrupt Status register (IntStatus - address 0x2008 4FE0) bit description . . . . .	217	Table 214. Palette data storage for STN monochrome mode. 278	
Table 190. Interrupt Enable register (intEnable - address 0x2008 4FE4) bit description . . . . .	217	Table 215. Addresses for 32 x 32 cursors . . . . .	280
Table 191. Interrupt Clear register (IntClear - address 0x2008 4FE8) bit description . . . . .	218	Table 216. Buffer to pixel mapping for 32 x 32 pixel cursor format. . . . .	281
Table 192. Interrupt Set register (IntSet - address 0x2008 4FEC) bit description . . . . .	219	Table 217. Buffer to pixel mapping for 64 x 64 pixel cursor format. . . . .	281
Table 193. Power-Down register (PowerDown - address 0x2008 4FF4) bit description . . . . .	219	Table 218. Pixel encoding. . . . .	282
Table 194. Receive Descriptor Fields . . . . .	221	Table 219. Color display driven with 2 2/3 pixel data . . . . .	282
		Table 220. Summary of LCD controller registers . . . . .	286
		Table 221. LCD Configuration register (LCD_CFG, RW - 0x400F C1B8) . . . . .	287
		Table 222. Horizontal Timing register (LCD_TIMH, RW - 0x2008 8000) . . . . .	287
		Table 223. Vertical Timing register (LCD_TIMV, RW - 0x2008 8004) . . . . .	288
		Table 224. Clock and Signal Polarity register (LCD_POL, RW - 0x2008 8008) . . . . .	289
		Table 225. Line End Control register (LCD_LE, RW - 0x2008 800C) . . . . .	291
		Table 226. Upper Panel Frame Base register (LCD_UPBASE, RW - 0x2008 8010) . . . . .	291
		Table 227. Lower Panel Frame Base register (LCD_LPBASE, RW - 0x2008 8014) . . . . .	292
		Table 228. LCD Control register (LCD_CTRL, RW - 0x2008 8018) . . . . .	292
		Table 229. Interrupt Mask register (LCD_INTMSK, RW - 0x2008 801C) . . . . .	294
		Table 230. Raw Interrupt Status register (LCD_INTRAW, RW - 0x2008 8020) . . . . .	294
		Table 231. Masked Interrupt Status register (LCD_INTSTAT, RW - 0x2008 8024) . . . . .	295
		Table 232. Interrupt Clear register (LCD_INTCLR, RW - 0x2008 8028) . . . . .	295
		Table 233. Upper Panel Current Address register	

(LCD_UPCURR, RW - 0x2008 802C) . . . . .	296	Table 263. USB Device Interrupt Clear register (USBDevIntClr - address 0x2008 C208) bit allocation . . . . .	320
Table 234. Lower Panel Current Address register (LCD_LPCURR, RW - 0x2008 8030) . . . . .	296	Table 264. USB Device Interrupt Clear register (USBDevIntClr - address 0x2008 C208) bit description . . . . .	320
Table 235. Color Palette registers (LCD_PAL, RW - 0x2008 8200 to 0x2008 83FC) . . . . .	296	Table 265. USB Device Interrupt Set register (USBDevIntSet - address 0x2008 C20C) bit allocation . . . . .	320
Table 236. Cursor Image registers (CRSR_IMG, RW - 0x2008 8800 to 0x2008 8BFC) . . . . .	297	Table 266. USB Device Interrupt Set register (USBDevIntSet - address 0x2008 C20C) bit description . . . . .	321
Table 237. Cursor Control register (CRSR_CTRL, RW - 0x2008 8C00) . . . . .	298	Table 267. USB Device Interrupt Priority register (USBDevIntPri - address 0x2008 C22C) bit description . . . . .	321
Table 238. Cursor Configuration register (CRSR_CFG, RW - 0x2008 8C04) . . . . .	298	Table 268. USB Endpoint Interrupt Status register (USBEPIntSt - address 0x2008 C230) bit allocation . . . . .	322
Table 239. Cursor Palette register 0 (CRSR_PAL0, RW - 0x2008 8C08) . . . . .	299	Table 269. USB Endpoint Interrupt Status register (USBEPIntSt - address 0x2008 C230) bit description . . . . .	322
Table 240. Cursor Palette register 1 (CRSR_PAL1, RW - 0x2008 8C0C) . . . . .	299	Table 270. USB Endpoint Interrupt Enable register (USBEPIntEn - address 0x2008 C234) bit allocation . . . . .	323
Table 241. Cursor XY Position register (CRSR_XY, RW - 0x2008 8C10) . . . . .	299	Table 271. USB Endpoint Interrupt Enable register (USBEPIntEn - address 0x2008 C234) bit description . . . . .	323
Table 242. Cursor Clip Position register (CRSR_CLIP, RW - 0x2008 8C14) . . . . .	300	Table 272. USB Endpoint Interrupt Clear register (USBEPIntClr - address 0x2008 C238) bit allocation . . . . .	324
Table 243. Cursor Interrupt Mask register (CRSR_INTMSK, RW - 0x2008 8C20) . . . . .	300	Table 273. USB Endpoint Interrupt Clear register (USBEPIntClr - address 0x2008 C238) bit description . . . . .	324
Table 244. Cursor Interrupt Clear register (CRSR_INTCLR, RW - 0x2008 8C24) . . . . .	301	Table 274. USB Endpoint Interrupt Set register (USBEPIntSet - address 0x2008 C23C) bit allocation . . . . .	324
Table 245. Cursor Raw Interrupt Status register (CRSR_INTRAW, RW - 0x2008 8C28) . . . . .	301	Table 275. USB Endpoint Interrupt Set register (USBEPIntSet - address 0x2008 C23C) bit description . . . . .	325
Table 246. Cursor Masked Interrupt Status register (CRSR_INTSTAT, RW - 0x2008 8C2C) . . . . .	301	Table 276. USB Endpoint Interrupt Priority register (USBEPIntPri - address 0x2008 C240) bit allocation . . . . .	325
Table 247. LCD panel connections for STN single panel mode . . . . .	305	Table 277. USB Endpoint Interrupt Priority register (USBEPIntPri - address 0x2008 C240) bit description . . . . .	325
Table 248. LCD panel connections for STN dual panel mode 306		Table 278. USB Realize Endpoint register (USBReEp - address 0x2008 C244) bit allocation . . . . .	326
Table 249. LCD panel connections for TFT panels . . . . .	307	Table 279. USB Realize Endpoint register (USBReEp - address 0x2008 C244) bit description . . . . .	327
Table 250. USB related acronyms, abbreviations, and definitions used in this chapter . . . . .	309	Table 280. USB Endpoint Index register (USBEPIn - address 0x2008 C248) bit description . . . . .	328
Table 251. Fixed endpoint configuration . . . . .	310	Table 281. USB MaxPacketSize register (USBMaxPSize - address 0x2008 C24C) bit description . . . . .	328
Table 252. USB external interface . . . . .	313	Table 282. USB Receive Data register (USBRxData - address 0x2008 C218) bit description . . . . .	329
Table 253. USB device controller clock sources . . . . .	314	Table 283. USB Receive Packet Length register (USBRxPlen - address 0x2008 C220) bit description . . . . .	329
Table 254. USB device register map . . . . .	315	Table 284. USB Transmit Data register (USBTxData - address 0x2008 C21C) bit description . . . . .	329
Table 255. USB Port Select register (USBPortSel - 0x2008 C110) bit description . . . . .	316		
Table 256. USBClkCtrl register (USBClkCtrl - address 0x2008 CFF4) bit description . . . . .	317		
Table 257. USB Clock Status register (USBClkSt - address 0x2008 CFF8) bit description . . . . .	317		
Table 258. USB Interrupt Status register (USBIntSt - address 0x2008 C1C0) bit description . . . . .	318		
Table 259. USB Device Interrupt Status register (USBDevIntSt - address 0x2008 C200) bit allocation . . . . .	318		
Table 260. USB Device Interrupt Status register (USBDevIntSt - address 0x2008 C200) bit description . . . . .	319		
Table 261. USB Device Interrupt Enable register (USBDevIntEn - address 0x2008 C204) bit allocation . . . . .	319		
Table 262. USB Device Interrupt Enable register (USBDevIntEn - address 0x2008 C204) bit description . . . . .	319		

Table 285. USB Transmit Packet Length register (USBTxPLen - address 0x2008 C224) bit description . . . . .	330	Table 308. SIE command code table . . . . .	342
Table 286. USB Control register (USBCtrl - address 0x2008 C228) bit description . . . . .	330	Table 309. Set Address command bit description. . . . .	342
Table 287. USB Command Code register (USBCmdCode - address 0x2008 C210) bit description . . . . .	331	Table 310. Configure Device command bit description. . . . .	343
Table 288. USB Command Data register (USBCmdData - address 0x2008 C214) bit description . . . . .	331	Table 311. Set Mode command bit description. . . . .	343
Table 289. USB DMA Request Status register (USBDMARSt - address 0x2008 C250) bit allocation . . . . .	332	Table 312. Set Device Status command bit description . . . . .	344
Table 290. USB DMA Request Status register (USBDMARSt - address 0x2008 C250) bit description . . . . .	332	Table 313. Get Error Code command bit description . . . . .	346
Table 291. USB DMA Request Clear register (USBDMARClr - address 0x2008 C254) bit description . . . . .	333	Table 314. Read Error Status command bit description . . . . .	346
Table 292. USB DMA Request Set register (USBDMARSet - address 0x2008 C258) bit description . . . . .	333	Table 315. Select Endpoint command bit description . . . . .	347
Table 293. USB UDCA Head register (USBUDCAH - address 0x2008 C280) bit description . . . . .	333	Table 316. Set Endpoint Status command bit description. . . . .	348
Table 294. USB EP DMA Status register (USBEPDMASt - address 0x2008 C284) bit description . . . . .	334	Table 317. Clear Buffer command bit description . . . . .	349
Table 295. USB EP DMA Enable register (USBEPDMAEn - address 0x2008 C288) bit description . . . . .	334	Table 318. DMA descriptor . . . . .	356
Table 296. USB EP DMA Disable register (USBEPDMADis - address 0x2008 C28C) bit description . . . . .	334	Table 319. USB (OHCI) related acronyms and abbreviations used in this chapter . . . . .	370
Table 297. USB DMA Interrupt Status register (USBDMAIntSt - address 0x2008 C290) bit description . . . . .	335	Table 320. USB OTG port pins . . . . .	372
Table 298. USB DMA Interrupt Enable register (USBDMAIntEn - address 0x2008 C294) bit description . . . . .	335	Table 321. USB Host register address definitions . . . . .	373
Table 299. USB End of Transfer Interrupt Status register (USBEoTIntSt - address 0x2008 C2A0) bit description . . . . .	335	Table 322. USB OTG port 1 pins . . . . .	377
Table 300. USB End of Transfer Interrupt Clear register (USBEoTIntClr - address 0x2008 C2A4) bit description . . . . .	336	Table 323. USB OTG and I2C register address definitions . . . . .	382
Table 301. USB End of Transfer Interrupt Set register (USBEoTIntSet - address 0x2008 C2A8) bit description . . . . .	336	Table 324. USB Interrupt Status register - (USBIntSt - address 0x2008 C1C0) bit description. . . . .	382
Table 302. USB New DD Request Interrupt Status register (USBNDDRIntSt - address 0x2008 C2AC) bit description . . . . .	336	Table 325. OTG Interrupt Status register (OTGIntSt - address 0x2008 C100) bit description . . . . .	383
Table 303. USB New DD Request Interrupt Clear register (USBNDDRIntClr - address 0x2008 C2B0) bit description . . . . .	336	Table 326. OTG Status Control register (OTGStCtrl - address 0x2008 C110) bit description . . . . .	384
Table 304. USB New DD Request Interrupt Set register (USBNDDRIntSet - address 0x2008 C2B4) bit description . . . . .	337	Table 327. OTG Timer register (OTGTmr - address 0x2008 C114) bit description . . . . .	385
Table 305. USB System Error Interrupt Status register (USBSysErrIntSt - address 0x2008 C2B8) bit description . . . . .	337	Table 328. OTG clock control register (OTGClkCtrl - address 0x2008 CFF4) bit description. . . . .	385
Table 306. USB System Error Interrupt Clear register (USBSysErrIntClr - address 0x2008 C2BC) bit description . . . . .	337	Table 329. OTG clock status register (OTGClkSt - address 0x2008 CFF8) bit description. . . . .	386
Table 307. USB System Error Interrupt Set register (USBSysErrIntSet - address 0x2008 C2C0) bit description . . . . .	337	Table 330. I2C Receive register (I2C_RX - address 0x2008 C300) bit description . . . . .	386
		Table 331. I2C Transmit register (I2C_TX - address 0x2008 C300) bit description . . . . .	387
		Table 332. I2C status register (I2C_STS - address 0x2008 C304) bit description . . . . .	387
		Table 333. I2C Control register (I2C_CTL - address 0x2008 C308) bit description . . . . .	388
		Table 334. I2C_CLKHI register (I2C_CLKHI - address 0x2008 C30C) bit description. . . . .	390
		Table 335. I2C_CLKLO register (I2C_CLKLO - address 0x2008 C310) bit description . . . . .	390
		Table 336. SD/MMC card interface pin description. . . . .	406
		Table 337. Command format . . . . .	410
		Table 338. Simple response format . . . . .	410
		Table 339. Long response format . . . . .	411
		Table 340. Command path status flags. . . . .	411
		Table 341. CRC token status . . . . .	414
		Table 342. Data path status flags . . . . .	415
		Table 343. Transmit FIFO status flags . . . . .	416
		Table 344. Receive FIFO status flags. . . . .	417
		Table 345. Summary of SD card interface registers . . . . .	418
		Table 346: Power Control register (MCIPWR - address 0x400C 0000) bit description . . . . .	418
		Table 347: Clock Control register (MCIClock - address 0x400C 0004) bit description . . . . .	419
		Table 348: Argument register (MCIArgument - address 0x400C 0008) bit description . . . . .	420

Table 349: Command register (MCICommand - address 0x400C 000C) bit description . . . . .	420	Table 377: UART1 Line Status Register (U1LSR - address 0x4001 0014) bit description . . . . .	440
Table 350: Command Response Types. . . . .	420	Table 378: UART1 Modem Status Register (U1MSR - address 0x4001 0018) bit description . . . . .	441
Table 351: Command Response register (MCIRspCommand - address 0x400C 0010) bit description . . . . .	421	Table 379: UART1 Scratch Pad Register (U1SCR - address 0x4001 0014) bit description . . . . .	442
Table 352: Response registers (MCIResponse0-3 - addresses 0x400C 0014, 0x400C 0018, 0x400C 001C and 0x400C 0020) bit description . . . . .	421	Table 380: Auto-baud Control Register (U1ACR - address 0x4001 0020) bit description . . . . .	442
Table 353: Response Register Type . . . . .	421	Table 381: UART1 Fractional Divider Register (U1FDR - address 0x4001 0028) bit description . . . . .	446
Table 354: Data Timer register (MCIDataTimer - address 0x400C 0024) bit description . . . . .	421	Table 382: Fractional Divider setting look-up table . . . . .	448
Table 355: Data Length register (MCIDataLength - address 0x400C 0028) bit description . . . . .	422	Table 383: UART1 Transmit Enable Register (U1TER - address 0x4001 0030) bit description . . . . .	449
Table 356: Data Control register (MCIDataCtrl - address 0x400C 002C) bit description . . . . .	422	Table 384: UART1 RS485 Control register (U1RS485CTRL - address 0x4001 004C) bit description . . . . .	449
Table 357: Data Block Length . . . . .	423	Table 385: UART1 RS-485 Address Match register (U1RS485ADRMATCH - address 0x4001 0050) bit description. . . . .	450
Table 358: Data Counter register (MCIDataCnt - address 0x400C 0030) bit description . . . . .	423	Table 386: UART1 RS-485 Delay value register (U1RS485DLY - address 0x4001 0054) bit description . . . . .	450
Table 359: Status register (MCIStatus - address 0x400C 0034) bit description . . . . .	423	Table 387: UARTn Pin description . . . . .	454
Table 360: Clear register (MCIClear - address 0x400C 0038) bit description . . . . .	424	Table 388: UART0/2/3 Register Map . . . . .	455
Table 361: Interrupt Mask registers (MCIMask0 - address 0x400C 003C) bit description . . . . .	425	Table 389: UARTn Receiver Buffer Register (U0RBR - address 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 0x4009 C000 when DLAB = 0) bit description . . . . .	456
Table 362: FIFO Counter register (MCIFifoCnt - address 0x400C 0048) bit description . . . . .	425	Table 390: UARTn Transmit Holding Register (U0THR - address 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0) bit description . . . . .	457
Table 363: Data FIFO register (MCIFIFO - address 0x400C 0080 to 0x400C 00BC) bit description. . . . .	426	Table 391: UARTn Divisor Latch LSB register (U0DLL - address 0x4000 C000, U2DLL - 0x4009 8000, U3DLL - 0x4009 C000 when DLAB = 1) bit description . . . . .	457
Table 364: UART1 Pin Description . . . . .	430	Table 392: UARTn Divisor Latch MSB register (U0DLM - address 0x4000 C004, U2DLM - 0x4009 8004, U3DLM - 0x4009 C004 when DLAB = 1) bit description . . . . .	457
Table 365: UART1 register map . . . . .	431	Table 393: UARTn Interrupt Enable Register (U0IER - address 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0) bit description . . . . .	457
Table 366: UART1 Receiver Buffer Register (U1RBR - address 0x4001 0000 when DLAB = 0) bit description . . . . .	432	Table 394: UARTn Interrupt Identification Register (U0IIR - address 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008) bit description. . . . .	458
Table 367: UART1 Transmitter Holding Register (U1THR - address 0x4001 0000 when DLAB = 0) bit description . . . . .	432	Table 395: UARTn Interrupt Handling. . . . .	459
Table 368: UART1 Divisor Latch LSB Register (U1DLL - address 0x4001 0000 when DLAB = 1) bit description . . . . .	433	Table 396: UARTn FIFO Control Register (U0FCR - address 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4007 C008) bit description . . . . .	460
Table 369: UART1 Divisor Latch MSB Register (U1DLM - address 0x4001 0004 when DLAB = 1) bit description . . . . .	433	Table 397: UARTn Line Control Register (U0LCR - address 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) bit description. . . . .	461
Table 370: UART1 Interrupt Enable Register (U1IER - address 0x4001 0004 when DLAB = 0) bit description . . . . .	433	Table 398: UARTn Line Status Register (U0LSR - address 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) bit description . . . . .	462
Table 371: UART1 Interrupt Identification Register (U1IIR - address 0x4001 0008) bit description. . . . .	434	Table 399: UARTn Scratch Pad Register (U0SCR - address 0x4000 C01C, U2SCR - 0x4009 801C, U3SCR -	
Table 372: UART1 Interrupt Handling . . . . .	435		
Table 373: UART1 FIFO Control Register (U1FCR - address 0x4001 0008) bit description . . . . .	436		
Table 374: UART1 Line Control Register (U1LCR - address 0x4001 000C) bit description . . . . .	437		
Table 375: UART1 Modem Control Register (U1MCR - address 0x4001 0010) bit description. . . . .	438		
Table 376: Modem status interrupt generation . . . . .	439		

0x4009 C01C) bit description . . . . .	463	Table 426. UART4 Smart Card Interface Control register (U4SCICTRL - 0x400A 4048) bit description	491
Table 400: UARTn Auto-baud Control Register (U0ACR - address 0x4000 C020, U2ACR - 0x4009 8020, U3ACR - 0x4009 C020) bit description. . . . .	464	Table 427: UART4 RS485 Control register (U4RS485CTRL - 0x400A 404C) bit description. . . . .	492
Table 401: UARTn Fractional Divider Register (U0FDR - address 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028) bit description. . . . .	467	Table 428. UART4 RS-485 Address Match register (U4RS485ADRMATCH - 0x400A 4050) bit description . . . . .	493
Table 402. Fractional Divider setting look-up table . . . . .	469	Table 429. UART4 RS-485 Delay value register (U4RS485DLY - 0x400A 4054) bit description	493
Table 403: UARTn Transmit Enable Register (U0TER - address 0x4000 C030, U2TER - 0x4009 8030, U3TER - 0x4009 C030) bit description. . . . .	470	Table 430. UART4 Synchronous mode control register (U4SYNCCTRL - 0x400A 4058) bit description . . . . .	495
Table 404: UARTn RS485 Control register (U0RS485CTRL - address 0x4000 C04C, U2RS485CTRL - 0x4009 804C, U3RS485CTRL - 0x4009 C04C) bit description . . . . .	470	Table 431: UART4 Transmit Enable Register (U4TER - 0x400A 4030) bit description . . . . .	497
Table 405. UARTn RS-485 Address Match register (U0RS485ADRMATCH - address 0x4000 C050, U2RS485ADRMATCH - 0x4009 8050, U3RS485ADRMATCH - 0x4009 C050) bit description . . . . .	471	Table 432. CAN Pin descriptions . . . . .	499
Table 406. UARTn RS-485 Delay value register (U0RS485DLY - address 0x4000 0054, U2RS485DLY - 0x4009 8054, U3RS485DLY - 0x4009 C054) bit description	471	Table 433. Memory map of the CAN block . . . . .	504
Table 407: UART4 Pin description. . . . .	475	Table 434. CAN acceptance filter and central CAN registers	504
Table 408. UART4 Register Map. . . . .	476	Table 435. CAN1 and CAN2 controller register map . . . . .	504
Table 409: UART4 Receiver Buffer Register (U4RBR - 0x400A 4000 when DLAB = 0) bit description . . . . .	477	Table 436. CAN1 and CAN2 controller register summary	506
Table 410: UART4 Transmit Holding Register (U4THR - 0x400A 4000 when DLAB = 0) bit description	477	Table 437. CAN Wake and Sleep registers. . . . .	506
Table 411: UART4 Divisor Latch LSB register (U4DLL - 0x400A 4000 when DLAB = 1) bit description	478	Table 438. CAN Mode register (CAN1MOD - address 0x4004 4000, CAN2MOD - address 0x4004 8000) bit description . . . . .	507
Table 412: UART4 Divisor Latch MSB register (U4DLL - 0x400A 4004 when DLAB = 1) bit description	478	Table 439. CAN Command Register (CAN1CMR - address 0x4004 4004, CAN2CMR - address 0x4004 8004) bit description. . . . .	508
Table 413: UART4 Interrupt Enable Register (U4IER - 0x400A 4004 when DLAB = 0) bit description	478	Table 440. CAN Global Status Register (CAN1GSR - address 0x4004 4008, CAN2GSR - address 0x4004 8008) bit description . . . . .	510
Table 414: UART4 Interrupt Identification Register (U4IIR - 0x400A 4008) bit description . . . . .	479	Table 441. CAN Interrupt and Capture Register (CAN1ICR - address 0x4004 400C, CAN2ICR - address 0x4004 800C) bit description . . . . .	512
Table 415: UART4 Interrupt Handling . . . . .	480	Table 442. CAN Interrupt Enable Register (CAN1IER - address 0x4004 4010, CAN2IER - address 0x4004 8010) bit description . . . . .	515
Table 416: UART4 FIFO Control Register (U4FCR - 0x400A 4008) bit description . . . . .	480	Table 443. CAN Bus Timing Register (CAN1BTR - address 0x4004 4014, CAN2BTR - address 0x4004 8014) bit description. . . . .	516
Table 417: UART4 Line Control Register (U4LCR - 0x400A 400C) bit description . . . . .	481	Table 444. CAN Error Warning Limit register (CAN1EWL - address 0x4004 4018, CAN2EWL - address 0x4004 8018) bit description . . . . .	518
Table 418: UART4 Line Status Register (U4LSR - 0x400A 4014) bit description . . . . .	482	Table 445. CAN Status Register (CAN1SR - address 0x4004 401C, CAN2SR - address 0x4004 801C) bit description. . . . .	518
Table 419: UART4 Scratch Pad Register (U4SCR - 0x400A 401C) bit description . . . . .	483	Table 446. CAN Receive Frame Status register (CAN1RFS - address 0x4004 4020, CAN2RFS - address 0x4004 8020) bit description . . . . .	520
Table 420: UART4 Auto-baud Control Register (U4ACR - 0x400A 4020) bit description . . . . .	484	Table 447. CAN Receive Identifier register (CAN1RID - address 0x4004 4024, CAN2RID - address 0x4004 8024) bit description . . . . .	521
Table 421: UART4 IrDA Control Register (U4ICR - 0x400A 4024) bit description . . . . .	486	Table 448. RX Identifier register when FF = 1 . . . . .	521
Table 422: IrDA Pulse Width . . . . .	487	Table 449. CAN Receive Data register A (CAN1RDA - address 0x4004 4028, CAN2RDA - address 0x4004 8028) bit description . . . . .	521
Table 423: UART4 Fractional Divider Register (U4FDR - 0x400A 4028) bit description . . . . .	487	Table 450. CAN Receive Data register B (CAN1RDB - address 0x4004 402C, CAN2RDB - address 0x4004 802C) bit description . . . . .	521
Table 424. Fractional Divider setting look-up table . . . . .	490		
Table 425. UART4 Oversampling Register (U4OSR - 0x400A 402C) bit description . . . . .	491		

Table 451. CAN Transmit Frame Information register (CAN1TFI[1/2/3] - address 0x4004 40[30/40/50], CAN2TFI[1/2/3] - 0x4004 80[30/40/50]) bit description . . . . .	522	Table 477. Used ID-Look-up Table sections . . . . .	551
Table 452. CAN Transfer Identifier register (CAN1TID[1/2/3] - address 0x4004 40[34/44/54], CAN2TID[1/2/3] - address 0x4004 80[34/44/54]) bit description. . . . .	523	Table 478. Used ID-Look-up Table sections . . . . .	552
Table 453. Transfer Identifier register when FF = 1 . . . . .	523	Table 479. SSP pin descriptions . . . . .	557
Table 454. CAN Transmit Data register A (CAN1TDA[1/2/3] - address 0x4004 40[38/48/58], CAN2TDA[1/2/3] - address 0x4004 80[38/48/58]) bit description. . . . .	524	Table 480. SSP Register Map . . . . .	565
Table 455. CAN Transmit Data register B (CAN1TDB[1/2/3] - address 0x4004 40[3C/4C/5C], CAN2TDB[1/2/3] - address 0x4004 80[3C/4C/5C]) bit description. . . . .	524	Table 481: SSPn Control Register 0 (SSP0CR0 - address 0x4008 8000, SSP1CR0 - 0x4003 0000, SSP2CR0 - 0x400A C000) bit description . . . . .	566
Table 456. CAN Sleep Clear register (CANSLEEPCLR - address 0x400F C110) bit description . . . . .	524	Table 482: SSPn Control Register 1 (SSP0CR1 - address 0x4008 8004, SSP1CR1 - 0x4003 0004, SSP2CR1 - 0x400A C004) bit description . . . . .	567
Table 457. CAN Wake-up Flags register (CANWAKEFLAGS - address 0x400F C114) bit description . . . . .	525	Table 483: SSPn Data Register (SSP0DR - address 0x4008 8008, SSP1DR - 0x4003 0008, SSP2DR - 0x400A C008) bit description. . . . .	567
Table 458. Central Transit Status Register (CANTxSR - address 0x4004 0000) bit description. . . . .	526	Table 484: SSPn Status Register (SSP0SR - address 0x4008 800C, SSP1SR - 0x4003 000C, SSP2SR - 0x400A C00C) bit description . . . . .	568
Table 459. Central Receive Status Register (CANRxSR - address 0x4004 0004) bit description. . . . .	527	Table 485: SSPn Clock Prescale Register (SSP0CPSR - address 0x4008 8010, SSP1CPSR - 0x4003 0010, SSP2CPSR - 0x400A C010) bit description . . . . .	568
Table 460. Central Miscellaneous Status Register (CANMSR - address 0x4004 0008) bit description . . . . .	527	Table 486: SSPn Interrupt Mask Set/Clear register (SSP0IMSC - address 0x4008 8014, SSP1IMSC - 0x4003 0014, SSP2IMSC - 0x400A C014) bit description . . . . .	568
Table 461. Acceptance filter modes and access control . . . . .	528	Table 487: SSPn Raw Interrupt Status register (SSP0RIS - address 0x4008 8018, SSP1RIS - 0x4003 0018, SSP2RIS - 0x400A C018) bit description . . . . .	569
Table 462. Section configuration register settings. . . . .	529	Table 488: SSPn Masked Interrupt Status register (SSP0MIS - address 0x4008 801C, SSP1MIS - 0x4003 001C, SSP2MIS - 0x400A C01C) bit description . . . . .	569
Table 463. Acceptance Filter Mode Register (AFMR - address 0x4003 C000) bit description . . . . .	531	Table 489: SSPn interrupt Clear Register (SSP0ICR - address 0x4008 8020, SSP1ICR - 0x4003 0020, SSP2ICR - 0x400A C020) bit description . . . . .	570
Table 464. Standard Frame Individual Start Address register (SFF_sa - address 0x4003 C004) bit description . . . . .	532	Table 490: SSPn DMA Control Register (SSP0DMACR - address 0x4008 8024, SSP1DMACR - 0x4003 0024, SSP2DMACR - 0x400A C024) bit description . . . . .	570
Table 465. Standard Frame Group Start Address register (SFF_GRP_sa - address 0x4003 C008) bit description . . . . .	533	Table 491. I <sup>2</sup> C Pin Description . . . . .	574
Table 466. Extended Frame Start Address register (EFF_sa - address 0x4003 C00C) bit description . . . . .	533	Table 492. I2C0CONSET and I2C1CONSET used to configure Master mode . . . . .	575
Table 467. Extended Frame Group Start Address register (EFF_GRP_sa - address 0x4003 C010) bit description . . . . .	533	Table 493. I2C0CONSET and I2C1CONSET used to configure Slave mode . . . . .	577
Table 468. End of AF Tables register (ENDofTable - address 0x4003 C014) bit description . . . . .	534	Table 494. I <sup>2</sup> C register map . . . . .	583
Table 469. LUT Error Address register (LUTerrAd - address 0x4003 C018) bit description . . . . .	534	Table 495. I <sup>2</sup> C Control Set register (I2CONSET: I <sup>2</sup> C0, I2C0CONSET - address 0x4001 C000, I <sup>2</sup> C1, I2C1CONSET - address 0x4005 C000, I <sup>2</sup> C2, I2C2CONSET - address 0x400A 0000) bit description . . . . .	585
Table 470. LUT Error register (LUTerr - address 0x4003 C01C) bit description . . . . .	534	Table 496. I <sup>2</sup> C Control Clear register (I2CONCLR: I <sup>2</sup> C0, I2C0CONCLR - 0x4001 C018; I <sup>2</sup> C1, I2C1CONCLR - 0x4005 C018; I <sup>2</sup> C2, I2C2CONCLR - 0x400A 0018) bit description . . . . .	586
Table 471. Global FullCAN Enable register (FCANIE - address 0x4003 C020) bit description . . . . .	535	Table 497. I <sup>2</sup> C Status register (I2STAT: I <sup>2</sup> C0, I2C0STAT - 0x4001 C004; I <sup>2</sup> C1, I2C1STAT - 0x4005 C004; I <sup>2</sup> C2, I2C2STAT - 0x400A 0004) bit description. . . . .	587
Table 472. FullCAN Interrupt and Capture register 0 (FCANIC0 - address 0x4003 C024) bit description . . . . .	535	Table 498. I <sup>2</sup> C Data register (I2DAT: I <sup>2</sup> C0, I2C0DAT - 0x4001 C008; I <sup>2</sup> C1, I2C1DAT - 0x4005 C008;	
Table 473. FullCAN Interrupt and Capture register 1 (FCANIC1 - address 0x4003 C028) bit description . . . . .	535		
Table 474. Format of automatically stored Rx messages . . . . .	538		
Table 475. FullCAN semaphore operation. . . . .	538		
Table 476. Example of Acceptance Filter Tables and ID index Values. . . . .	549		



Table 499. I <sup>2</sup> C Monitor mode control register (I2MMCTRL: I <sup>2</sup> C0, I2COMMCTRL - 0x4001 C01C; I <sup>2</sup> C1, I2C1MMCTRL- 0x4005 C01C; I <sup>2</sup> C2, I2C2MMCTRL- 0x400A 001C) bit description	587	Table 523: Interrupt Request Control register (I2SIRQ - address 0x400A 801C) bit description	623
Table 500. I <sup>2</sup> C Data buffer register (I2DATA_BUFFER: I <sup>2</sup> C0, I2CDATA_BUFFER - 0x4001 C02C; I <sup>2</sup> C1, I2C1DATA_BUFFER- 0x4005 C02C; I <sup>2</sup> C2, I2C2DATA_BUFFER- 0x400A 002C) bit description	589	Table 524: Transmit Clock Rate register (I2TXRATE - address 0x400A 8020) bit description	624
Table 501. I <sup>2</sup> C Slave Address registers (I2ADR0 to 3: I <sup>2</sup> C0, I2C0ADR[0, 1, 2, 3]- 0x4001 C0[0C, 20, 24, 28]; I <sup>2</sup> C1, I2C1ADR[0, 1, 2, 3] - address 0x4005 C0[0C, 20, 24, 28]; I <sup>2</sup> C2, I2C2ADR[0, 1, 2, 3] - address 0x400A 00[0C, 20, 24, 28]) bit description	590	Table 525: Receive Clock Rate register (I2SRXRATE - address 0x400A 8024) bit description	625
Table 502. I <sup>2</sup> C Mask registers (I2MASK0 to 3: I <sup>2</sup> C0, I2C0MASK[0, 1, 2, 3]- 0x4001 C0[30, 34, 38, 3C]; I <sup>2</sup> C1, I2C1MASK[0, 1, 2, 3] - address 0x4005 C0[30, 34, 38, 3C]; I <sup>2</sup> C2, I2C2MASK[0, 1, 2, 3] - address 0x400A 00[30, 34, 38, 3C]) bit description	590	Table 526: Transmit Clock Bit Rate register (I2TXBITRATE - address 0x400A 8028) bit description	625
Table 503. I <sup>2</sup> C SCL HIGH Duty Cycle register (I2SCLH: I <sup>2</sup> C0, I2C0SCLH - address 0x4001 C010; I <sup>2</sup> C1, I2C1SCLH - address 0x4005 C010; I <sup>2</sup> C2, I2C2SCLH - 0x400A 0010) bit description	591	Table 527: Receive Clock Rate Bit register (I2SRXBITRATE - address 0x400A 802C) bit description	625
Table 504. I <sup>2</sup> C SCL Low duty cycle register (I2SCLL: I <sup>2</sup> C0 - I2C0SCLL: 0x4001 C014; I <sup>2</sup> C1 - I2C1SCLL: 0x4005 C014; I <sup>2</sup> C2 - I2C2SCLL: 0x400A 0014) bit description	591	Table 528: Transmit Mode Control register (I2STXMODE - 0x400A 8030) bit description	626
Table 505. Example I <sup>2</sup> C clock rates	591	Table 529: Receive Mode Control register (I2SRXMODE - 0x400A 8034) bit description	626
Table 506. Abbreviations used to describe an I <sup>2</sup> C operation	593	Table 530: I <sup>2</sup> S transmit modes	628
Table 507. I2CONSET used to initialize Master Transmitter mode	594	Table 531: I <sup>2</sup> S receive modes	630
Table 508. I2CONSET used to initialize Slave Receiver mode	598	Table 532. Conditions for FIFO level comparison	632
Table 509. Master Transmitter mode	601	Table 533. DMA and interrupt request generation	632
Table 510. Master Receiver mode	602	Table 534. Status feedback in the I2SSTATE register	632
Table 511. Slave Receiver mode	603	Table 535. Timer/Counter pin description	637
Table 512. Slave Transmitter mode	605	Table 536. TIMER/COUNTER0-3 register map	638
Table 513. Miscellaneous States	606	Table 537. Interrupt Register (T[0/1/2/3]IR - addresses 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000) bit description	639
Table 514. Pin descriptions	619	Table 538. Timer Control Register (TCR, TIMERn: TnTCR - addresses 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004) bit description	640
Table 515. I <sup>2</sup> S register map	620	Table 539. Count Control Register (T[0/1/2/3]CTCR - addresses 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) bit description	640
Table 516: Digital Audio Output register (I2SDAO - address 0x400A 8000) bit description	621	Table 540. Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014) bit description	642
Table 517: Digital Audio Input register (I2SDAI - address 0x400A 8004) bit description	621	Table 541. Capture Control Register (T[0/1/2/3]CCR - addresses 0x4000 4028, 0x4000 8020, 0x4009 0028, 0x4009 4028) bit description	643
Table 518: Transmit FIFO register (I2STXFIFO - address 0x400A 8008) bit description	621	Table 542. External Match Register (T[0/1/2/3]EMR - addresses 0x4000 403C, 0x4000 803C, 0x4009 003C, 0x4009 403C) bit description	644
Table 519: Receive FIFO register (I2RXFIFO - address 0x400A 800C) bit description	622	Table 543. External Match Control	644
Table 520: Status Feedback register (I2SSTATE - address 0x400A 8010) bit description	622	Table 544. System Tick Timer register map	648
Table 521: DMA Configuration register 1 (I2SDMA1 - address 0x400A 8014) bit description	622	Table 545. System Timer Control and status register (STCTRL - 0xE000 E010) bit description	648
Table 522: DMA Configuration register 2 (I2SDMA2 - address 0x400A 8018) bit description	623	Table 546. System Timer Reload value register (STRELOAD - 0xE000 E014) bit description	649
		Table 547. System Timer Current value register (STCURR - 0xE000 E018) bit description	649
		Table 548. System Timer Calibration value register (STCALIB - 0xE000 E01C) bit description	650
		Table 549. Set and reset inputs for PWM Flip-Flops	655
		Table 550. Pin summary	656
		Table 551. PWM0 and PWM1 register map	657
		Table 552: PWM Interrupt Register (PWM0IR - 0x4001 4000 and PWM1IR - 0x4001 8000) bit description	658
		Table 553: PWM Timer Control Register (PWM0TCR - 0x4001 4004 PWM1TCR - 0x4001 8004) bit description	658
		Table 554: PWM Count control Register (PWMOCTCR -	

0x4001 4070 and PWM1CTCR - 0x4001 8070) bit description . . . . .	659	- 0x400B 8018, 0x400B 801C, 0x400B 8020) bit description . . . . .	679
Table 555: Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014) bit description . . . . .	660	Table 579: MCPWM Limit 0-2 registers (MCLIM0-2 - 0x400B 8024, 0x400B 8028, 0x400B 802C) bit description . . . . .	680
Table 556: PWM Capture Control Register (PWM0CCR - 0x4001 4028 and PWM1CCR - 0x4001 8028) bit description . . . . .	662	Table 580: MCPWM Match 0-2 registers (MCMAT0-2 - addresses 0x400B 8030, 0x400B 8034, 0x400B 8038) bit description . . . . .	681
Table 557: PWM Control Registers (PWMPCCR - 0x4001 404C and PWM1PCR - 0x4001 804C) bit description . . . . .	663	Table 581: MCPWM Dead-time register (MCDT - address 0x400B 803C) bit description . . . . .	682
Table 558: PWM Latch Enable Register (PWM0LER - 0x4001 4050 and PWM1LER - 0x4001 8050) bit description . . . . .	664	Table 582: MCPWM Commutation Pattern register (MCCP - address 0x400B 8040) bit description . . . . .	682
Table 559: Pin summary . . . . .	666	Table 583: MCPWM Capture read addresses (MCCAP0/1/2 - 0x400B 8044, 0x400B 8048, 0x400B 804C) bit description . . . . .	683
Table 560: Motor Control Pulse Width Modulator (MCPWM) register map . . . . .	669	Table 584: MCPWM Capture clear address (CAP_CLR - 0x400B 8074) bit description . . . . .	683
Table 561: MCPWM Control read address (MCCON - 0x400B 8000) bit description . . . . .	670	Table 585: Encoder states . . . . .	692
Table 562: MCPWM Control set address (MCCON_SET - 0x400B 8004) bit description . . . . .	671	Table 586: Encoder state transitions <sup>[1]</sup> . . . . .	692
Table 563: MCPWM Control clear address (MCCON_CLR - 0x400B 8008) bit description . . . . .	672	Table 587: Encoder direction . . . . .	693
Table 564: MCPWM Capture Control read address (MCCAPCON - 0x400B 800C) bit description . . . . .	673	Table 588: QEI pin description . . . . .	695
Table 565: MCPWM Capture Control set address (MCCAPCON_SET - 0x400B 8010) bit description . . . . .	674	Table 589: Register summary . . . . .	696
Table 566: MCPWM Capture control clear register (MCCAPCON_CLR - address 0x400B 8014) bit description . . . . .	674	Table 590: QEI Control register (QEICON - address 0x400B C000) bit description . . . . .	697
Table 567: Motor Control PWM interrupts . . . . .	675	Table 591: QEI Configuration register (QEICONF - address 0x400B C008) bit description . . . . .	697
Table 568: Interrupt sources bit allocation table . . . . .	675	Table 592: QEI Status register (QEISTAT - address 0x400B C004) bit description . . . . .	698
Table 569: MCPWM Interrupt Enable read address (MCINTEN - 0x400B 8050) bit description . . . . .	675	Table 593: QEI Position register (QEIPOS - address 0x400B C00C) bit description . . . . .	698
Table 570: PWM interrupt enable set register (MCINTEN_SET - address 0x400B 8054) bit description . . . . .	675	Table 594: QEI Maximum Position register (QEIMAXPOS - address 0x400B C010) bit description . . . . .	698
Table 571: PWM interrupt enable clear register (MCINTEN_CLR - address 0x400B 8058) bit description . . . . .	676	Table 595: QEI Position Compare register 0 (CMPOS0 - address 0x400B C014) bit description . . . . .	698
Table 572: MCPWM Interrupt Flags read address (MCINTF - 0x400B 8068) bit description . . . . .	676	Table 596: QEI Position Compare register 1 (CMPOS1 - address 0x400B C018) bit description . . . . .	699
Table 573: MCPWM Interrupt Flags set address (PWMINTF_SET - 0x400B 806C) bit description . . . . .	676	Table 597: QEI Position Compare register 2 (CMPOS2 - address 0x400B C01C) bit description . . . . .	699
Table 574: MCPWM Interrupt Flags clear address (PWMINTF_CLR - 0x400B 8070) bit description . . . . .	676	Table 598: QEI Index Count register (INXCNT - address 0x400B C020) bit description . . . . .	699
Table 575: MCPWM Count Control read address (MCCNTCON - 0x400B 805C) bit description . . . . .	677	Table 599: QEI Index Compare register 0 (INXCMPO - address 0x400B C024) bit description . . . . .	699
Table 576: MCPWM Count Control set address (MCCNTCON_SET - 0x400B 8060) bit description . . . . .	678	Table 600: QEI Index Compare register 1 (INXCMP1 - address 0x400B C04C) bit description . . . . .	699
Table 577: MCPWM Count Control clear address (MCCAPCON_CLR - 0x400B 8064) bit description . . . . .	678	Table 601: QEI Index Compare register 2 (INXCMP2 - address 0x400B C050) bit description . . . . .	700
Table 578: MCPWM Timer/Counter 0-2 registers (MCTC0-2		Table 602: QEI Timer Load register (QEILOAD - address 0x400B C028) bit description . . . . .	700
		Table 603: QEI Timer register (QEITIME - address 0x400B C02C) bit description . . . . .	700
		Table 604: QEI Velocity register (QEIVEL - address 0x400B C030) bit description . . . . .	700
		Table 605: QEI Velocity Capture register (QEICAP - address 0x400B C034) bit description . . . . .	700
		Table 606: QEI Velocity Compare register (VELCOMP - address 0x400B C038) bit description . . . . .	701
		Table 607: QEI Digital Filter ON PHA (FILTERPHA - address 0x400B C03C) bit description . . . . .	701

Table 608: QEI Digital Filter on PHB (FILTERPHB - address 0x400B C040) bit description . . . . .	701	Table 640: Watchdog Timer Window register (WDWINDOW - 0x4000 0018) bit description . . . . .	724
Table 609: QEI Digital Filter on INX (FILTERINX - address 0x400B C044) bit description . . . . .	701	Table 641: ADC pin description . . . . .	728
Table 610: QEI index acceptance Window (WINDOW - address 0x400B C048) bit description . . . . .	701	Table 642: ADC registers . . . . .	729
Table 611: QEI Interrupt Status register (QEIINTSTAT - address 0x400B CFE0) bit description . . . . .	702	Table 643: A/D Control Register (ADOCR - address 0x4003 4000) bit description . . . . .	730
Table 612: QEI Interrupt Set register (QEISET - address 0x400B CFEC) bit description . . . . .	703	Table 644: A/D Global Data Register (AD0GDR - address 0x4003 4004) bit description . . . . .	731
Table 613: QEI Interrupt Clear register (QEICLR - address 0x400B CFE8) bit description . . . . .	704	Table 645: A/D Interrupt Enable register (AD0INTEN - address 0x4003 400C) bit description . . . . .	731
Table 614: QEI Interrupt Enable register (QEIIE - address 0x400B CFE4) bit description . . . . .	705	Table 646: A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C) bit description . . . . .	732
Table 615: QEI Interrupt Enable Set register (QEIIES - address 0x400B CFDC) bit description . . . . .	706	Table 647: A/D Status register (AD0STAT - address 0x4003 4030) bit description . . . . .	733
Table 616: QEI Interrupt Enable Clear register (QEIEEC - address 0x400B CFD8) bit description . . . . .	707	Table 648: A/D Trim register (ADTRM - address 0x4003 4034) bit description . . . . .	733
Table 617: RTC pin description . . . . .	710	Table 649: D/A Pin Description . . . . .	736
Table 618: Real-Time Clock register map . . . . .	711	Table 650: DAC registers . . . . .	737
Table 619: Interrupt Location Register (ILR - address 0x4002 4000) bit description . . . . .	712	Table 651: D/A Converter Register (DACR - address 0x4008 C000) bit description . . . . .	737
Table 620: Clock Control Register (CCR - address 0x4002 4008) bit description . . . . .	712	Table 652: D/A Control register (DACCTRL - address 0x4008 C004) bit description . . . . .	738
Table 621: Counter Increment Interrupt Register (CIIR - address 0x4002 400C) bit description . . . . .	713	Table 653: D/A Converter Counter Value register (DACCNTVAL - address 0x4008 C008) bit description . . . . .	738
Table 622: Alarm Mask Register (AMR - address 0x4002 4010) bit description . . . . .	714	Table 654: Endian behavior . . . . .	743
Table 623: RTC Auxiliary control register (RTC_AUX - address 0x4002 405C) bit description . . . . .	714	Table 655: DMA connections . . . . .	746
Table 624: RTC Auxiliary Enable register (RTC_AUXEN - address 0x4002 4058) bit description . . . . .	714	Table 656: GPDMA register map . . . . .	747
Table 625: Consolidated Time register 0 (CTIME0 - address 0x4002 4014) bit description . . . . .	715	Table 657: DMA Interrupt Status register (DMACIntStat - 0x2008 0000) . . . . .	749
Table 626: Consolidated Time register 1 (CTIME1 - address 0x4002 4018) bit description . . . . .	715	Table 658: DMA Interrupt Terminal Count Request Status register (DMACIntTCStat - 0x2008 0004) . . . . .	749
Table 627: Consolidated Time register 2 (CTIME2 - address 0x4002 401C) bit description . . . . .	715	Table 659: DMA Interrupt Terminal Count Request Clear register (DMACIntTCClear - 0x2008 0008) . . . . .	749
Table 628: Time Counter relationships and values . . . . .	716	Table 660: DMA Interrupt Error Status register (DMACIntErrStat - 0x2008 000C) . . . . .	750
Table 629: Time Counter registers . . . . .	716	Table 661: DMA Interrupt Error Clear register (DMACIntErrClr - 0x2008 0010) . . . . .	750
Table 630: Calibration register (CALIBRATION - address 0x4002 4040) bit description . . . . .	717	Table 662: DMA Raw Interrupt Terminal Count Status register (DMACRawIntTCStat - 0x2008 0014) . . . . .	750
Table 631: General purpose registers 0 to 4 (GPREG0 to GPREG4 - addresses 0x4002 4044 to 0x4002 4054) bit description . . . . .	718	Table 663: DMA Raw Error Interrupt Status register (DMACRawIntErrStat - 0x2008 0018) . . . . .	751
Table 632: Alarm registers . . . . .	718	Table 664: DMA Enabled Channel register (DMACEnbldChns - 0x2008 001C) . . . . .	751
Table 633: Watchdog register map . . . . .	722	Table 665: DMA Software Burst Request register (DMACSoftBReq - 0x2008 0020) . . . . .	751
Table 634: Watchdog Mode register (WDMOD - address 0x4000 0000) bit description . . . . .	722	Table 666: DMA Software Single Request register (DMACSoftSReq - 0x2008 0024) . . . . .	752
Table 635: Watchdog operating modes selection . . . . .	723	Table 667: DMA Software Last Burst Request register (DMACSoftLBReq - 0x2008 0028) . . . . .	752
Table 636: Watchdog Timer Constant register (WDTC - address 0x4000 0004) bit description . . . . .	723	Table 668: DMA Software Last Single Request register (DMACSoftLSReq - 0x2008 002C) . . . . .	752
Table 637: Watchdog Feed register (WDFEED - address 0x4000 0008) bit description . . . . .	724	Table 669: DMA Configuration register (DMACConfig - 0x2008 0030) . . . . .	753
Table 638: Watchdog Timer Value register (WDTV - address 0x4000 000C) bit description . . . . .	724	Table 670: DMA Synchronization register (DMACSync - 0x2008 0034) . . . . .	753
Table 639: Watchdog Timer Warning Interrupt register (WDWINDOW - 0x4000 0014) bit description . . . . .	724	Table 671: DMA Request Select register (DMACReqSel -	

0x400F C1C4) . . . . .	754	frequency (in MHz). . . . .	795
Table 672. DMA Channel Source Address registers (DMACCxSrcAddr - 0x2008 01x0) . . . . .	755	Table 705. ISP Echo command . . . . .	796
Table 673. DMA Channel Destination Address registers (DMACCxDestAddr - 0x2008 01x4) . . . . .	756	Table 706. ISP Write to RAM command . . . . .	796
Table 674. DMA Channel Linked List Item registers (DMACCxLLI - 0x2008 01x8) . . . . .	756	Table 707. ISP Read Memory command . . . . .	797
Table 675. DMA channel control registers (DMACCxControl - 0x2008 01xC) . . . . .	757	Table 708. ISP Prepare sector(s) for write operation command . . . . .	797
Table 676. DMA Channel Configuration registers (DMACCxConfig - 0x2008 01x0) . . . . .	759	Table 709. ISP Copy command . . . . .	798
Table 677. Transfer type bits . . . . .	760	Table 710. ISP Go command . . . . .	798
Table 678. DMA request signal usage . . . . .	763	Table 711. ISP Erase sector command . . . . .	799
Table 679. Register overview: CRC engine. . . . .	771	Table 712. ISP Blank check sector command . . . . .	799
Table 680. CRC mode register (CRC_MODE - 0x2009 0000) bit description . . . . .	771	Table 713. ISP Read Part Identification command . . . . .	799
Table 681. CRC seed register (CRC_SEED - 0x2009 0004) bit description . . . . .	771	Table 714. LPC178x/177x part identification numbers . . . . .	800
Table 682. CRC checksum register (CRC_SUM - 0x2009 0008) bit description . . . . .	771	Table 715. ISP Read Boot Code version number command 800	800
Table 683. CRC data register (CRC_DATA - 0x2009 0008) bit description . . . . .	772	Table 716. ISP Read device serial number command . . . . .	800
Table 684. Register overview . . . . .	775	Table 717. ISP Compare command . . . . .	800
Table 685. EEPROM command register (EECMD - address 0x0020 0080) bit description . . . . .	776	Table 718. ISP Return Codes Summary. . . . .	801
Table 686. EEPROM address register (EEADDR - address 0x0020 0084) address description . . . . .	776	Table 719. IAP Command Summary . . . . .	803
Table 687. EEPROM write data register (EEWDATA - address 0x0020 0088) bit description. . . . .	777	Table 720. IAP Prepare sector(s) for write operation command . . . . .	804
Table 688. EEPROM read data register (EERDATA - address 0x0020 008C) bit description . . . . .	777	Table 721. IAP Copy RAM to Flash command . . . . .	804
Table 689. EEPROM wait state register (EEESTATE - address 0x0020 0090) bit description. . . . .	778	Table 722. IAP Erase Sector(s) command . . . . .	805
Table 690. EEPROM clock divider register (EECLKDIV - address 0x0020 0094) bit description. . . . .	778	Table 723. IAP Blank check sector(s) command . . . . .	805
Table 691. EEPROM power down/DCM register (EEPWRDWN - address 0x0020 0098) bit description . . . . .	778	Table 724. IAP Read part identification number command. . 805	805
Table 692. Interrupt enable register (EEINTEN - address 0x0020 0FE4) bit description . . . . .	779	Table 725. IAP Read Boot Code version number command 806	806
Table 693. Interrupt enable clear register (EEINTENCLR - address 0x0020 0FD8) bit description . . . . .	779	Table 726. IAP Read device serial number command . . . . .	806
Table 694. Interrupt enable set register (EEINTENSET - address 0x0020 0FDC) bit description . . . . .	780	Table 727. IAP Compare command . . . . .	806
Table 695. Interrupt status register (EEINTSTAT - address 0x0020 0FE0) bit description . . . . .	780	Table 728. Re-invoke ISP . . . . .	807
Table 696. Interrupt status clear register (EEINTSTATCLR - address 0x0020 0FE8) bit description . . . . .	781	Table 729. IAP Status Codes Summary . . . . .	807
Table 697. Interrupt status set register (EEINTSTATSET - address 0x0020 0FEC). . . . .	781	Table 730. Register overview: FMC (base address 0x0020 0000) . . . . .	808
Table 698. Sectors in a LPC178x/177x device . . . . .	791	Table 731. Flash Module Signature Start register (FMSSTART - 0x0020 0020) bit description . . . . .	809
Table 699. Code Read Protection options. . . . .	792	Table 732. Flash Module Signature Stop register (FMSSTOP - 0x0020 0024) bit description . . . . .	809
Table 700. Code Read Protection hardware/software interaction . . . . .	793	Table 733. FMSW0 register bit description (FMSW0, address: 0x0020 002C) . . . . .	809
Table 701. ISP command summary. . . . .	794	Table 734. FMSW1 register bit description (FMSW1, address: 0x0020 0030) . . . . .	809
Table 702. ISP Unlock command . . . . .	794	Table 735. FMSW2 register bit description (FMSW2, address: 0x0020 0034) . . . . .	810
Table 703. ISP Set Baud Rate command . . . . .	795	Table 736. FMSW3 register bit description (FMSW3, address: 0x0020 0038) . . . . .	810
Table 704. Examples of supported ISP baud rates and PCLK frequency (in MHz). . . . .	795	Table 737. Flash module Status register (FMSTAT - 0x0020 0FE0) bit description . . . . .	810
		Table 738. Flash Module Status Clear register (FMSTATCLR - 0x0020 0FE8) bit description . . . . .	810
		Table 739. JTAG pin description . . . . .	813
		Table 740. Serial Wire Debug pin description. . . . .	813
		Table 741. Parallel Trace pin description . . . . .	813
		Table 742. Memory Mapping Control register (MEMMAP - 0x400F C040) bit description. . . . .	816
		Table 743. Cortex-M3 instructions . . . . .	820
		Table 744. CMSIS intrinsic functions to generate some Cortex-M3 instructions . . . . .	823
		Table 745. CMSIS intrinsic functions to access the special	

registers . . . . .	823	Table 799. MMFSR bit assignments . . . . .	950
Table 746. Condition code suffixes . . . . .	830	Table 800. BFSR bit assignments . . . . .	951
Table 747. Memory access instructions . . . . .	833	Table 801. UFSR bit assignments . . . . .	953
Table 748. Offset ranges . . . . .	836	Table 802. HFSR bit assignments . . . . .	954
Table 749. Offset ranges . . . . .	842	Table 803. MMFAR bit assignments . . . . .	954
Table 750. Data processing instructions . . . . .	850	Table 804. BFAR bit assignments . . . . .	955
Table 751. Multiply and divide instructions . . . . .	865	Table 805. System timer registers summary . . . . .	956
Table 752. Packing and unpacking instructions . . . . .	873	Table 806. SysTick CTRL register bit assignments . . . . .	956
Table 753. Branch and control instructions . . . . .	878	Table 807. LOAD register bit assignments . . . . .	957
Table 754. Branch ranges . . . . .	879	Table 808. VAL register bit assignments . . . . .	957
Table 755. Miscellaneous instructions . . . . .	887	Table 809. CALIB register bit assignments . . . . .	957
Table 756. Summary of processor mode, execution privilege level, and stack use options . . . . .	901	Table 810. Memory attributes summary . . . . .	959
Table 757. Core register set summary . . . . .	901	Table 811. MPU registers summary . . . . .	960
Table 758. PSR register combinations . . . . .	903	Table 812. TYPE register bit assignments . . . . .	960
Table 759. APSR bit assignments . . . . .	904	Table 813. MPU CTRL register bit assignments . . . . .	961
Table 760. IPSR bit assignments . . . . .	905	Table 814. RNR bit assignments . . . . .	962
Table 761. EPSR bit assignments . . . . .	905	Table 815. RBAR bit assignments . . . . .	963
Table 762. PRIMASK register bit assignments . . . . .	906	Table 816. RASR bit assignments . . . . .	964
Table 763. FAULTMASK register bit assignments . . . . .	906	Table 817. Example SIZE field values . . . . .	964
Table 764. BASEPRI register bit assignments . . . . .	907	Table 818. TEX, C, B, and S encoding . . . . .	965
Table 765. CONTROL register bit assignments . . . . .	907	Table 819. Cache policy for memory attribute encoding . . . . .	965
Table 766. Memory access behavior . . . . .	912	Table 820. AP encoding . . . . .	966
Table 767. SRAM memory bit-banding regions . . . . .	914	Table 821. Memory region attributes for a microcontroller . . . . .	969
Table 768. Peripheral memory bit-banding regions . . . . .	914	Table 822. Abbreviations . . . . .	974
Table 769. C compiler intrinsic functions for exclusive access instructions . . . . .	917		
Table 770. Properties of the different exception types . . . . .	919		
Table 771. Exception return behavior . . . . .	924		
Table 772. Faults . . . . .	926		
Table 773. Fault status and fault address registers . . . . .	927		
Table 774. Core peripheral register regions . . . . .	932		
Table 775. NVIC register summary . . . . .	933		
Table 776. Mapping of interrupts to the interrupt variables . . . . .	934		
Table 777. ISER bit assignments . . . . .	934		
Table 778. ICER bit assignments . . . . .	935		
Table 779. ISPR bit assignments . . . . .	935		
Table 780. ICPR bit assignments . . . . .	936		
Table 781. IABR bit assignments . . . . .	936		
Table 782. IPR bit assignments . . . . .	937		
Table 783. STIR bit assignments . . . . .	937		
Table 784. CMSIS functions for NVIC control . . . . .	939		
Table 785. Summary of the system control block registers . . . . .	940		
Table 786. ACTLR bit assignments . . . . .	941		
Table 787. CPUID register bit assignments . . . . .	941		
Table 788. ICSR bit assignments . . . . .	942		
Table 789. VTOR bit assignments . . . . .	944		
Table 790. AIRCR bit assignments . . . . .	944		
Table 791. Priority grouping . . . . .	945		
Table 792. SCR bit assignments . . . . .	946		
Table 793. CCR bit assignments . . . . .	947		
Table 794. System fault handler priority fields . . . . .	948		
Table 795. SHPR1 register bit assignments . . . . .	948		
Table 796. SHPR2 register bit assignments . . . . .	948		
Table 797. SHPR3 register bit assignments . . . . .	948		
Table 798. SHCSR bit assignments . . . . .	949		

39.4 Figures

Fig 1.	LPC178x/177x simplified block diagram	10	hosts	380	
Fig 2.	LPC178x/177x block diagram, CPU and buses	13	Fig 50.	USB device port configuration: port U1 host and port U2 device	381
Fig 3.	LPC1788 system memory map	16	Fig 51.	Port selection for PORT_FUNC bit 0 = 0 and PORT_FUNC bit 1 = 0	385
Fig 4.	Reset block diagram including the wake-up timer	22	Fig 52.	USB OTG interrupt handling	391
Fig 5.	Example of start-up after reset	23	Fig 53.	USB OTG controller with software stack	393
Fig 6.	External interrupt logic	28	Fig 54.	Hardware support for B-device switching from peripheral state to host state	394
Fig 7.	Clock generation for the LPC178x/177x	35	Fig 55.	State transitions implemented in software during B-device switching from peripheral to host	395
Fig 8.	Oscillator modes and models: a) slave mode of operation, b) oscillation mode of operation, c) external crystal model used for C <sub>X1/X2</sub> evaluation	38	Fig 56.	Hardware support for A-device switching from host state to peripheral state	397
Fig 9.	PLL1 block diagram	42	Fig 57.	State transitions implemented in software during A-device switching from host to peripheral	398
Fig 10.	CLKOUT selection	61	Fig 58.	Clocking and power control	402
Fig 11.	Simplified block diagram of the flash accelerator showing potential bus connections	63	Fig 59.	Secure digital memory card connection	406
Fig 12.	I/O configurations	108	Fig 60.	Multimedia card system	407
Fig 13.	EMC block diagram	155	Fig 61.	SD card interface	408
Fig 14.	EMC programmable delays	183	Fig 62.	Command path state machine	409
Fig 15.	EMC delay calibration	184	Fig 63.	Command transfer	410
Fig 16.	32 bit bank external memory interfaces (bits MW = 10)	186	Fig 64.	Data path state machine	412
Fig 17.	16 bit bank external memory interfaces (bits MW = 01)	186	Fig 65.	Pending command start	414
Fig 18.	8 bit bank external memory interface (bits MW = 00)	187	Fig 66.	UART1 block diagram	429
Fig 19.	Typical memory configuration diagram	188	Fig 67.	Auto-RTS Functional Timing	439
Fig 20.	Ethernet block diagram	191	Fig 68.	Auto-CTS Functional Timing	440
Fig 21.	Ethernet packet fields	193	Fig 69.	Auto-baud a) mode 0 and b) mode 1 waveform	445
Fig 22.	Receive descriptor memory layout	220	Fig 70.	Algorithm for setting UART dividers	447
Fig 23.	Transmit descriptor memory layout	223	Fig 71.	UART0, 2, and 3 block diagram	454
Fig 24.	Transmit example memory and registers	236	Fig 72.	Auto-baud a) mode 0 and b) mode 1 waveform	466
Fig 25.	Receive Example Memory and Registers	242	Fig 73.	Algorithm for setting UART dividers	468
Fig 26.	Transmit Flow Control	247	Fig 74.	UART 4 block diagram	475
Fig 27.	Receive filter block diagram	249	Fig 75.	Auto-baud a) mode 0 and b) mode 1 waveform	486
Fig 28.	Receive Active/Inactive state machine	253	Fig 76.	Algorithm for setting UART dividers	489
Fig 29.	Transmit Active/Inactive state machine	254	Fig 77.	CAN controller block diagram	500
Fig 30.	LCD controller block diagram	270	Fig 78.	Transmit buffer layout for standard and extended frame format configurations	501
Fig 31.	Cursor movement	278	Fig 79.	Receive buffer layout for standard and extended frame format configurations	502
Fig 32.	Cursor clipping	279	Fig 80.	Global Self-Test (high-speed CAN Bus example)	503
Fig 33.	Cursor image format	280	Fig 81.	Local self test (high-speed CAN Bus example)	503
Fig 34.	Power-up and power-down sequences	285	Fig 82.	Entry in FullCAN and individual standard identifier tables	530
Fig 35.	Horizontal timing for STN displays	302	Fig 83.	Entry in standard identifier range table	530
Fig 36.	Vertical timing for STN displays	303	Fig 84.	Entry in either extended identifier table	530
Fig 37.	Horizontal timing for TFT displays	303	Fig 85.	ID Look-up table example explaining the search algorithm	536
Fig 38.	Vertical timing for TFT displays	304	Fig 86.	Semaphore procedure for reading an auto-stored message	540
Fig 39.	USB device controller block diagram	311	Fig 87.	FullCAN section example of the ID look-up table	542
Fig 40.	USB MaxPacketSize register array indexing	328	Fig 88.	FullCAN message object layout	542
Fig 41.	Interrupt event handling	340	Fig 89.	Normal case, no messages lost	544
Fig 42.	UDCA Head register and DMA Descriptors	355	Fig 90.	Message lost	544
Fig 43.	Isochronous OUT endpoint operation example	362	Fig 91.	Message gets overwritten	545
Fig 44.	Data transfer in ATLE mode	363			
Fig 45.	USB Host controller block diagram	371			
Fig 46.	USB OTG controller block diagram	376			
Fig 47.	USB OTG port configuration: port U1 OTG dual-role device, port U2 host	378			
Fig 48.	USB OTG port configuration: VP_VM mode	379			
Fig 49.	USB host port configuration: port U1 and U2 as				

Fig 92. Message overwritten indicated by semaphore bits and message lost. . . . .	546	output . . . . .	631
Fig 93. Message overwritten indicated by message lost	547	Fig 130. Receiver master mode sharing the transmitter reference clock . . . . .	631
Fig 94. Clearing message lost . . . . .	548	Fig 131. 4-wire receiver master mode sharing the transmitter bit clock and WS . . . . .	631
Fig 95. Detailed example of acceptance filter tables and ID index values. . . . .	550	Fig 132. Typical receiver slave mode . . . . .	631
Fig 96. ID Look-up table configuration example (no FullICAN) . . . . .	552	Fig 133. Receiver slave mode sharing the transmitter reference clock . . . . .	631
Fig 97. ID Look-up table configuration example (FullICAN activated and enabled) . . . . .	554	Fig 134. 4-wire receiver slave mode sharing the transmitter bit clock and WS . . . . .	632
Fig 98. Texas Instruments Synchronous Serial Frame Format: a) Single and b) Continuous/back-to-back Two Frames Transfer. . . . .	558	Fig 135. FIFO contents for various I <sup>2</sup> S modes . . . . .	633
Fig 99. SPI frame format with CPOL=0 and CPHA=0 (a) Single and b) Continuous Transfer) . . . . .	559	Fig 136. Timer block diagram . . . . .	636
Fig 100. SPI frame format with CPOL=0 and CPHA=1 . . . . .	560	Fig 137. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled. . . . .	646
Fig 101. SPI frame format with CPOL = 1 and CPHA = 0 (a) Single and b) Continuous Transfer) . . . . .	561	Fig 138. A timer Cycle in Which PR=2, MRx=6, and both interrupt and stop on match are enabled . . . . .	646
Fig 102. SPI Frame Format with CPOL = 1 and CPHA = 1. . . . .	562	Fig 139. System Tick Timer block diagram . . . . .	648
Fig 103. Microwire frame format (single transfer) . . . . .	563	Fig 140. PWM block diagram . . . . .	654
Fig 104. Microwire frame format (continuous transfers) . . . . .	564	Fig 141. Sample PWM waveforms . . . . .	655
Fig 105. Microwire frame format setup and hold details . . . . .	564	Fig 142. MCPWM Block Diagram . . . . .	667
Fig 106. I <sup>2</sup> C-bus configuration. . . . .	573	Fig 143. Edge-aligned PWM waveform without dead time, POLA = 0. . . . .	684
Fig 107. Format in the Master Transmitter mode . . . . .	575	Fig 144. Center-aligned PWM waveform without dead time, POLA = 0. . . . .	685
Fig 108. Format of Master Receiver mode . . . . .	576	Fig 145. Edge-aligned PWM waveform with dead time, POLA = 0. . . . .	685
Fig 109. A Master Receiver switches to Master Transmitter after sending repeated START . . . . .	576	Fig 146. Center-aligned waveform with dead time, POLA = 0 686	686
Fig 110. Format of Slave Receiver mode . . . . .	577	Fig 147. Three-phase DC mode sample waveforms . . . . .	688
Fig 111. Format of Slave Transmitter mode . . . . .	578	Fig 148. Three-phase AC mode sample waveforms, edge aligned PWM mode. . . . .	689
Fig 112. I <sup>2</sup> C serial interface block diagram . . . . .	579	Fig 149. Encoder interface block diagram . . . . .	691
Fig 113. Arbitration procedure . . . . .	581	Fig 150. Quadrature Encoder Basic Operation. . . . .	693
Fig 114. Serial clock synchronization . . . . .	581	Fig 151. RTC domain conceptual diagram . . . . .	709
Fig 115. Format and states in the Master Transmitter mode. . . . .	595	Fig 152. RTC functional block diagram. . . . .	709
Fig 116. Format and states in the Master Receiver mode. . . . .	597	Fig 153. Watchdog Timer block diagram . . . . .	721
Fig 117. Format and states in the Slave Receiver mode. . . . .	599	Fig 154. Early Watchdog Feed with Windowed Mode Enabled. . . . .	725
Fig 118. Format and states in the Slave Transmitter mode. . . . .	600	Fig 155. Correct Watchdog Feed with Windowed Mode Enabled. . . . .	725
Fig 119. Simultaneous repeated START conditions from two masters . . . . .	608	Fig 156. Watchdog Warning Interrupt . . . . .	726
Fig 120. Forced access to a busy I <sup>2</sup> C-bus . . . . .	608	Fig 157. ADC block diagram . . . . .	728
Fig 121. Recovering from a bus obstruction caused by a LOW level on SDA. . . . .	608	Fig 158. DAC control with DMA interrupt and timer . . . . .	736
Fig 122. Simple I <sup>2</sup> S configurations and bus timing . . . . .	619	Fig 159. DMA controller block diagram. . . . .	741
Fig 123. Typical transmitter master mode, with or without MCLK output . . . . .	629	Fig 160. LLI example . . . . .	767
Fig 124. Transmitter master mode sharing the receiver reference clock . . . . .	629	Fig 161. CRC block diagram. . . . .	770
Fig 125. 4-wire transmitter master mode sharing the receiver bit clock and WS . . . . .	629	Fig 162. Starting a write operation . . . . .	783
Fig 126. Typical transmitter slave mode . . . . .	629	Fig 163. (16-bit) write operations with post-incrementing of address . . . . .	783
Fig 127. Transmitter slave mode sharing the receiver reference clock . . . . .	629	Fig 164. Programming a page into memory . . . . .	784
Fig 128. 4-wire transmitter slave mode sharing the receiver bit clock and WS . . . . .	630	Fig 165. Starting a read operation (32-bit read from address A). . . . .	785
Fig 129. Typical receiver master mode, with or without MCLK		Fig 166. Map of lower memory . . . . .	787





## 39.5 Contents

### Chapter 1: Introductory information

<b>1.1</b>	<b>Introduction</b> .....	<b>3</b>	<b>1.7</b>	<b>ARM Cortex-M3 processor</b> .....	<b>11</b>
<b>1.2</b>	<b>Features</b> .....	<b>4</b>	<b>1.7.1</b>	Cortex-M3 Configuration Options .....	11
<b>1.3</b>	<b>Applications</b> .....	<b>7</b>		System options: .....	11
<b>1.4</b>	<b>Ordering information</b> .....	<b>8</b>		Debug related options: .....	11
1.4.1	Part options summary .....	9	<b>1.8</b>	<b>On-chip flash memory system</b> .....	<b>12</b>
<b>5</b>	<b>Simplified block diagram</b> .....	<b>10</b>	<b>1.9</b>	<b>On-chip Static RAM</b> .....	<b>12</b>
<b>1.6</b>	<b>Architectural overview</b> .....	<b>11</b>	<b>1.10</b>	<b>On-chip EEPROM</b> .....	<b>12</b>
			<b>1.11</b>	<b>Detailed block diagram</b> .....	<b>13</b>

### Chapter 2: LPC178x/7x Memory map

<b>2.1</b>	<b>Memory map and peripheral addressing</b> . . . .	<b>14</b>	<b>2.4</b>	<b>Memory re-mapping</b> .....	<b>18</b>
<b>2.2</b>	<b>Memory maps</b> .....	<b>15</b>		Boot ROM re-mapping .....	19
<b>2.3</b>	<b>On-chip peripherals</b> .....	<b>17</b>	<b>2.5</b>	<b>AHB arbitration</b> .....	<b>19</b>
2.3.1	AHB peripherals .....	17	2.5.1	Matrix Arbitration register (Matrix_Arb - 0x400F C188) .....	19
2.3.2	APB peripheral addresses .....	17			

### Chapter 3: LPC178x/7x System control

<b>3.1</b>	<b>Introduction</b> .....	<b>20</b>	<b>3.6</b>	<b>Brown-out detection</b> .....	<b>27</b>
<b>3.2</b>	<b>Pin description</b> .....	<b>20</b>	<b>3.7</b>	<b>External interrupt inputs</b> .....	<b>28</b>
<b>3.3</b>	<b>Register description</b> .....	<b>21</b>	3.7.1	Register description .....	29
<b>3.4</b>	<b>Chip reset</b> .....	<b>22</b>	3.7.2	External Interrupt flag register (EXTINT - 0x400F C140) .....	29
3.4.1	Reset Source Identification Register (RSID - 0x400F C180) .....	24	3.7.3	External Interrupt Mode register (EXTMODE - 0x400F C148) .....	30
<b>3.5</b>	<b>Peripheral reset control</b> .....	<b>25</b>	3.7.4	External Interrupt Polarity register (EXTPOLAR - 0x400F C14C) .....	31
3.5.1	Reset control register 0 (RSTCON0 - 0x400F C1CC) .....	25	<b>3.8</b>	<b>Other system controls and status flags</b> . . . .	<b>32</b>
3.5.2	Reset control register 1 (RSTCON1 - 0x400F C1D0) .....	26	3.8.1	System Controls and Status register (SCS - 0x400F C1A0) .....	32

### Chapter 4: LPC178x/7x Clocking and power control

<b>4.1</b>	<b>Summary of clocking and power control functions</b> .....	<b>34</b>	4.5.4	PLL Configuration registers (PLL0CFG - 0x400F C084, PLL1CFG - 0x400F C0A4) . . . .	43
<b>4.2</b>	<b>Register description</b> .....	<b>36</b>	4.5.5	PLL Status registers (PLL0STAT - 0x400F C088, PLL1STAT - 0x400F C0A8) .....	43
<b>4.3</b>	<b>Oscillators</b> .....	<b>37</b>	4.5.6	PLL Interrupts: PLOCK0 and PLOCK1 .....	44
4.3.1	Internal RC oscillator .....	37	4.5.7	PLL Feed registers (PLL0FEED - 0x400F C08C, PLL1FEED - 0x400F C0AC) .....	44
4.3.2	Main oscillator .....	37	4.5.8	PLLs and Power-down mode .....	45
4.3.2.1	Main oscillator startup .....	39	4.5.9	PLL frequency calculation .....	46
4.3.3	RTC oscillator .....	39	4.5.10	Procedure for determining PLL settings. . . . .	46
4.3.4	Watchdog oscillator .....	39	4.5.11	PLL configuration sequence .....	48
<b>4.4</b>	<b>Clock source selection multiplexer</b> .....	<b>40</b>		To set up a PLL and switch clocks to its output: . . . . .	48
4.4.1	Clock Source Selection register (CLKSRCSEL - 0x400F C10C) .....	40		To switch clocks away from a PLL output: . . . . .	48
<b>4.5</b>	<b>PLL0 and PLL1 (Phase Locked Loops)</b> . . . . .	<b>41</b>	4.5.12	PLL configuration examples .....	48
4.5.1	PLL and startup/boot code interaction .....	41	<b>4.6</b>	<b>Clock selection and division</b> .....	<b>50</b>
4.5.2	PLL register description .....	42	4.6.1	CPU Clock Selection register (CCLKSEL - 0x400F C104) .....	50
4.5.3	PLL Control registers (PLL0CON - 0x400F C080, PLL1CON - 0x400F C0A0) .....	43			

4.6.2	USB Clock Selection register (USBCLKSEL - 0x400F C108) . . . . .	51	4.7.7	Power Mode Control register (PCON - 0x400F C0C0) . . . . .	56
4.6.3	EMC Clock Selection register (EMCCLKSEL - 0x400F C100) . . . . .	51	4.7.7.1	Encoding of Reduced Power Modes . . . . .	56
4.6.4	Peripheral Clock Selection register (PCLKSEL - 0x400F C1A8) . . . . .	52	4.7.8	Wake-up from Reduced Power Modes . . . . .	57
<b>4.7</b>	<b>Power control . . . . .</b>	<b>53</b>	4.7.9	Power Control for Peripherals register (PCONP - 0x400F C0C4) . . . . .	57
4.7.1	Sleep mode . . . . .	53	4.7.10	Power control usage notes . . . . .	59
4.7.2	Deep Sleep mode . . . . .	53	4.7.11	Power domains . . . . .	59
4.7.3	Power-down mode . . . . .	54	<b>4.8</b>	<b>Wake-up timer . . . . .</b>	<b>60</b>
4.7.4	Deep Power-down mode . . . . .	55	<b>4.9</b>	<b>External clock output pin . . . . .</b>	<b>61</b>
4.7.5	Peripheral power control . . . . .	55	4.9.1	Clock Output Configuration register (CLKOUTCFG - 0x400F C1C8) . . . . .	61
4.7.6	Register description . . . . .	55			

**Chapter 5: LPC178x/7x Flash accelerator**

<b>5.1</b>	<b>Introduction . . . . .</b>	<b>63</b>	<b>5.3</b>	<b>Register description . . . . .</b>	<b>64</b>
<b>5.2</b>	<b>Flash accelerator blocks . . . . .</b>	<b>63</b>	<b>5.4</b>	<b>Flash Accelerator Configuration register (FLASHCFG - 0x400F C000) . . . . .</b>	<b>65</b>
5.2.1	Flash memory bank . . . . .	63	<b>5.5</b>	<b>Operation . . . . .</b>	<b>65</b>
5.2.2	Flash programming Issues . . . . .	64			

**Chapter 6: LPC178x/7x Nested Vectored Interrupt Controller (NVIC)**

<b>6.1</b>	<b>Features . . . . .</b>	<b>67</b>	6.5.10	Interrupt Active Bit Register 1 (IABR1 - 0xE000 E304) . . . . .	81
<b>6.2</b>	<b>Description . . . . .</b>	<b>67</b>	6.5.11	Interrupt Priority Register 0 (IPR0 - 0xE000 E400) . . . . .	82
<b>6.3</b>	<b>Interrupt sources . . . . .</b>	<b>67</b>	6.5.12	Interrupt Priority Register 1 (IPR1 - 0xE000 E404) . . . . .	82
<b>6.4</b>	<b>Vector table remapping . . . . .</b>	<b>70</b>	6.5.13	Interrupt Priority Register 2 (IPR2 - 0xE000 E408) . . . . .	82
	Examples: . . . . .	70	6.5.14	Interrupt Priority Register 3 (IPR3 - 0xE000 E40C) . . . . .	83
<b>6.5</b>	<b>Register description . . . . .</b>	<b>71</b>	6.5.15	Interrupt Priority Register 4 (IPR4 - 0xE000 E410) . . . . .	83
6.5.1	Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100) . . . . .	72	6.5.16	Interrupt Priority Register 5 (IPR5 - 0xE000 E414) . . . . .	83
6.5.2	Interrupt Set-Enable Register 1 register (ISER1 - 0xE000 E104) . . . . .	73	6.5.17	Interrupt Priority Register 6 (IPR6 - 0xE000 E418) . . . . .	84
6.5.3	Interrupt Clear-Enable Register 0 (ICER0 - 0xE000 E180) . . . . .	74	6.5.18	Interrupt Priority Register 7 (IPR7 - 0xE000 E41C) . . . . .	84
5.4	Interrupt Clear-Enable Register 1 register (ICER1 - 0xE000 E184) . . . . .	75	6.5.19	Interrupt Priority Register 8 (IPR8 - 0xE000 E420) . . . . .	84
6.5.5	Interrupt Set-Pending Register 0 register (ISPR0 - 0xE000 E200) . . . . .	76	6.5.20	Interrupt Priority Register 9 (IPR9 - 0xE000 E424) . . . . .	85
6.5.6	Interrupt Set-Pending Register 1 register (ISPR1 - 0xE000 E204) . . . . .	77	6.5.21	Interrupt Priority Register 10 (IPR10 - 0xE000 E428) . . . . .	85
6.5.7	Interrupt Clear-Pending Register 0 register (ICPR0 - 0xE000 E280) . . . . .	78	6.5.22	Software Trigger Interrupt Register (STIR - 0xE000 EF00) . . . . .	85
6.5.8	Interrupt Clear-Pending Register 1 register (ICPR1 - 0xE000 E284) . . . . .	79			
6.5.9	Interrupt Active Bit Register 0 (IABR0 - 0xE000 E300) . . . . .	80			

**Chapter 7: LPC178x/7x Pin configuration**

<b>7.1</b>	<b>LPC178x/177x pin configuration . . . . .</b>	<b>86</b>	<b>7.2</b>	<b>Boot control . . . . .</b>	<b>106</b>
------------	---	-----------	------------	-------------------------------	------------

**Chapter 8: LPC178x/7x I/O configuration**

<b>8.1</b>	<b>How to read this chapter . . . . .</b>	<b>107</b>		Multiple connections . . . . .	108
<b>8.2</b>	<b>Description . . . . .</b>	<b>107</b>	8.3.1	Pin function . . . . .	108
<b>8.3</b>	<b>IOCON registers . . . . .</b>	<b>107</b>	8.3.2	Pin mode . . . . .	109

8.3.3	Hysteresis . . . . .	109	8.4.1.1	Type D IOCON registers (applies to most GPIO port pins) . . . . .	118
8.3.4	Input Inversion . . . . .	109	8.4.1.2	Type A IOCON registers (applies to pins that include an analog function) . . . . .	124
8.3.5	Analog/digital mode . . . . .	109	8.4.1.3	Type U IOCON registers (applies to pins that include a USB D+ or D- function) . . . . .	126
8.3.6	Input filter . . . . .	109	8.4.1.4	Type I IOCON registers (applies to pins that include a specialized I <sup>2</sup> C function) . . . . .	127
8.3.7	Output slew rate . . . . .	109	8.4.1.5	Type W IOCON registers (these pins are otherwise the same as Type D, but include a selectable input glitch filter, and default to pull-down/pull-up disabled) . . . . .	128
8.3.8	I <sup>2</sup> C modes . . . . .	110			
8.3.9	Open-Drain Mode . . . . .	110			
8.3.10	DAC enable . . . . .	110			
<b>8.4</b>	<b>Register description . . . . .</b>	<b>111</b>			
8.4.1	I/O configuration register contents (IOCON) . . . . .	117			

**Chapter 9: LPC178x/7x GPIO**

<b>9.1</b>	<b>Basic configuration . . . . .</b>	<b>129</b>	9.5.6.2	GPIO Interrupt Enable for port 0 Rising Edge (IO0IntEnR - 0x4002 8090) . . . . .	141
<b>9.2</b>	<b>Features . . . . .</b>	<b>129</b>	9.5.6.3	GPIO Interrupt Enable for port 2 Rising Edge (IO2IntEnR - 0x4002 80B0) . . . . .	142
9.2.1	Digital I/O ports . . . . .	129	9.5.6.4	GPIO Interrupt Enable for port 0 Falling Edge (IO0IntEnF - 0x4002 8094) . . . . .	143
9.2.2	Interrupt generating digital ports . . . . .	129	9.5.6.5	GPIO Interrupt Enable for port 2 Falling Edge (IO2IntEnF - 0x4002 80B4) . . . . .	144
<b>9.3</b>	<b>Applications . . . . .</b>	<b>130</b>	9.5.6.6	GPIO Interrupt Status for port 0 Rising Edge Interrupt (IO0IntStatR - 0x4002 8084) . . . . .	145
<b>9.4</b>	<b>Pin description . . . . .</b>	<b>130</b>	9.5.6.7	GPIO Interrupt Status for port 2 Rising Edge Interrupt (IO2IntStatR - 0x4002 80A4) . . . . .	146
<b>9.5</b>	<b>Register description . . . . .</b>	<b>131</b>	9.5.6.8	GPIO Interrupt Status for port 0 Falling Edge Interrupt (IO0IntStatF - 0x4002 8088) . . . . .	147
9.5.1	GPIO port Direction register FIOxDIR (FIO0DIR to FIO5DIR- 0x2009 8000 to 0x2009 80A0) . . . . .	132	9.5.6.9	GPIO Interrupt Status for port 2 Falling Edge Interrupt (IO2IntStatF - 0x4002 80A8) . . . . .	148
9.5.2	GPIO port output Set register FIOxSET (FIO0SET to FIO5SET - 0x2009 8018 to 0x2009 80B8) . . . . .	133	9.5.6.10	GPIO Interrupt Clear register for port 0 (IO0IntClr - 0x4002 808C) . . . . .	149
9.5.3	GPIO port output Clear register FIOxCLR (FIO0CLR to FIO5CLR- 0x2009 801C to 0x2009 80BC) . . . . .	135	9.5.6.11	GPIO Interrupt Clear register for port 0 (IO2IntClr - 0x4002 80AC) . . . . .	150
9.5.4	GPIO port Pin value register FIOxPIN (FIO0PIN to FIO5PIN- 0x2009 8014 to 0x2009 80B4) . . . . .	136	<b>9.6</b>	<b>GPIO usage notes . . . . .</b>	<b>151</b>
9.5.5	Fast GPIO port Mask register FIOxMASK (FIO0MASK to FIO5MASK - 0x2009 8010 to 0x2009 80B0) . . . . .	138	9.6.1	Example: An instantaneous output of 0s and 1s on a GPIO port . . . . .	151
9.5.6	GPIO interrupt registers . . . . .	140	9.6.2	Writing to FIOSET/FIOCLR vs. FIOPIN . . . . .	151
9.5.6.1	GPIO overall Interrupt Status register (IOIntStatus - 0x4002 8080) . . . . .	140			

**Chapter 10: LPC178x/7x External Memory Controller (EMC)**

<b>10.1</b>	<b>How to read this chapter . . . . .</b>	<b>152</b>	10.5.6	Timing control with programmable delay elements . . . . .	158
<b>10.2</b>	<b>Basic configuration . . . . .</b>	<b>153</b>	<b>10.6</b>	<b>Low-power operation . . . . .</b>	<b>158</b>
<b>10.3</b>	<b>Introduction . . . . .</b>	<b>153</b>	10.6.1	Low-power SDRAM Deep-sleep Mode . . . . .	158
<b>10.4</b>	<b>Features . . . . .</b>	<b>154</b>	10.6.2	Low-power SDRAM partial array refresh . . . . .	158
<b>10.5</b>	<b>EMC functional description . . . . .</b>	<b>155</b>	<b>10.7</b>	<b>Memory bank select . . . . .</b>	<b>159</b>
10.5.1	AHB slave register interface . . . . .	155	<b>10.8</b>	<b>EMC Reset . . . . .</b>	<b>159</b>
10.5.2	AHB slave memory interface . . . . .	156	<b>10.9</b>	<b>Address shift mode . . . . .</b>	<b>159</b>
10.5.2.1	Memory transaction endianness . . . . .	156	<b>10.10</b>	<b>Memory mapped I/O and burst disable . . . . .</b>	<b>160</b>
10.5.2.2	Memory transaction size . . . . .	156	<b>10.11</b>	<b>Pin description . . . . .</b>	<b>161</b>
10.5.2.3	Write protected memory areas . . . . .	156	<b>10.12</b>	<b>Register description . . . . .</b>	<b>162</b>
10.5.3	Pad interface . . . . .	156	10.12.1	EMC Control register (EMCControl - 0x2009 C000) . . . . .	165
10.5.4	Data buffers . . . . .	156	10.12.2	EMC Status register (EMCStatus - 0x2009 C004) . . . . .	165
10.5.4.1	Write buffers . . . . .	156			
10.5.4.2	Read buffers . . . . .	157			
10.5.5	Memory controller state machine . . . . .	157			

10.12.3	EMC Configuration register (EMCConfig - 0x2009 C008) . . . . .	166	10.12.19	Dynamic Memory Configuration registers (EMCDynamicConfig0-3 - 0x2009 C100, 120, 140, 160) . . . . .	175
10.12.4	Dynamic Memory Control register (EMCDynamicControl - 0x2009 C020) . . . . .	166	10.12.20	Dynamic Memory RAS & CAS Delay registers (EMCDynamicRASCAS0-3 - 0x2009 C104, 124, 144, 164) . . . . .	177
10.12.5	Dynamic Memory Refresh Timer register (EMCDynamicRefresh - 0x2009 C024) . . . . .	167	10.12.21	Static Memory Configuration registers (EMCStaticConfig0-3 - 0x2009 C200, 220, 240, 260) . . . . .	178
10.12.6	Dynamic Memory Read Configuration register (EMCDynamicReadConfig - 0x2009 C028) . . . . .	168	10.12.22	Static Memory Write Enable Delay registers (EMCStaticWaitWen0-3 - 0x2009 C204, 224, 244, 264) . . . . .	179
10.12.7	Dynamic Memory Precharge Command Period register (EMCDynamictRP - 0x2009 C030) . . . . .	169	10.12.23	Static Memory Output Enable Delay registers (EMCStaticWaitOen0-3 - 0x2009 C208, 228, 248, 268) . . . . .	180
10.12.8	Dynamic Memory Active to Precharge Command Period register (EMCDynamictRAS - 0x2009 C034) . . . . .	169	10.12.24	Static Memory Read Delay registers (EMCStaticWaitRd0-3 - 0x2009 C20C, 22C, 24C, 26C) . . . . .	180
10.12.9	Dynamic Memory Self-refresh Exit Time register (EMCDynamictSREX - 0x2009 C038) . . . . .	170	10.12.25	Static Memory Page Mode Read Delay registers (EMCStaticwaitPage0-3 - 0x2009 C210, 230, 250, 270) . . . . .	181
10.12.10	Dynamic Memory Last Data Out to Active Time register (EMCDynamictAPR - 0x2009 C03C) . . . . .	170	10.12.26	Static Memory Write Delay registers (EMCStaticWaitWr0-3 - 0x2009 C214, 234, 254, 274) . . . . .	181
10.12.11	Dynamic Memory Data-in to Active Command Time register (EMCDynamictDAL - 0x2009 C040) . . . . .	171	10.12.27	Static Memory Turn Round Delay registers (EMCStaticWaitTurn0-3 - 0x2009 C218, 238, 258, 278) . . . . .	182
10.12.12	Dynamic Memory Write Recovery Time register (EMCDynamictWR - 0x2009 C044) . . . . .	171	10.12.28	Delay Control register (EMCDLYCTL - 0x400F C1DC) . . . . .	182
10.12.13	Dynamic Memory Active to Active Command Period register (EMCDynamictRC - 0x2009 C048) . . . . .	172	10.12.29	EMC Calibration register (EMCCAL - 0x400F C1E0) . . . . .	184
10.12.14	Dynamic Memory Auto-refresh Period register (EMCDynamictRFC - 0x2009 C04C) . . . . .	172		<b>10.13 External memory interface . . . . .</b>	<b>185</b>
10.12.15	Dynamic Memory Exit Self-refresh register (EMCDynamictXSR - 0x2009 C050) . . . . .	173	10.13.1	32-bit wide memory bank connection . . . . .	185
10.12.16	Dynamic Memory Active Bank A to Active Bank B Time register (EMCDynamictRRD - 0x2009 C054) . . . . .	173	10.13.2	16-bit wide memory bank connection . . . . .	186
10.12.17	Dynamic Memory Load Mode register to Active Command Time (EMCDynamictMRD - 0x2009 C058) . . . . .	174	10.13.3	8-bit wide memory bank connection . . . . .	187
10.12.18	Static Memory Extended Wait register (EMCStaticExtendedWait - 0x2009 C080) . . . . .	174	10.13.4	Memory configuration example . . . . .	188

**Chapter 11: LPC178x/7x Ethernet**

<b>11.1</b>	<b>Basic configuration . . . . .</b>	<b>189</b>	11.11.1	MAC Configuration Register 1 (MAC1 - 0x2008 4000) . . . . .	199
<b>11.2</b>	<b>Introduction . . . . .</b>	<b>189</b>	11.11.2	MAC Configuration Register 2 (MAC2 - 0x2008 4004) . . . . .	199
<b>11.3</b>	<b>Features . . . . .</b>	<b>190</b>	11.11.3	Back-to-Back Inter-Packet-Gap Register (IPGT - 0x2008 4008) . . . . .	201
<b>11.4</b>	<b>Architecture and operation . . . . .</b>	<b>191</b>	11.11.4	Non Back-to-Back Inter-Packet-Gap Register (IPGR - 0x2008 400C) . . . . .	201
<b>11.5</b>	<b>DMA engine functions . . . . .</b>	<b>192</b>	11.11.5	Collision Window / Retry Register (CLRT - 0x2008 4010) . . . . .	201
<b>11.6</b>	<b>Overview of DMA operation . . . . .</b>	<b>192</b>	11.11.6	Maximum Frame Register (MAXF - 0x2008 4014) . . . . .	202
<b>11.7</b>	<b>Ethernet packet . . . . .</b>	<b>193</b>	11.11.7	PHY Support Register (SUPP - 0x2008 4018) . . . . .	202
<b>11.8</b>	<b>Overview . . . . .</b>	<b>194</b>	11.11.8	Test Register (TEST - 0x2008 401C) . . . . .	202
11.8.1	Partitioning . . . . .	194	11.11.9	MII Mgmt Configuration Register (MCFG - 0x2008 4020) . . . . .	203
11.8.2	Example PHY Devices . . . . .	195			
<b>11.9</b>	<b>Pin description . . . . .</b>	<b>196</b>			
<b>11.10</b>	<b>Registers and software interface . . . . .</b>	<b>197</b>			
11.10.1	Register map . . . . .	197			
<b>11.11</b>	<b>Ethernet MAC register definitions . . . . .</b>	<b>199</b>			

11.11.10	MII Mgmt Command Register (MCMD - 0x2008 4024) . . . . .	204	11.13.2	Receive Filter WoL Status Register (RxFilterWoLStatus - 0x2008 4204) . . . . .	215
11.11.11	MII Mgmt Address Register (MADR - 0x2008 4028) . . . . .	204	11.13.3	Receive Filter WoL Clear Register (RxFilterWoLClear - 0x2008 4208) . . . . .	216
11.11.12	MII Mgmt Write Data Register (MWTD - 0x2008 402C) . . . . .	204	11.13.4	Hash Filter Table LSBs Register (HashFilterL - 0x2008 4210) . . . . .	216
11.11.13	MII Mgmt Read Data Register (MRDD - 0x2008 4030) . . . . .	205	11.13.5	Hash Filter Table MSBs Register (HashFilterH - 0x2008 4214) . . . . .	216
11.11.14	MII Mgmt Indicators Register (MIND - 0x2008 4034) . . . . .	205	<b>11.14 Module control register definitions . . . . .</b>	<b>217</b>	
11.11.15	Station Address 0 Register (SA0 - 0x2008 4040) . 205		11.14.1	Interrupt Status Register (IntStatus - 0x2008 4FE0) . . . . .	217
11.11.16	Station Address 1 Register (SA1 - 0x2008 4044) . 206		11.14.2	Interrupt Enable Register (IntEnable - 0x2008 4FE4) . . . . .	217
11.11.17	Station Address 2 Register (SA2 - 0x2008 4048) . 206		11.14.3	Interrupt Clear Register (IntClear - 0x2008 4FE8) 218	
<b>11.12 Control register definitions . . . . .</b>	<b>207</b>		11.14.4	Interrupt Set Register (IntSet - 0x2008 4FEC) . . . 218	
11.12.1	Command Register (Command - 0x2008 4100) . . 207		11.14.5	Power-Down Register (PowerDown - 0x2008 4FF4) . . . . .	219
11.12.2	Status Register (Status - 0x2008 4104) . . . . 207		<b>11.15 Descriptor and status formats . . . . .</b>	<b>220</b>	
11.12.3	Receive Descriptor Base Address Register (RxDescriptor - 0x2008 4108) . . . . .	208	11.15.1	Receive descriptors and statuses . . . . .	220
11.12.4	Receive Status Base Address Register (RxStatus - 0x2008 410C) . . . . .	208	11.15.2	Transmit descriptors and statuses . . . . .	223
11.12.5	Receive Number of Descriptors Register (RxDescriptor - 0x2008 4110) . . . . .	208	<b>11.16 Ethernet block functional description . . . . .</b>	<b>226</b>	
11.12.6	Receive Produce Index Register (RxProduceIndex - 0x2008 4114) . . . . .	209	11.16.1	Overview . . . . .	226
11.12.7	Receive Consume Index Register (RxConsumeIndex - 0x2008 4118) . . . . .	209	11.16.2	AHB interface . . . . .	226
11.12.8	Transmit Descriptor Base Address Register (TxDescriptor - 0x2008 411C) . . . . .	209	<b>11.17 Interrupts . . . . .</b>	<b>228</b>	
11.12.9	Transmit Status Base Address Register (TxStatus - 0x2008 4120) . . . . .	210	11.17.1	Direct Memory Access (DMA) . . . . .	228
11.12.10	Transmit Number of Descriptors Register (TxDescriptorNumber - 0x2008 4124) . . . . .	210	11.17.2	Initialization . . . . .	230
11.12.11	Transmit Produce Index Register (TxProduceIndex - 0x2008 4128) . . . . .	210	11.17.3	Transmit process . . . . .	232
11.12.12	Transmit Consume Index Register (TxConsumeIndex - 0x2008 412C) . . . . .	211	11.17.4	Receive process . . . . .	238
11.12.13	Transmit Status Vector 0 Register (TSV0 - 0x2008 4158) . . . . .	211	11.17.5	Transmission retry . . . . .	244
11.12.14	Transmit Status Vector 1 Register (TSV1 - 0x2008 415C) . . . . .	212	11.17.6	Status hash CRC calculations . . . . .	244
11.12.15	Receive Status Vector Register (RSV - 0x2008 4160) . . . . .	212	11.17.7	Duplex modes . . . . .	245
11.12.16	Flow Control Counter Register (FlowControlCounter - 0x2008 4170) . . . . .	213	11.17.8	IEE 802.3/Clause 31 flow control . . . . .	245
11.12.17	Flow Control Status Register (FlowControlStatus - 0x2008 4174) . . . . .	213	11.17.9	Half-Duplex mode backpressure . . . . .	247
<b>11.13 Receive filter register definitions . . . . .</b>	<b>215</b>		11.17.10	Receive filtering . . . . .	248
11.13.1	Receive Filter Control Register (RxFilterCtrl - 0x2008 4200) . . . . .	215	11.17.11	Power management . . . . .	250
			11.17.12	Wake-up on LAN . . . . .	251
			11.17.13	Enabling and disabling receive and transmit 252	
			11.17.14	Transmission padding and CRC . . . . .	254
			11.17.15	Huge frames and frame length checking . . . . .	255
			11.17.16	Statistics counters . . . . .	255
			11.17.17	MAC status vectors . . . . .	255
			11.17.18	Reset . . . . .	256
			11.17.19	Ethernet errors . . . . .	257
			<b>11.18 AHB bandwidth . . . . .</b>	<b>258</b>	
			11.18.1	DMA access . . . . .	258
			11.18.2	Types of CPU access . . . . .	259
			11.18.3	Overall bandwidth . . . . .	259
			<b>11.19 CRC calculation . . . . .</b>	<b>261</b>	

**Chapter 12: LPC178x/7x LCD controller**

<b>12.1</b>	<b>How to read this chapter . . . . .</b>	<b>263</b>	<b>12.3</b>	<b>Introduction . . . . .</b>	<b>263</b>
<b>12.2</b>	<b>Basic configuration . . . . .</b>	<b>263</b>	<b>12.4</b>	<b>Features . . . . .</b>	<b>264</b>

12.4.1	Programmable parameters . . . . .	264	12.7.4	Clock and Signal Polarity register (LCD_POL, RW - 0x2008 8008) . . . . .	289
12.4.2	Hardware cursor support . . . . .	265	12.7.5	Line End Control register (LCD_LE, RW - 0x2008 800C) . . . . .	290
12.4.3	Types of LCD panels supported . . . . .	265	12.7.6	Upper Panel Frame Base Address register (LCD_UPBASE, RW - 0x2008 8010) . . . . .	291
12.4.4	TFT panels . . . . .	265	12.7.7	Lower Panel Frame Base Address register (LCD_LPBASE, RW - 0x2008 8014) . . . . .	291
12.4.5	Color STN panels . . . . .	266	12.7.8	LCD Control register (LCD_CTRL, RW - 0x2008 8018) . . . . .	292
12.4.6	Monochrome STN panels . . . . .	266	12.7.9	Interrupt Mask register (LCD_INTMSK, RW - 0x2008 801C) . . . . .	293
<b>12.5</b>	<b>Pin description . . . . .</b>	<b>267</b>	12.7.10	Raw Interrupt Status register (LCD_INTRAW, RW - 0x2008 8020) . . . . .	294
12.5.1	Signal usage . . . . .	267	12.7.11	Masked Interrupt Status register (LCD_INTSTAT, RW - 0x2008 8024) . . . . .	295
12.5.1.1	Signals used for single panel STN displays . . . . .	267	12.7.12	Interrupt Clear register (LCD_INTCLR, RW - 0x2008 8028) . . . . .	295
12.5.1.2	Signals used for dual panel STN displays . . . . .	267	12.7.13	Upper Panel Current Address register (LCD_UPCURR, RW - 0x2008 802C) . . . . .	295
12.5.1.3	Signals used for TFT displays . . . . .	268	12.7.14	Lower Panel Current Address register (LCD_LPCURR, RW - 0x2008 8030) . . . . .	296
<b>12.6</b>	<b>LCD controller functional description . . . . .</b>	<b>269</b>	12.7.15	Color Palette registers (LCD_PAL, RW - 0x2008 8200 to 0x2008 83FC) . . . . .	296
12.6.1	AHB interfaces . . . . .	270	12.7.16	Cursor Image registers (CRSR_IMG, RW - 0x2008 8800 to 0x2008 8BFC) . . . . .	297
12.6.1.1	AMBA AHB slave interface . . . . .	270	12.7.17	Cursor Control register (CRSR_CTRL, RW - 0x2008 8C00) . . . . .	297
12.6.1.2	AMBA AHB master interface . . . . .	270	12.7.18	Cursor Configuration register (CRSR_CFG, RW - 0x2008 8C04) . . . . .	298
12.6.2	Dual DMA FIFOs and associated control logic . . . . .	271	12.7.19	Cursor Palette register 0 (CRSR_PAL0, RW - 0x2008 8C08) . . . . .	298
12.6.3	Pixel serializer . . . . .	271	12.7.20	Cursor Palette register 1 (CRSR_PAL1, RW - 0x2008 8C0C) . . . . .	299
12.6.4	RAM palette . . . . .	275	12.7.21	Cursor XY Position register (CRSR_XY, RW - 0x2008 8C10) . . . . .	299
12.6.5	Hardware cursor . . . . .	277	12.7.22	Cursor Clip Position register (CRSR_CLIP, RW - 0x2008 8C14) . . . . .	300
12.6.5.1	Cursor operation . . . . .	277	12.7.23	Cursor Interrupt Mask register (CRSR_INTMSK, RW - 0x2008 8C20) . . . . .	300
12.6.5.2	Cursor sizes . . . . .	277	12.7.24	Cursor Interrupt Clear register (CRSR_INTCLR, RW - 0x2008 8C24) . . . . .	300
12.6.5.3	Cursor movement . . . . .	278	12.7.25	Cursor Raw Interrupt Status register (CRSR_INTRAW, RW - 0x2008 8C28) . . . . .	301
12.6.5.4	Cursor XY positioning . . . . .	278	12.7.26	Cursor Masked Interrupt Status register (CRSR_INTSTAT, RW - 0x2008 8C2C) . . . . .	301
12.6.5.5	Cursor clipping . . . . .	279	<b>12.8</b>	<b>LCD timing diagrams . . . . .</b>	<b>302</b>
12.6.5.6	Cursor image format . . . . .	279	<b>12.9</b>	<b>LCD panel signal usage . . . . .</b>	<b>305</b>
12.6.6	Gray scaler . . . . .	282			
12.6.7	Upper and lower panel formatters . . . . .	282			
12.6.8	Panel clock generator . . . . .	283			
12.6.9	Timing controller . . . . .	283			
12.6.10	STN and TFT data select . . . . .	283			
12.6.10.1	STN displays . . . . .	283			
12.6.10.2	TFT displays . . . . .	283			
12.6.11	Interrupt generation . . . . .	283			
12.6.11.1	Master bus error interrupt . . . . .	284			
12.6.11.2	Vertical compare interrupt . . . . .	284			
12.6.11.2.1	Next base address update interrupt . . . . .	284			
12.6.11.2.2	FIFO underflow interrupt . . . . .	284			
12.6.12	LCD power-up and power-down sequence . . . . .	284			
<b>12.7</b>	<b>Register description . . . . .</b>	<b>286</b>			
12.7.1	LCD Configuration register (LCD_CFG, RW - 0x400F C1B8) . . . . .	287			
12.7.2	Horizontal Timing register (LCD_TIMH, RW - 0x2008 8000) . . . . .	287			
12.7.2.1	Horizontal timing restrictions . . . . .	287			
12.7.3	Vertical Timing register (LCD_TIMV, RW - 0x2008 8004) . . . . .	288			

**Chapter 13: LPC178x/7x USB device controller**

<b>13.1</b>	<b>How to read this chapter . . . . .</b>	<b>308</b>	13.6.1	Analog transceiver . . . . .	311
<b>13.2</b>	<b>Basic configuration . . . . .</b>	<b>308</b>	13.6.2	Serial Interface Engine (SIE) . . . . .	311
<b>13.3</b>	<b>Introduction . . . . .</b>	<b>308</b>	13.6.3	Endpoint RAM (EP_RAM) . . . . .	311
<b>13.4</b>	<b>Features . . . . .</b>	<b>309</b>	13.6.4	EP_RAM access control . . . . .	311
<b>13.5</b>	<b>Fixed endpoint configuration . . . . .</b>	<b>309</b>	13.6.5	DMA engine and bus master interface . . . . .	312
<b>13.6</b>	<b>Functional description . . . . .</b>	<b>310</b>	13.6.6	Register interface . . . . .	312
			13.6.7	SoftConnect . . . . .	312

13.6.8	GoodLink	312	13.10.6.4	USB Transmit Packet Length register (USBTxPLen - 0x2008 C224)	330
<b>13.7</b>	<b>Operational overview</b>	<b>312</b>	13.10.6.5	USB Control register (USBCtrl - 0x2008 C228)	330
<b>13.8</b>	<b>Pin description</b>	<b>313</b>	13.10.7	SIE command code registers	331
<b>13.9</b>	<b>Clocking and power management</b>	<b>313</b>	13.10.7.1	USB Command Code register (USBCmdCode - 0x2008 C210)	331
13.9.1	Power requirements	313	13.10.7.2	USB Command Data register (USBCmdData - 0x2008 C214)	331
13.9.2	Clocks	313	13.10.8	DMA registers	332
13.9.3	Power management support	314	13.10.8.1	USB DMA Request Status register (USBDMARSt - 0x2008 C250)	332
13.9.4	Remote wake-up	315	13.10.8.2	USB DMA Request Clear register (USBDMARClr - 0x2008 C254)	332
<b>13.10</b>	<b>Register description</b>	<b>315</b>	13.10.8.3	USB DMA Request Set register (USBDMARSet - 0x2008 C258)	333
13.10.1	Port select register	316	13.10.8.4	USB UDCA Head register (USBUDCAH - 0x2008 C280)	333
13.10.1.1	USB Port Select register (USBPortSel - 0x2008 C110)	316	13.10.8.5	USB EP DMA Status register (USBEPDMASt - 0x2008 C284)	334
13.10.2	Clock control registers	317	13.10.8.6	USB EP DMA Enable register (USBEPDMAEn - 0x2008 C288)	334
13.10.2.1	USB Clock Control register (USBClkCtrl - 0x2008 CFF4)	317	13.10.8.7	USB EP DMA Disable register (USBEPDMADis - 0x2008 C28C)	334
13.10.2.2	USB Clock Status register (USBClkSt - 0x2008 CFF8)	317	13.10.8.8	USB DMA Interrupt Status register (USBDMAIntSt - 0x2008 C290)	335
13.10.3	Device interrupt registers	318	13.10.8.9	USB DMA Interrupt Enable register (USBDMAIntEn - 0x2008 C294)	335
13.10.3.1	USB Interrupt Status register (USBIntSt - 0x2008 C1C0)	318	13.10.8.10	USB End of Transfer Interrupt Status register (USBEoTIntSt - 0x2008 C2A0)	335
13.10.3.2	USB Device Interrupt Status register (USBDevIntSt - 0x2008 C200)	318	13.10.8.11	USB End of Transfer Interrupt Clear register (USBEoTIntClr - 0x2008 C2A4)	336
13.10.3.3	USB Device Interrupt Enable register (USBDevIntEn - 0x2008 C204)	319	13.10.8.12	USB End of Transfer Interrupt Set register (USBEoTIntSet - 0x2008 C2A8)	336
13.10.3.4	USB Device Interrupt Clear register (USBDevIntClr - 0x2008 C208)	320	13.10.8.13	USB New DD Request Interrupt Status register (USBNDDRIntSt - 0x2008 C2AC)	336
13.10.3.5	USB Device Interrupt Set register (USBDevIntSet - 0x2008 C20C)	320	13.10.8.14	USB New DD Request Interrupt Clear register (USBNDDRIntClr - 0x2008 C2B0)	336
13.10.3.6	USB Device Interrupt Priority register (USBDevIntPri - 0x2008 C22C)	321	13.10.8.15	USB New DD Request Interrupt Set register (USBNDDRIntSet - 0x2008 C2B4)	336
13.10.4	Endpoint interrupt registers	322	13.10.8.16	USB System Error Interrupt Status register (USBSysErrIntSt - 0x2008 C2B8)	337
13.10.4.1	USB Endpoint Interrupt Status register (USBEPIntSt - 0x2008 C230)	322	13.10.8.17	USB System Error Interrupt Clear register (USBSysErrIntClr - 0x2008 C2BC)	337
13.10.4.2	USB Endpoint Interrupt Enable register (USBEPIntEn - 0x2008 C234)	323	13.10.8.18	USB System Error Interrupt Set register (USBSysErrIntSet - 0x2008 C2C0)	337
13.10.4.3	USB Endpoint Interrupt Clear register (USBEPIntClr - 0x2008 C238)	324	<b>13.11</b>	<b>Interrupt handling</b>	<b>338</b>
13.10.4.4	USB Endpoint Interrupt Set register (USBEPIntSet - 0x2008 C23C)	324		Slave mode	338
13.10.4.5	USB Endpoint Interrupt Priority register (USBEPIntPri - 0x2008 C240)	325		DMA mode	338
13.10.5	Endpoint realization registers	326	<b>13.12</b>	<b>Serial interface engine command description</b>	<b>341</b>
13.10.5.1	EP RAM requirements	326	13.12.1	Set Address (Command: 0xD0, Data: write 1 byte)	342
13.10.5.2	USB Realize Endpoint register (USBReEp - 0x2008 C244)	326	13.12.2	Configure Device (Command: 0xD8, Data: write 1 byte)	342
13.10.5.3	USB Endpoint Index register (USBEPIn - 0x2008 C248)	327	13.12.3	Set Mode (Command: 0xF3, Data: write 1 byte)	343
13.10.5.4	USB MaxPacketSize register (USBMaxPSize - 0x2008 C24C)	328			
13.10.6	USB transfer registers	329			
13.10.6.1	USB Receive Data register (USBRxData - 0x2008 C218)	329			
13.10.6.2	USB Receive Packet Length register (USBRxPLen - 0x2008 C220)	329			
13.10.6.3	USB Transmit Data register (USBTxData - 0x2008 C21C)	329			

13.12.4	Read Current Frame Number (Command: 0xF5, Data: read 1 or 2 bytes) . . . . .	343	13.15.4.8	DD_retired . . . . .	357
13.12.5	Read Test Register (Command: 0xFD, Data: read 2 bytes) . . . . .	344	13.15.4.9	DD_status . . . . .	357
13.12.6	Set Device Status (Command: 0xFE, Data: write 1 byte) . . . . .	344	13.15.4.10	Packet_valid . . . . .	358
13.12.7	Get Device Status (Command: 0xFE, Data: read 1 byte) . . . . .	345	13.15.4.11	LS_byte_extracted . . . . .	358
13.12.8	Get Error Code (Command: 0xFF, Data: read 1 byte) . . . . .	345	13.15.4.12	MS_byte_extracted . . . . .	358
13.12.9	Read Error Status (Command: 0xFB, Data: read 1 byte) . . . . .	346	13.15.4.13	Present_DMA_count . . . . .	358
13.12.10	Select Endpoint (Command: 0x00 - 0x1F, Data: read 1 byte (optional)) . . . . .	346	13.15.4.14	Message_length_position . . . . .	358
13.12.11	Select Endpoint/Clear Interrupt (Command: 0x40 - 0x5F, Data: read 1 byte) . . . . .	347	13.15.4.15	Isochronous_packet_size_memory_address . . . . .	358
13.12.12	Set Endpoint Status (Command: 0x40 - 0x55, Data: write 1 byte (optional)) . . . . .	348	13.15.5	Non-isochronous endpoint operation . . . . .	359
13.12.13	Clear Buffer (Command: 0xF2, Data: read 1 byte (optional)) . . . . .	348	13.15.5.1	Setting up DMA transfers . . . . .	359
13.12.14	Validate Buffer (Command: 0xFA, Data: none) . . . . .	349	13.15.5.2	Finding DMA Descriptor . . . . .	359
<b>13.13</b>	<b>USB device controller initialization . . . . .</b>	<b>350</b>	13.15.5.3	Transferring the data . . . . .	359
<b>13.14</b>	<b>Slave mode operation . . . . .</b>	<b>352</b>	13.15.5.4	Optimizing descriptor fetch . . . . .	359
13.14.1	Interrupt generation . . . . .	352	13.15.5.5	Ending the packet transfer . . . . .	360
13.14.2	Data transfer for OUT endpoints . . . . .	352	13.15.5.6	No_Packet DD . . . . .	360
13.14.3	Data transfer for IN endpoints . . . . .	352	13.15.6	Isochronous endpoint operation . . . . .	360
<b>13.15</b>	<b>DMA operation . . . . .</b>	<b>354</b>	13.15.6.1	Setting up DMA transfers . . . . .	360
13.15.1	Transfer terminology . . . . .	354	13.15.6.2	Finding the DMA Descriptor . . . . .	361
13.15.2	USB device communication area . . . . .	354	13.15.6.3	Transferring the Data . . . . .	361
13.15.3	Triggering the DMA engine . . . . .	355	OUT endpoints . . . . .	361	
13.15.4	The DMA descriptor . . . . .	355	IN endpoints . . . . .	361	
13.15.4.1	Next_DD_pointer . . . . .	356	13.15.6.4	DMA descriptor completion . . . . .	361
13.15.4.2	DMA_mode . . . . .	357	13.15.6.5	Isochronous OUT Endpoint Operation Example . . . . .	361
13.15.4.3	Next_DD_valid . . . . .	357	13.15.7	Auto Length Transfer Extraction (ATLE) mode operation . . . . .	362
13.15.4.4	Isochronous_endpoint . . . . .	357	OUT transfers in ATLE mode . . . . .	362	
13.15.4.5	Max_packet_size . . . . .	357	IN transfers in ATLE mode . . . . .	364	
13.15.4.6	DMA_buffer_length . . . . .	357	13.15.7.1	Setting up the DMA transfer . . . . .	364
13.15.4.7	DMA_buffer_start_addr . . . . .	357	13.15.7.2	Finding the DMA Descriptor . . . . .	364
			13.15.7.3	Transferring the Data . . . . .	364
			OUT endpoints . . . . .	364	
			IN endpoints . . . . .	364	
			13.15.7.4	Ending the packet transfer . . . . .	365
			OUT endpoints . . . . .	365	
			IN endpoints . . . . .	365	
			<b>13.16</b>	<b>Double buffered endpoint operation . . . . .</b>	<b>366</b>
			13.16.1	Bulk endpoints . . . . .	366
			13.16.2	Isochronous endpoints . . . . .	367

**Chapter 14: LPC178x/7x USB Host controller**

<b>14.1</b>	<b>How to read this chapter . . . . .</b>	<b>369</b>	<b>14.4</b>	<b>Interfaces . . . . .</b>	<b>372</b>
<b>14.2</b>	<b>Basic configuration . . . . .</b>	<b>369</b>	14.4.1	Pin description . . . . .	372
<b>14.3</b>	<b>Introduction . . . . .</b>	<b>370</b>	14.4.1.1	USB host usage note . . . . .	372
14.3.1	Features . . . . .	370	14.4.2	Software interface . . . . .	373
14.3.2	Architecture . . . . .	371	14.4.2.1	Register map . . . . .	373
			14.4.2.2	USB Host Register Definitions . . . . .	374

**Chapter 15: LPC178x/7x USB OTG controller**

<b>15.1</b>	<b>How to read this chapter . . . . .</b>	<b>375</b>	<b>15.7</b>	<b>Pin configuration . . . . .</b>	<b>377</b>
<b>15.2</b>	<b>Basic configuration . . . . .</b>	<b>375</b>	15.7.1	Using port U1 for OTG operation . . . . .	377
<b>15.3</b>	<b>Introduction . . . . .</b>	<b>375</b>	15.7.2	Using both ports U1 and U2 for host operation . . . . .	380
<b>15.4</b>	<b>Features . . . . .</b>	<b>375</b>	15.7.3	Using U1 for host operation and U2 for device operation . . . . .	381
<b>15.5</b>	<b>Architecture . . . . .</b>	<b>376</b>	<b>15.8</b>	<b>Register description . . . . .</b>	<b>382</b>
<b>15.6</b>	<b>Modes of operation . . . . .</b>	<b>376</b>			



15.8.1	USB Interrupt Status Register (USBIntSt - 0x2008 C1C0) . . . . .	382	15.8.13	I2C Control Register (I2C_CTL - 0x2008 C308) . . . . .	388
15.8.2	OTG Interrupt Status Register (OTGIntSt - 0x2008 C100) . . . . .	383	15.8.14	I2C Clock High Register (I2C_CLKHI - 0x2008 C30C) . . . . .	390
15.8.3	OTG Interrupt Enable Register (OTGIntEn - 0x2008 C104) . . . . .	383	15.8.15	I2C Clock Low Register (I2C_CLKLO - 0x2008 C310) . . . . .	390
15.8.4	OTG Interrupt Set Register (OTGIntSet - 0x2008 C20C) . . . . .	383	15.8.16	Interrupt handling . . . . .	390
15.8.5	OTG Interrupt Clear Register (OTGIntClr - 0x2008 C10C) . . . . .	383	<b>15.9 HNP support . . . . .</b>	<b>392</b>	
15.8.6	OTG Status and Control Register (OTGStCtrl - 0x2008 C110) . . . . .	383	15.9.1	B-device: peripheral to host switching . . . . .	393
15.8.7	OTG Timer Register (OTGTmr - 0x2008 C114) . . . . .	385		Remove D+ pull-up . . . . .	395
15.8.8	OTG Clock Control Register (OTGClkCtrl - 0x2008 CFF4) . . . . .	385		Add D+ pull-up . . . . .	396
15.8.9	OTG Clock Status Register (OTGClkSt - 0x2008 CFF8) . . . . .	386	15.9.2	A-device: host to peripheral HNP switching . . . . .	396
15.8.10	I2C Receive Register (I2C_RX - 0x2008 C300) . . . . .	386		Set BDIS_ACON_EN in external OTG transceiver . . . . .	398
15.8.11	I2C Transmit Register (I2C_TX - 0x2008 C300) . . . . .	386		Clear BDIS_ACON_EN in external OTG transceiver . . . . .	399
15.8.12	I2C Status Register (I2C_STS - 0x2008 C304) . . . . .	387		Discharge V <sub>BUS</sub> . . . . .	399
				Load and enable OTG timer . . . . .	400
				Stop OTG timer . . . . .	400
				Suspend host on port 1 . . . . .	400
			<b>15.10 Clocking and power management . . . . .</b>	<b>401</b>	
			15.10.1	Device clock request signals . . . . .	402
			15.10.1.1	Host clock request signals . . . . .	403
			15.10.2	Power-down mode support . . . . .	403
			<b>15.11 USB OTG controller initialization . . . . .</b>	<b>404</b>	

**Chapter 16: LPC178x/7x SD card interface**

<b>16.1 Basic configuration . . . . .</b>	<b>405</b>	16.6.2	Clock Control Register (MCIClock - 0x400C 0004) . . . . .	419	
<b>16.2 Introduction . . . . .</b>	<b>405</b>	16.6.3	Argument Register (MCIArument - 0x400C 0008) . . . . .	420	
<b>16.3 Features . . . . .</b>	<b>405</b>	16.6.4	Command Register (MCICommand - 0x400C 000C) . . . . .	420	
<b>16.4 Pin description . . . . .</b>	<b>406</b>	16.6.5	Command Response Register (MCIRspCommand - 0x400C 0010) . . . . .	420	
<b>16.5 Functional overview . . . . .</b>	<b>406</b>	16.6.6	Response Registers (MCIResponse0-3 - 0x400C 0014, 0x400C 0018, 0x400C 001C and 0x400C 0020) . . . . .	421	
16.5.1	Secure digital memory card . . . . .	406	16.6.7	Data Timer Register (MCIDataTimer - 0x400C 0024) . . . . .	421
16.5.1.1	Secure digital memory card bus signals . . . . .	406	16.6.8	Data Length Register (MCIDataLength - 0x400C 0028) . . . . .	422
16.5.2	Multimedia card . . . . .	406	16.6.9	Data Control Register (MCIDataCtrl - 0x400C 002C) . . . . .	422
16.5.3	SD card interface details . . . . .	407	16.6.10	Data Counter Register (MCIDataCnt - 0x400C 0030) . . . . .	423
16.5.3.1	Adapter register block . . . . .	408	16.6.11	Status Register (MCIStatus - 0x400C 0034) . . . . .	423
16.5.3.2	Control unit . . . . .	408	16.6.12	Clear Register (MCIClear - 0x400C 0038) . . . . .	424
16.5.3.3	Command path . . . . .	409	16.6.13	Interrupt Mask Registers (MCIMask0 - 0x400C 003C) . . . . .	424
16.5.3.4	Command path state machine . . . . .	409	16.6.14	FIFO Counter Register (MCIFifoCnt - 0x400C 0048) . . . . .	425
16.5.3.5	Command format . . . . .	410	16.6.15	Data FIFO Register (MCIFIFO - 0x400C 0080 to 0x400C 00BC) . . . . .	426
16.5.3.6	Data path . . . . .	411			
16.5.3.7	Data path state machine . . . . .	412			
16.5.3.8	Data counter . . . . .	413			
16.5.3.9	Bus mode . . . . .	414			
16.5.3.10	CRC Token status . . . . .	414			
16.5.3.11	Status flags . . . . .	415			
16.5.3.12	CRC generator . . . . .	415			
16.5.3.13	Data FIFO . . . . .	415			
16.5.3.14	Transmit FIFO . . . . .	416			
16.5.3.15	Receive FIFO . . . . .	416			
16.5.3.16	APB interfaces . . . . .	417			
16.5.3.17	Interrupt logic . . . . .	417			
<b>16.6 Register description . . . . .</b>	<b>418</b>				
16.6.1	Power Control Register (MCIPWR - 0x400C 0000) . . . . .	418			

**Chapter 17: LPC178x/7x UART1**

<b>17.1</b>	<b>Basic configuration</b> .....	<b>427</b>	17.5.11	UART1 Modem Status Register (U1MSR - 0x4001 0018) .....	441
<b>17.2</b>	<b>Features</b> .....	<b>427</b>	17.5.12	UART1 Scratch Pad Register (U1SCR - 0x4001 001C) .....	442
<b>17.3</b>	<b>Architecture</b> .....	<b>428</b>	17.5.13	UART1 Auto-baud Control Register (U1ACR - 0x4001 0020) .....	442
<b>17.4</b>	<b>Pin description</b> .....	<b>430</b>	17.5.14	Auto-baud .....	443
<b>17.5</b>	<b>Register description</b> .....	<b>431</b>	17.5.15	Auto-baud modes .....	444
17.5.1	UART1 Receiver Buffer Register (U1RBR - 0x4001 0000, when DLAB = 0) .....	432	17.5.16	UART1 Fractional Divider Register (U1FDR - 0x4001 0028) .....	445
17.5.2	UART1 Transmitter Holding Register (U1THR - 0x4001 0000 when DLAB = 0) .....	432	17.5.16.1	Baud rate calculation .....	446
17.5.3	UART1 Divisor Latch LSB and MSB Registers (U1DLL - 0x4001 0000 and U1DLM - 0x4001 0004, when DLAB = 1) .....	432	17.5.16.1.1	Example 1: PCLK = 14.7456 MHz, BR = 9600 . 448	
17.5.4	UART1 Interrupt Enable Register (U1IER - 0x4001 0004, when DLAB = 0) .....	433	17.5.16.1.2	Example 2: PCLK = 12 MHz, BR = 115200 448	
17.5.5	UART1 Interrupt Identification Register (U1IIR - 0x4001 0008) .....	434	17.5.17	UART1 Transmit Enable Register (U1TER - 0x4001 0030) .....	448
17.5.6	UART1 FIFO Control Register (U1FCR - 0x4001 0008) .....	436	17.5.18	UART1 RS485 Control register (U1RS485CTRL - 0x4001 004C) .....	449
17.5.6.1	DMA Operation .....	436	17.5.19	UART1 RS-485 Address Match register (U1RS485ADRMATCH - 0x4001 0050) . . . . .	450
	UART receiver DMA .....	437	17.5.20	UART1 RS-485 Delay value register (U1RS485DLY - 0x4001 0054) .....	450
	UART transmitter DMA .....	437	17.5.21	RS-485/EIA-485 modes of operation . . . . .	450
17.5.7	UART1 Line Control Register (U1LCR - 0x4001 000C) .....	437		RS-485/EIA-485 Normal Multidrop Mode (NMM) 450	
17.5.8	UART1 Modem Control Register (U1MCR - 0x4001 0010) .....	437		RS-485/EIA-485 Auto Address Detection (AAD) mode .....	450
17.5.9	Auto-flow control .....	438		RS-485/EIA-485 Auto Direction Control. . . . .	451
17.5.9.1	Auto-RTS .....	438		RS485/EIA-485 driver delay time. . . . .	451
17.5.9.2	Auto-CTS .....	439		RS485/EIA-485 output inversion .....	451
17.5.10	UART1 Line Status Register (U1LSR - 0x4001 0014) .....	440			

**Chapter 18: LPC178x/7x UART0/2/3**

<b>18.1</b>	<b>How to read this chapter</b> .....	<b>452</b>	18.6.5	UARTn Interrupt Identification Register (U0IIR - 0x4000 C008, U2IIR - 0x4009 8008, U3IIR - 0x4009 C008) .....	458
<b>18.2</b>	<b>Basic configuration</b> .....	<b>452</b>	18.6.6	UARTn FIFO Control Register (U0FCR - 0x4000 C008, U2FCR - 0x4009 8008, U3FCR - 0x4009 C008) .....	460
<b>18.3</b>	<b>Features</b> .....	<b>453</b>	18.6.6.1	DMA Operation .....	461
<b>18.4</b>	<b>Architecture</b> .....	<b>453</b>		UART receiver DMA .....	461
<b>18.5</b>	<b>Pin description</b> .....	<b>454</b>		UART transmitter DMA .....	461
<b>18.6</b>	<b>Register description</b> .....	<b>455</b>	18.6.7	UARTn Line Control Register (U0LCR - 0x4000 C00C, U2LCR - 0x4009 800C, U3LCR - 0x4009 C00C) .....	461
18.6.1	UARTn Receiver Buffer Register (U0RBR - 0x4000 C000, U2RBR - 0x4009 8000, U3RBR - 0x4009 C000 when DLAB = 0) .....	456	18.6.8	UARTn Line Status Register (U0LSR - 0x4000 C014, U2LSR - 0x4009 8014, U3LSR - 0x4009 C014) .....	462
18.6.2	UARTn Transmit Holding Register (U0THR - 0x4000 C000, U2THR - 0x4009 8000, U3THR - 0x4009 C000 when DLAB = 0) .....	456	18.6.9	UARTn Scratch Pad Register (U0SCR - 0x4000 C01C, U2SCR - 0x4009 801C U3SCR - 0x4009 C01C) .....	463
18.6.3	UARTn Divisor Latch LSB register (U0DLL - 0x4000 C000, U2DLL - 0x4009 8000, U3DLL - 0x4009 C000 when DLAB = 1) and UARTn Divisor Latch MSB register (U0DLM - 0x4000 C004, U2DLM - 0x4009 8004, U3DLM - 0x4009 C004 when DLAB = 1) .....	457	18.6.10	UARTn Auto-baud Control Register (U0ACR - 0x4000 C020, U2ACR - 0x4009 8020, U3ACR - 0x4009 C020) .....	464
18.6.4	UARTn Interrupt Enable Register (U0IER - 0x4000 C004, U2IER - 0x4009 8004, U3IER - 0x4009 C004 when DLAB = 0) .....	457	18.6.10.1	Auto-baud .....	464
			18.6.10.2	Auto-baud modes .....	465

18.6.11	UARTn Fractional Divider Register (U0FDR - 0x4000 C028, U2FDR - 0x4009 8028, U3FDR - 0x4009 C028) . . . . .	466	18.6.14	UARTn RS-485 Address Match register (U0RS485ADRMATCH - 0x4000 C050, U2RS485ADRMATCH - 0x4009 8050, U3RS485ADRMATCH - 0x4009 C050) . . . . .	470
18.6.11.1	Baud rate calculation . . . . .	467	18.6.15	UARTn RS-485 Delay value register (U0RS485DLY - 0x4000 C054, U2RS485DLY - 0x4009 8054, U3RS485DLY - 0x4009 C054) . . . . .	471
18.6.11.1.1	Example 1: PCLK = 14.7456 MHz, BR = 9600 . . . . .	469	18.6.16	RS-485/EIA-485 modes of operation . . . . .	471
18.6.11.1.2	Example 2: PCLK = 12 MHz, BR = 115200 . . . . .	469		RS-485/EIA-485 Normal Multidrop Mode (NMM) . . . . .	471
18.6.12	UARTn Transmit Enable Register (U0TER - 0x4000 C030, U2TER - 0x4009 8030, U3TER - 0x4009 C030) . . . . .	469		RS-485/EIA-485 Auto Address Detection (AAD) mode . . . . .	471
18.6.13	UARTn RS485 Control register (U0RS485CTRL - 0x4000 C04C, U2RS485CTRL - 0x4009 804C, U3RS485CTRL - 0x4009 C04C) . . . . .	470		RS-485/EIA-485 Auto Direction Control . . . . .	472
				RS485/EIA-485 driver delay time . . . . .	472
				RS485/EIA-485 output inversion . . . . .	472

**Chapter 19: LPC178x/7x UART4**

<b>19.1</b>	<b>How to read this chapter . . . . .</b>	<b>473</b>	19.6.10.2	Auto-baud modes . . . . .	485
<b>19.2</b>	<b>Basic configuration . . . . .</b>	<b>473</b>	19.6.11	UART4 IrDA Control Register (U4ICR - 0x400A C024) . . . . .	486
<b>19.3</b>	<b>Features . . . . .</b>	<b>474</b>	19.6.12	UART4 Fractional Divider Register (U4FDR - 0x400A C028) . . . . .	487
<b>19.4</b>	<b>Architecture . . . . .</b>	<b>474</b>	19.6.12.1	Baud rate calculation . . . . .	488
<b>19.5</b>	<b>Pin description . . . . .</b>	<b>475</b>	19.6.12.1.1	Example 1: PCLK = 14.7456 MHz, BR = 9600 . . . . .	490
<b>19.6</b>	<b>Register description . . . . .</b>	<b>476</b>	19.6.12.1.2	Example 2: PCLK = 12 MHz, BR = 115200 . . . . .	490
19.6.1	UART4 Receiver Buffer Register (U4RBR - 0x400A C000 when DLAB = 0) . . . . .	477	19.6.13	UART4 Oversampling Register (U4OSR - 0x400A 402C) . . . . .	490
19.6.2	UART4 Transmit Holding Register (U4THR - 0x400A C000 when DLAB = 0) . . . . .	477	19.6.14	UART4 Smart Card Interface Control register (U4SCICTRL - 0x400A 4048) . . . . .	491
19.6.3	UART4 Divisor Latch LSB register (U4DLL - 0x400A 4000 when DLAB = 1) and UART4 Divisor Latch MSB register (U4DLM - 0x400A C004 when DLAB = 1) . . . . .	477	19.6.14.1	Smart Card Connection . . . . .	492
19.6.4	UART4 Interrupt Enable Register (U4IER - 0x400A C004 when DLAB = 0) . . . . .	478	19.6.14.2	Smart Card Setup . . . . .	492
19.6.5	UART4 Interrupt Identification Register (U4IIR - 0x400A C008) . . . . .	478	19.6.15	UART4 RS485 Control register (U4RS485CTRL - 0x400A 404C) . . . . .	492
19.6.6	UART4 FIFO Control Register (U4FCR - 0x400A C008) . . . . .	480	19.6.16	UART4 RS-485 Address Match register (U4RS485ADRMATCH - 0x400A 4050) . . . . .	493
19.6.6.1	DMA Operation . . . . .	481	19.6.17	UART4 RS-485 Delay value register (U4RS485DLY - 0x400A 4054) . . . . .	493
	UART receiver DMA . . . . .	481	19.6.18	RS-485/EIA-485 modes of operation . . . . .	493
	UART transmitter DMA . . . . .	481		RS-485/EIA-485 Normal Multidrop Mode (NMM) . . . . .	493
19.6.7	UART4 Line Control Register (U4LCR - 0x400A C00C) . . . . .	481		RS-485/EIA-485 Auto Address Detection (AAD) mode . . . . .	494
19.6.8	UART4 Line Status Register (U4LSR - 0x400A C014) . . . . .	482		RS-485/EIA-485 Auto Direction Control . . . . .	494
19.6.9	UART4 Scratch Pad Register (U4SCR - 0x400A C01C) . . . . .	483		RS485/EIA-485 driver delay time . . . . .	494
19.6.10	UART4 Auto-baud Control Register (U4ACR - 0x400A C020) . . . . .	484		RS485/EIA-485 output inversion . . . . .	494
19.6.10.1	Auto-baud . . . . .	484	19.6.19	UART4 Synchronous mode control register (U4SYNCCTRL - 0x400A 4058) . . . . .	495
			19.6.20	UART4 Transmit Enable Register (U4TER - 0x400A C05C) . . . . .	496

**Chapter 20: LPC178x/7x CAN controller**

<b>20.1</b>	<b>Basic configuration . . . . .</b>	<b>498</b>	20.3.3	Acceptance filter features . . . . .	499
<b>20.2</b>	<b>CAN controllers . . . . .</b>	<b>498</b>	<b>20.4</b>	<b>Pin description . . . . .</b>	<b>499</b>
<b>20.3</b>	<b>Features . . . . .</b>	<b>498</b>	<b>20.5</b>	<b>CAN controller architecture . . . . .</b>	<b>499</b>
20.3.1	General CAN features . . . . .	498	20.5.1	APB Interface Block (AIB) . . . . .	500
20.3.2	CAN controller features . . . . .	499	20.5.2	Interface Management Logic (IML) . . . . .	500

20.5.3	Transmit Buffers (TXB) . . . . .	500	<b>20.8</b>	<b>CAN controller operation . . . . .</b>	<b>525</b>
20.5.4	Receive Buffer (RXB) . . . . .	501	20.8.1	Error handling . . . . .	525
20.5.5	Error Management Logic (EML) . . . . .	502	20.8.2	Sleep mode . . . . .	525
20.5.6	Bit Timing Logic (BTL) . . . . .	502	20.8.3	Interrupts . . . . .	526
20.5.7	Bit Stream Processor (BSP) . . . . .	502	20.8.4	Transmit priority . . . . .	526
20.5.8	CAN controller self-tests . . . . .	502	<b>20.9</b>	<b>Centralized CAN registers . . . . .</b>	<b>526</b>
	Global self test . . . . .	503	20.9.1	Central Transmit Status Register (CANTxSR - 0x4004 0000) . . . . .	526
	Local self test . . . . .	503	20.9.2	Central Receive Status Register (CANRxSR - 0x4004 0004) . . . . .	527
<b>20.6</b>	<b>Memory map of the CAN block. . . . .</b>	<b>504</b>	20.9.3	Central Miscellaneous Status Register (CANMSR - 0x4004 0008) . . . . .	527
<b>20.7</b>	<b>CAN controller registers . . . . .</b>	<b>504</b>	<b>20.10</b>	<b>Global acceptance filter . . . . .</b>	<b>528</b>
20.7.1	CAN Mode register (CAN1MOD - 0x4004 4000, CAN2MOD - 0x4004 8000) . . . . .	506	<b>20.11</b>	<b>Acceptance filter modes. . . . .</b>	<b>528</b>
20.7.2	CAN Command Register (CAN1CMR - 0x4004 x004, CAN2CMR - 0x4004 8004) . . . . .	508	20.11.1	Acceptance filter Off mode. . . . .	528
20.7.3	CAN Global Status Register (CAN1GSR - 0x4004 x008, CAN2GSR - 0x4004 8008) . . . . .	509	20.11.2	Acceptance filter Bypass mode . . . . .	528
	RX error counter . . . . .	511	20.11.3	Acceptance filter Operating mode . . . . .	529
	TX error counter . . . . .	511	20.11.4	FullCAN mode . . . . .	529
20.7.4	CAN Interrupt and Capture Register (CAN1ICR - 0x4004 400C, CAN2ICR - 0x4004 800C) . . . . .	512	<b>20.12</b>	<b>Sections of the ID look-up table RAM . . . . .</b>	<b>529</b>
20.7.5	CAN Interrupt Enable Register (CAN1IER - 0x4004 4010, CAN2IER - 0x4004 8010) . . . . .	515	<b>20.13</b>	<b>ID look-up table RAM . . . . .</b>	<b>529</b>
20.7.6	CAN Bus Timing Register (CAN1BTR - 0x4004 4014, CAN2BTR - 0x4004 8014) . . . . .	516	<b>20.14</b>	<b>Acceptance filter registers . . . . .</b>	<b>531</b>
	Baud rate prescaler . . . . .	517	20.14.1	Acceptance Filter Mode Register (AFMR - 0x4003 C000) . . . . .	531
	Synchronization jump width . . . . .	517	20.14.2	Section configuration registers. . . . .	532
	Time segment 1 and time segment 2 . . . . .	517	20.14.3	Standard Frame Individual Start Address register (SFF_sa - 0x4003 C004) . . . . .	532
20.7.7	CAN Error Warning Limit register (CAN1EWL - 0x4004 4018, CAN2EWL - 0x4004 8018) . . . . .	517	20.14.4	Standard Frame Group Start Address register (SFF_GRP_sa - 0x4003 C008) . . . . .	533
20.7.8	CAN Status Register (CAN1SR - 0x4004 401C, CAN2SR - 0x4004 801C) . . . . .	518	20.14.5	Extended Frame Start Address register (EFF_sa - 0x4003 C00C) . . . . .	533
20.7.9	CAN Receive Frame Status register (CAN1RFS - 0x4004 4020, CAN2RFS - 0x4004 8020) . . . . .	520	20.14.6	Extended Frame Group Start Address register (EFF_GRP_sa - 0x4003 C010) . . . . .	533
20.7.9.1	ID index field. . . . .	520	20.14.7	End of AF Tables register (ENDofTable - 0x4003 C014) . . . . .	534
20.7.10	CAN Receive Identifier register (CAN1RID - 0x4004 4024, CAN2RID - 0x4004 8024) . . . . .	520	20.14.8	Status registers . . . . .	534
20.7.11	CAN Receive Data register A (CAN1RDA - 0x4004 4028, CAN2RDA - 0x4004 8028) . . . . .	521	20.14.9	LUT Error Address register (LUTerrAd - 0x4003 C018) . . . . .	534
20.7.12	CAN Receive Data register B (CAN1RDB - 0x4004 402C, CAN2RDB - 0x4004 802C) . . . . .	521	20.14.10	LUT Error register (LUTerr - 0x4003 C01C) . . . . .	534
20.7.13	CAN Transmit Frame Information register (CAN1TFI[1/2/3] - 0x4004 40[30/ 40/50], CAN2TFI[1/2/3] - 0x4004 80[30/40/50]) . . . . .	522	20.14.11	Global FullCANInterrupt Enable register (FCANIE - 0x4003 C020) . . . . .	535
	Automatic transmit priority detection. . . . .	522	20.14.12	FullCAN Interrupt and Capture registers (FCANIC0 - 0x4003 C024 and FCANIC1 - 0x4003 C028) . . . . .	535
	Tx DLC . . . . .	523	<b>20.15</b>	<b>Configuration and search algorithm . . . . .</b>	<b>535</b>
20.7.14	CAN Transmit Identifier register (CAN1TID[1/2/3] - 0x4004 40[34/44/54], CAN2TID[1/2/3] - 0x4004 80[34/44/54]) . . . . .	523	20.15.1	Acceptance filter search algorithm. . . . .	535
20.7.15	CAN Transmit Data register A (CAN1TDA[1/2/3] - 0x4004 40[38/48/58], CAN2TDA[1/2/3] - 0x4004 80[38/48/58]) . . . . .	523	<b>20.16</b>	<b>FullCAN mode . . . . .</b>	<b>537</b>
20.7.16	CAN Transmit Data register B (CAN1TDB[1/2/3] - 0x4004 40[3C/4C/5C], CAN2TDB[1/2/3] - 0x4004 80[3C/4C/5C]) . . . . .	524	20.16.1	FullCAN message layout . . . . .	538
20.7.17	CAN Sleep Clear register (CANSLEEPCLR - 0x400F C110) . . . . .	524	20.16.2	FullCAN interrupts . . . . .	541
20.7.18	CAN Wake-up Flags register (CANWAKEFLAGS - 0x400F C114) . . . . .	525	20.16.2.1	FullCAN message interrupt enable bit . . . . .	541
			20.16.2.2	Message lost bit and CAN channel number. . . . .	542
			20.16.2.3	Setting the interrupt pending bits (IntPnd 63 to 0) . . . . .	543
			20.16.2.4	Clearing the interrupt pending bits (IntPnd 63 to 0) . . . . .	543
			20.16.2.5	Setting the message lost bit of a FullCAN message object (MsgLost 63 to 0) . . . . .	543

20.16.2.6	Clearing the message lost bit of a FullCAN message object (MsgLost 63 to 0) . . . . .	543	20.17.4	Configuration example 4 . . . . .	549
20.16.3	Set and clear mechanism of the FullCAN interrupt 543		20.17.5	Configuration example 5 . . . . .	549
20.16.3.1	Scenario 1: Normal case, no message lost . . . . .	543	20.17.6	Configuration example 6 . . . . .	550
20.16.3.2	Scenario 2: Message lost . . . . .	544		Explicit standard frame format identifier section (11-bit CAN ID): . . . . .	551
20.16.3.3	Scenario 3: Message gets overwritten indicated by Semaphore bits . . . . .	545		Group of standard frame format identifier section (11-bit CAN ID): . . . . .	551
20.16.3.4	Scenario 3.1: Message gets overwritten indicated by Semaphore bits and Message Lost . . . . .	545		Explicit extended frame format identifier section (29-bit CAN ID, <a href="#">Figure 96</a> ) . . . . .	551
20.16.3.5	Scenario 3.2: Message gets overwritten indicated by Message Lost . . . . .	546	20.17.7	Configuration example 7 . . . . .	552
20.16.3.6	Scenario 4: Clearing Message Lost bit. . . . .	547		FullCAN explicit standard frame format identifier section (11-bit CAN ID) . . . . .	553
<b>20.17</b>	<b>Examples of acceptance filter tables and ID index values . . . . .</b>	<b>548</b>		Explicit standard frame format identifier section (11-bit CAN ID) . . . . .	553
20.17.1	Example 1: only one section is used . . . . .	548		FullCAN message object data section . . . . .	553
20.17.2	Example 2: all sections are used . . . . .	548	20.17.8	Look-up table programming guidelines . . . . .	554
20.17.3	Example 3: more than one but not all sections are used . . . . .	548			

**Chapter 21: LPC178x/7x SSP interfaces**

<b>21.1</b>	<b>Basic configuration . . . . .</b>	<b>556</b>	21.6.3	SSPn Data Register (SSP0DR - 0x4008 8008, SSP1DR - 0x4003 0008, SSP2DR - 0x400A C008) . . . . .	567
<b>21.2</b>	<b>Features . . . . .</b>	<b>556</b>	21.6.4	SSPn Status Register (SSP0SR - 0x4008 800C, SSP1SR - 0x4003 000C, SSP2SR - 0x400A C00C) . . . . .	567
<b>21.3</b>	<b>Description . . . . .</b>	<b>556</b>	21.6.5	SSPn Clock Prescale Register (SSP0CPSR - 0x4008 8010, SSP1CPSR - 0x4003 0010, SSP2CPSR - 0x400A C010) . . . . .	568
<b>21.4</b>	<b>Pin descriptions . . . . .</b>	<b>557</b>	21.6.6	SSPn Interrupt Mask Set/Clear Register (SSP0IMSC - 0x4008 8014, SSP1IMSC - 0x4003 0014, SSP2IMSC - 0x400A C014) . . . . .	568
<b>21.5</b>	<b>Bus description . . . . .</b>	<b>557</b>	21.6.7	SSPn Raw Interrupt Status Register (SSP0RIS - 0x4008 8018, SSP1RIS - 0x4003 0018, SSP2RIS - 0x400A C018) . . . . .	569
21.5.1	Texas Instruments synchronous serial frame format . . . . .	557	21.6.8	SSPn Masked Interrupt Status Register (SSP0MIS - 0x4008 801C, SSP1MIS - 0x4003 001C, SSP2MIS - 0x400A C01C) . . . . .	569
21.5.2	SPI frame format . . . . .	558	21.6.9	SSPn Interrupt Clear Register (SSP0ICR - 0x4008 8020, SSP1ICR - 0x4003 0020, SSP2ICR - 0x400A C020) . . . . .	569
21.5.2.1	Clock Polarity (CPOL) and Phase (CPHA) control 558		21.6.10	SSPn DMA Control Register (SSP0DMACR - 0x4008 8024, SSP1DMACR - 0x4003 0024, SSP2DMACR - 0x400A C024) . . . . .	570
21.5.2.2	SPI format with CPOL=0,CPHA=0 . . . . .	559			
21.5.2.3	SPI format with CPOL=0,CPHA=1 . . . . .	560			
21.5.2.4	SPI format with CPOL = 1,CPHA = 0 . . . . .	560			
21.5.2.5	SPI format with CPOL = 1,CPHA = 1 . . . . .	562			
21.5.3	National Semiconductor Microwire frame format . 562				
21.5.3.1	Setup and hold time requirements on CS with respect to SK in Microwire mode . . . . .	564			
<b>21.6</b>	<b>Register description . . . . .</b>	<b>565</b>			
21.6.1	SSPn Control Register 0 (SSP0CR0 - 0x4008 8000, SSP1CR0 - 0x4003 0000, SSP2CR0 - 0x400A C000) . . . . .	566			
21.6.2	SSPn Control Register 1 (SSP0CR1 - 0x4008 8004, SSP1CR1 - 0x4003 0004, SSP2CR1 - 0x400A C004) . . . . .	566			

**Chapter 22: LPC178x/7x I2C-bus interfaces**

<b>22.1</b>	<b>Basic configuration . . . . .</b>	<b>571</b>	22.6.1	Master Transmitter mode . . . . .	574
<b>22.2</b>	<b>Features . . . . .</b>	<b>572</b>	22.6.2	Master Receiver mode . . . . .	575
<b>22.3</b>	<b>Applications . . . . .</b>	<b>572</b>	22.6.3	Slave Receiver mode . . . . .	576
<b>22.4</b>	<b>Description . . . . .</b>	<b>572</b>	22.6.4	Slave Transmitter mode . . . . .	577
22.4.1	I2C FAST Mode Plus . . . . .	573	<b>22.7</b>	<b>I2C implementation and operation . . . . .</b>	<b>578</b>
<b>22.5</b>	<b>Pin description . . . . .</b>	<b>574</b>	22.7.1	Input filters and output stages . . . . .	578
<b>22.6</b>	<b>I2C operating modes . . . . .</b>	<b>574</b>	22.7.2	Address Registers, I2ADR0 to I2ADR3 . . . . .	580

22.7.3	Address mask registers, I2MASK0 to I2MASK3. . . . .	580	22.9.4	Slave Transmitter mode . . . . .	600
22.7.4	Comparator . . . . .	580	22.9.5	Detailed state tables . . . . .	601
22.7.5	Shift register, I2DAT . . . . .	580	22.9.6	Miscellaneous states . . . . .	606
22.7.6	Arbitration and synchronization logic . . . . .	580	22.9.6.1	I2STAT = 0xF8 . . . . .	606
22.7.7	Serial clock generator . . . . .	581	22.9.6.2	I2STAT = 0x00 . . . . .	606
22.7.8	Timing and control . . . . .	582	22.9.7	Some special cases . . . . .	606
22.7.9	Control register, I2CONSET and I2CONCLR . . . . .	582	22.9.7.1	Simultaneous repeated START conditions from two masters . . . . .	606
22.7.10	Status decoder and status register . . . . .	582	22.9.7.2	Data transfer after loss of arbitration . . . . .	607
<b>22.8</b>	<b>Register description . . . . .</b>	<b>583</b>	22.9.7.3	Forced access to the I <sup>2</sup> C-bus . . . . .	607
22.8.1	I <sup>2</sup> C Control Set register (I2CONSET: I <sup>2</sup> C0, I2C0CONSET - 0x4001 C000; I <sup>2</sup> C1, I2C1CONSET - 0x4005 C000; I <sup>2</sup> C2, I2C2CONSET - 0x400A 0000) . . . . .	584	22.9.7.4	I <sup>2</sup> C-bus obstructed by a LOW level on SCL or SDA . . . . .	607
22.8.2	I <sup>2</sup> C Control Clear register (I2CONCLR: I <sup>2</sup> C0, I2C0CONCLR - 0x4001 C018; I <sup>2</sup> C1, I2C1CONCLR - 0x4005 C018; I <sup>2</sup> C2, I2C2CONCLR - 0x400A 0018) . . . . .	586	22.9.7.5	Bus error . . . . .	607
22.8.3	I <sup>2</sup> C Status register (I2STAT: I <sup>2</sup> C0, I2C0STAT - 0x4001 C004; I <sup>2</sup> C1, I2C1STAT - 0x4005 C004; I <sup>2</sup> C2, I2C2STAT - 0x400A 0004) . . . . .	587	22.9.8	I <sup>2</sup> C state service routines . . . . .	609
22.8.4	I <sup>2</sup> C Data register (I2DAT: I <sup>2</sup> C0, I2C0DAT - 0x4001 C008; I <sup>2</sup> C1, I2C1DAT - 0x4005 C008; I <sup>2</sup> C2, I2C2DAT - 0x400A 0008) . . . . .	587	22.9.8.1	Initialization . . . . .	609
22.8.5	I <sup>2</sup> C Monitor mode control register (I2MMCTRL: I <sup>2</sup> C0, I2COMMCTRL - 0x4001 C01C; I <sup>2</sup> C1, I2C1MMCTRL - 0x4005 C01C; I <sup>2</sup> C2, I2C2MMCTRL - 0x400A 001C) . . . . .	587	22.9.8.2	I <sup>2</sup> C interrupt service . . . . .	609
22.8.5.1	Interrupt in Monitor mode . . . . .	588	22.9.8.3	The state service routines . . . . .	609
22.8.5.2	Loss of arbitration in Monitor mode . . . . .	589	22.9.8.4	Adapting state services to an application . . . . .	609
22.8.6	I <sup>2</sup> C Data buffer register (I2DATA_BUFFER: I <sup>2</sup> C0, I2CDATA_BUFFER - 0x4001 C02C; I <sup>2</sup> C1, I2C1DATA_BUFFER - 0x4005 C02C; I <sup>2</sup> C2, I2C2DATA_BUFFER - 0x400A 002C) . . . . .	589	<b>22.10</b>	<b>Software example . . . . .</b>	<b>610</b>
22.8.7	I <sup>2</sup> C Slave Address registers (I2ADR0 to 3: I <sup>2</sup> C0, I2C0ADR[0, 1, 2, 3] - 0x4001 C0[0C, 20, 24, 28]; I <sup>2</sup> C1, I2C1ADR[0, 1, 2, 3] - address 0x4005 C0[0C, 20, 24, 28]; I <sup>2</sup> C2, I2C2ADR[0, 1, 2, 3] - address 0x400A 00[0C, 20, 24, 28]) . . . . .	590	22.10.1	Initialization routine . . . . .	610
22.8.8	I <sup>2</sup> C Mask registers (I2MASK0 to 3: I <sup>2</sup> C0, I2C0MASK[0, 1, 2, 3] - 0x4001 C0[30, 34, 38, 3C]; I <sup>2</sup> C1, I2C1MASK[0, 1, 2, 3] - address 0x4005 C0[30, 34, 38, 3C]; I <sup>2</sup> C2, I2C2MASK[0, 1, 2, 3] - address 0x400A 00[30, 34, 38, 3C]) . . . . .	590	22.10.2	Start Master Transmit function . . . . .	610
22.8.9	I <sup>2</sup> C SCL HIGH duty cycle register (I2SCLH: I <sup>2</sup> C0, I2C0SCLH - 0x4001 C010; I <sup>2</sup> C1, I2C1SCLH - 0x4005 C010; I <sup>2</sup> C2, I2C2SCLH - 0x400A 0010) . . . . .	591	22.10.3	Start Master Receive function . . . . .	610
22.8.10	I <sup>2</sup> C SCL Low duty cycle register (I2SCLL: I <sup>2</sup> C0 - I2C0SCLL: 0x4001 C014; I <sup>2</sup> C1 - I2C1SCLL: 0x4005 C014; I <sup>2</sup> C2 - I2C2SCLL: 0x400A 0014) . . . . .	591	22.10.4	I <sup>2</sup> C interrupt routine . . . . .	610
22.8.11	Selecting the appropriate I <sup>2</sup> C data rate and duty cycle . . . . .	591	22.10.5	Non mode specific states . . . . .	610
<b>22.9</b>	<b>Details of I<sup>2</sup>C operating modes . . . . .</b>	<b>593</b>	22.10.5.1	State: 0x00 . . . . .	610
22.9.1	Master Transmitter mode . . . . .	594	22.10.5.2	Master States . . . . .	611
22.9.2	Master Receiver mode . . . . .	596	22.10.5.3	State: 0x08 . . . . .	611
22.9.3	Slave Receiver mode . . . . .	598	22.10.5.4	State: 0x10 . . . . .	611
			22.10.6	Master Transmitter states . . . . .	611
			22.10.6.1	State: 0x18 . . . . .	611
			22.10.6.2	State: 0x20 . . . . .	612
			22.10.6.3	State: 0x28 . . . . .	612
			22.10.6.4	State: 0x30 . . . . .	612
			22.10.6.5	State: 0x38 . . . . .	612
			22.10.7	Master Receive states . . . . .	612
			22.10.7.1	State: 0x40 . . . . .	612
			22.10.7.2	State: 0x48 . . . . .	613
			22.10.7.3	State: 0x50 . . . . .	613
			22.10.7.4	State: 0x58 . . . . .	613
			22.10.8	Slave Receiver states . . . . .	613
			22.10.8.1	State: 0x60 . . . . .	613
			22.10.8.2	State: 0x68 . . . . .	614
			22.10.8.3	State: 0x70 . . . . .	614
			22.10.8.4	State: 0x78 . . . . .	614
			22.10.8.5	State: 0x80 . . . . .	614
			22.10.8.6	State: 0x88 . . . . .	615
			22.10.8.7	State: 0x90 . . . . .	615
			22.10.8.8	State: 0x98 . . . . .	615
			22.10.8.9	State: 0xA0 . . . . .	615
			22.10.9	Slave Transmitter states . . . . .	615
			22.10.9.1	State: 0xA8 . . . . .	615
			22.10.9.2	State: 0xB0 . . . . .	616
			22.10.9.3	State: 0xB8 . . . . .	616
			22.10.9.4	State: 0xC0 . . . . .	616
			22.10.9.5	State: 0xC8 . . . . .	616

**Chapter 23: LPC178x/7x I<sup>2</sup>S interface**

<b>23.1</b>	<b>Basic configuration</b> .....	<b>617</b>	<b>23.5.8</b>	Interrupt Request Control register (I2SIRQ - 0x400A 801C) .....	623
<b>23.2</b>	<b>Features</b> .....	<b>617</b>	<b>23.5.9</b>	Transmit Clock Rate register (I2STXRATE - 0x400A 8020) .....	623
<b>23.3</b>	<b>Description</b> .....	<b>618</b>	<b>23.5.9.1</b>	Notes on fractional rate generators .....	624
<b>23.4</b>	<b>Pin descriptions</b> .....	<b>619</b>	<b>23.5.10</b>	Receive Clock Rate register (I2SRXRATE - 0x400A 8024) .....	624
<b>23.5</b>	<b>Register description</b> .....	<b>620</b>	<b>23.5.11</b>	Transmit Clock Bit Rate register (I2STXBITRATE - 0x400A 8028) .....	625
23.5.1	Digital Audio Output register (I2SDAO - 0x400A 8000) .....	620	<b>23.5.12</b>	Receive Clock Bit Rate register (I2SRXBITRATE - 0x400A 802C) .....	625
23.5.2	Digital Audio Input register (I2SDAI - 0x400A 8004) .....	621	<b>23.5.13</b>	Transmit Mode Control register (I2STXMODE - 0x400A 8030) .....	625
23.5.3	Transmit FIFO register (I2STXFIFO - 0x400A 8008) .....	621	<b>23.5.14</b>	Receive Mode Control register (I2SRXMODE - 0x400A 8034) .....	626
23.5.4	Receive FIFO register (I2SRXFIFO - 0x400A 800C) .....	622	<b>23.6</b>	<b>I<sup>2</sup>S transmit and receive interfaces</b> .....	<b>627</b>
23.5.5	Status Feedback register (I2SSTATE - 0x400A 8010) .....	622	<b>23.7</b>	<b>I<sup>2</sup>S operating modes</b> .....	<b>628</b>
23.5.6	DMA Configuration Register 1 (I2SDMA1 - 0x400A 8014) .....	622	<b>23.8</b>	<b>FIFO controller</b> .....	<b>632</b>
23.5.7	DMA Configuration Register 2 (I2SDMA2 - 0x400A 8018) .....	623			

**Chapter 24: LPC178x/7x Timer0/1/2/3**

<b>24.1</b>	<b>Basic configuration</b> .....	<b>634</b>	<b>24.6.5</b>	Prescale register (T0PR - T3PR, 0x4000 400C, 0x4000 800C, 0x4009 000C, 0x4009 400C) .....	641
<b>24.2</b>	<b>Features</b> .....	<b>634</b>	<b>24.6.6</b>	Prescale Counter register (T0PC - T3PC, 0x4000 4010, 0x4000 8010, 0x4009 0010, 0x4009 4010) .....	641
<b>24.3</b>	<b>Applications</b> .....	<b>635</b>	<b>24.6.7</b>	Match Registers (MR0 - MR3) .....	642
<b>24.4</b>	<b>Description</b> .....	<b>635</b>	<b>24.6.8</b>	Match Control Register (T[0/1/2/3]MCR - 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014) .....	642
<b>24.5</b>	<b>Pin description</b> .....	<b>637</b>	<b>24.6.9</b>	Capture Registers (CR0 - CR1) .....	643
24.5.1	Multiple CAP and MAT pins .....	637	<b>24.6.10</b>	Capture Control Register (T[0/1/2/3]CCR - 0x4000 4028, 0x4000 8028, 0x4009 0028, 0x4009 4028) .....	643
<b>24.6</b>	<b>Register description</b> .....	<b>638</b>	<b>24.6.11</b>	External Match Register (T[0/1/2/3]EMR - 0x4000 403C, 0x4000 803C, 0x4009 003C, 0x4009 403C) .....	643
24.6.1	Interrupt Register (T[0/1/2/3]IR - 0x4000 4000, 0x4000 8000, 0x4009 0000, 0x4009 4000) .....	639	<b>24.6.12</b>	DMA operation .....	645
24.6.2	Timer Control Register (T[0/1/2/3]CR - 0x4000 4004, 0x4000 8004, 0x4009 0004, 0x4009 4004) .....	639	<b>24.7</b>	<b>Example timer operation</b> .....	<b>646</b>
24.6.3	Count Control Register (T[0/1/2/3]CTCR - 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) .....	640			
24.6.4	Timer Counter .....	registers (T0TC - T3TC, 0x4000 4008, 0x4000 8008, 0x4009 0008, 0x4009 4008) .....	641		

**Chapter 25: LPC178x/7x System Tick timer**

<b>25.1</b>	<b>Basic configuration</b> .....	<b>647</b>	<b>25.5.4</b>	System Timer Calibration value register (STCALIB - 0xE000 E01C) .....	649
<b>25.2</b>	<b>Features</b> .....	<b>647</b>	<b>25.6</b>	<b>Example timer calculations</b> .....	<b>651</b>
<b>3</b>	<b>Description</b> .....	<b>647</b>	Example 1) .....	651	
<b>25.4</b>	<b>Operation</b> .....	<b>647</b>	Example 2) .....	651	
<b>25.5</b>	<b>Register description</b> .....	<b>648</b>	Example 3) .....	651	
25.5.1	System Timer Control and status register (STCTRL - 0xE000 E010) .....	648	Example 4) .....	651	
25.5.2	System Timer Reload value register (STRELOAD - 0xE000 E014) .....	649			
25.5.3	System Timer Current value register (STCURRE - 0xE000 E018) .....	649			

**Chapter 26: LPC178x/7x Pulse Width Modulators (PWM0/1)**

<b>26.1</b>	<b>Basic configuration</b> . . . . .	<b>652</b>	26.6.4	PWM Timer Counter (PWM0TC - 0x4001 4008 and PWM1TC - 0x4001 8008) . . . . .	660
<b>26.2</b>	<b>Features</b> . . . . .	<b>652</b>	26.6.5	PWM Prescale Register (PWM0PR - 0x4001 400C and PWM1PR - 0x4001 800C) . . . . .	660
<b>26.3</b>	<b>Description</b> . . . . .	<b>653</b>	26.6.6	PWM Prescale Counter Register (PWM0PC - 0x4001 4010 and PWM1PC - 0x4001 8010) . . . . .	660
<b>26.4</b>	<b>Sample waveform with rules for single and double edge control.</b> . . . . .	<b>655</b>	26.6.7	PWM Match Control Register (PWM0MCR - 0x4001 4014 and PWM1MCR - 0x4001 8014) . . . . .	660
26.4.1	Rules for Single Edge Controlled PWM Outputs . . . . .	656	26.6.8	PWM Match Registers (PWM0MR0 to PWM0MR6 and PWM1MR0 to PWM1MR6) . . . . .	662
26.4.2	Rules for Double Edge Controlled PWM Outputs . . . . .	656	26.6.9	PWM Capture Control Register (PWM0CCR - 0x4001 4028 and PWM1CCR - 0x4001 8028) . . . . .	662
<b>26.5</b>	<b>Pin description</b> . . . . .	<b>656</b>	26.6.10	PWM Capture Registers (PWM0CR0 - 0x4001 402C and PWM1CR0/CR1 - 0x4001 802C/30) . . . . .	663
<b>26.6</b>	<b>Register description</b> . . . . .	<b>657</b>	26.6.11	PWM Control Registers (PWM0PCR - 0x4001 404C and PWM1PCR 0x4001 804C) . . . . .	663
26.6.1	PWM Interrupt Register (PWM0IR - 0x4001 4000 and PWM1IR - 0x4001 8000) . . . . .	658	26.6.12	PWM Latch Enable Register (PWM0LER - 0x4001 4050 and PWM1LER 0x4001 8050) . . . . .	664
26.6.2	PWM Timer Control Register (PWM0TCR - 0x4001 4004 and PWM1TCR - 0x4001 8004) . . . . .	658			
26.6.3	PWM Count Control Register (PWM0CTCR - 0x4001 4070 and PWM1CTCR - 0x4001 8070) . . . . .	659			

**Chapter 27: LPC178x/7x Motor control PWM**

<b>27.1</b>	<b>Basic configuration</b> . . . . .	<b>665</b>	27.8.3.5	MCPWM Interrupt Flags set address (MCINTF_SET - 0x400B 806C) . . . . .	676
<b>27.2</b>	<b>Introduction</b> . . . . .	<b>665</b>	27.8.3.6	MCPWM Interrupt Flags clear address (MCINTF_CLR - 0x400B 8070) . . . . .	676
<b>27.3</b>	<b>Description</b> . . . . .	<b>665</b>	27.8.4	MCPWM Count Control register . . . . .	677
<b>27.4</b>	<b>Pin description</b> . . . . .	<b>666</b>	27.8.4.1	MCPWM Count Control read address (MCCNTCON - 0x400B 805C) . . . . .	677
<b>27.5</b>	<b>Block diagram</b> . . . . .	<b>667</b>	27.8.4.2	MCPWM Count Control set address (MCCNTCON_SET - 0x400B 8060) . . . . .	678
<b>27.6</b>	<b>Configuring other modules for MCPWM use</b> . . . . .	<b>668</b>	27.8.4.3	MCPWM Count Control clear address (MCCNTCON_CLR - 0x400B 8064) . . . . .	678
<b>27.7</b>	<b>General operation</b> . . . . .	<b>668</b>	27.8.5	MCPWM Timer/Counter 0-2 registers (MCTC0-2 - 0x400B 8018, 0x400B 801C, 0x400B 8020) . . . . .	679
<b>27.8</b>	<b>Register description</b> . . . . .	<b>669</b>	27.8.6	MCPWM Limit 0-2 registers (MCLIM0-2 - 0x400B 8024, 0x400B 8028, 0x400B 802C) . . . . .	680
27.8.1	MCPWM Control register . . . . .	670	27.8.7	MCPWM Match 0-2 registers (MCMAT0-2 - 0x400B 8030, 0x400B 8034, 0x400B 8038) . . . . .	681
27.8.1.1	MCPWM Control read address (MCCON - 0x400B 8000) . . . . .	670	27.8.7.1	Match register in Edge-Aligned mode . . . . .	681
27.8.1.2	MCPWM Control set address (MCCON_SET - 0x400B 8004) . . . . .	671	27.8.7.2	Match register in Center-Aligned mode . . . . .	681
27.8.1.3	MCPWM Control clear address (MCCON_CLR - 0x400B 8008) . . . . .	672	27.8.7.3	0 and 100% duty cycle . . . . .	681
27.8.2	MCPWM Capture Control register . . . . .	673	27.8.8	MCPWM Dead-time register (MCDT - 0x400B 803C) . . . . .	682
27.8.2.1	MCPWM Capture Control read address (MCCAPCON - 0x400B 800C) . . . . .	673	27.8.9	MCPWM Commutation Pattern register (MCCP - 0x400B 8040) . . . . .	682
27.8.2.2	MCPWM Capture Control set address (MCCAPCON_SET - 0x400B 8010) . . . . .	673	27.8.10	MCPWM Capture Registers . . . . .	683
27.8.2.3	MCPWM Capture control clear address (MCCAPCON_CLR - 0x400B 8014) . . . . .	674	27.8.10.1	MCPWM Capture read addresses (MCCAP0-2 - 0x400B 8044, 0x400B 8048, 0x400B 804C) . . . . .	683
27.8.3	MCPWM Interrupt registers . . . . .	675	27.8.10.2	MCPWM Capture clear address (MCCAP_CLR - 0x400B 8074) . . . . .	683
27.8.3.1	MCPWM Interrupt Enable read address (MCINTEN - 0x400B 8050) . . . . .	675	<b>27.9</b>	<b>PWM operation</b> . . . . .	<b>684</b>
27.8.3.2	MCPWM Interrupt Enable set address (MCINTEN_SET - 0x400B 8054) . . . . .	675	27.9.1	Pulse-width modulation . . . . .	684
27.8.3.3	MCPWM Interrupt Enable clear address (MCINTEN_CLR - 0x400B 8058) . . . . .	676		Edge-aligned PWM without dead-time. . . . .	684
27.8.3.4	MCPWM Interrupt Flags read address (MCINTF - 0x400B 8068) . . . . .	676			



Center-aligned PWM without dead-time . . . . .	684	27.9.5	External event counting (Counter mode) . . . . .	687
Dead-time counter . . . . .	685	27.9.6	Three-phase DC mode . . . . .	687
27.9.2 Shadow registers and simultaneous updates	686	27.9.7	Three phase AC mode . . . . .	688
27.9.3 Fast Abort (ABORT) . . . . .	686	27.9.8	Interrupts . . . . .	689
27.9.4 Capture events . . . . .	686			

**Chapter 28: LPC178x/7x Quadrature Encoder Interface (QEI)**

<b>28.1 Basic configuration . . . . .</b>	<b>690</b>	28.6.3.8	QEI Index Compare register 1 (INXCMP1 - 0x400B C04C) . . . . .	699
<b>28.2 Features . . . . .</b>	<b>690</b>	28.6.3.9	QEI Index Compare register 2 (INXCMP2 - 0x400B C050) . . . . .	700
<b>28.3 Introduction . . . . .</b>	<b>691</b>	28.6.3.10	QEI Velocity Timer Reload register (QEILOAD - 0x400B C028) . . . . .	700
<b>28.4 Functional description . . . . .</b>	<b>692</b>	28.6.3.11	QEI Velocity Timer register (QEITIME - 0x400B C02C) . . . . .	700
28.4.1 Input signals . . . . .	692	28.6.3.12	QEI Velocity register (QEIVEL - 0x400B C030) . . . . .	700
28.4.1.1 Quadrature input signals . . . . .	692	28.6.3.13	QEI Velocity Capture register (QEICAP - 0x400B C034) . . . . .	700
28.4.1.2 Digital input filtering . . . . .	693	28.6.3.14	QEI Velocity Compare register (VELCOMP - 0x400B C038) . . . . .	701
28.4.2 Position capture . . . . .	693	28.6.3.15	QEI Digital Filter on PHA (FILTERPHA - 0x400B C03C) . . . . .	701
28.4.3 Velocity capture . . . . .	694	28.6.3.16	QEI Digital Filter on PHB (FILTERPHB - 0x400B C040) . . . . .	701
28.4.4 Velocity compare . . . . .	694	28.6.3.17	QEI Digital Filter on INX (FILTERINX - 0x400B C044) . . . . .	701
<b>28.5 Pin description . . . . .</b>	<b>695</b>	28.6.3.18	QEI index acceptance Window (WINDOW - 0x400B C048) . . . . .	701
<b>28.6 Register description . . . . .</b>	<b>696</b>	28.6.4	Interrupt registers . . . . .	702
28.6.1 Register summary . . . . .	696	28.6.4.1	QEI Interrupt Status register (QEIINTSTAT - 0x400B CFE0) . . . . .	702
28.6.2 Control registers . . . . .	697	28.6.4.2	QEI Interrupt Set register (QEISET - 0x400B CFEC) . . . . .	703
28.6.2.1 QEI Control register (QEICON - 0x400B C000) . . . . .	697	28.6.4.3	QEI Interrupt Clear register (QEICLR - 0x400B CFE8) . . . . .	704
28.6.2.2 QEI Configuration register (QEICONF - 0x400B C008) . . . . .	697	28.6.4.4	QEI Interrupt Enable register (QEIIE - 0x400B CFE4) . . . . .	705
28.6.2.3 QEI Status register (QEISTAT - 0x400B C004) . . . . .	698	28.6.4.5	QEI Interrupt Enable Set register (QEIIES - 0x400B CFDC) . . . . .	706
28.6.3 Position, index and timer registers . . . . .	698	28.6.4.6	QEI Interrupt Enable Clear register (QEIEEC - 0x400B CFD8) . . . . .	707
28.6.3.1 QEI Position register (QEIPOS - 0x400B C00C) . . . . .	698			
28.6.3.2 QEI Maximum Position register (QEIMAXPOS - 0x400B C010) . . . . .	698			
28.6.3.3 QEI Position Compare register 0 (CMPOS0 - 0x400B C014) . . . . .	698			
28.6.3.4 QEI Position Compare register 1 (CMPOS1 - 0x400B C018) . . . . .	699			
28.6.3.5 QEI Position Compare register 2 (CMPOS2 - 0x400B C01C) . . . . .	699			
28.6.3.6 QEI Index Count register (INXCNT - 0x400B C020) . . . . .	699			
28.6.3.7 QEI Index Compare register 0 (INXCMP0 - 0x400B C024) . . . . .	699			

**Chapter 29: LPC178x/7x Real Time Clock (RTC)**

<b>29.1 Basic configuration . . . . .</b>	<b>708</b>	29.6.2.3	Counter Increment Interrupt Register (CIIR - 0x4002 400C) . . . . .	713
<b>29.2 Features . . . . .</b>	<b>708</b>	29.6.2.4	Alarm Mask Register (AMR - 0x4002 4010) . . . . .	713
<b>29.3 Description . . . . .</b>	<b>708</b>	29.6.2.5	RTC Auxiliary control register (RTC_AUX - 0x4002 405C) . . . . .	714
<b>29.4 Architecture . . . . .</b>	<b>709</b>	29.6.2.6	RTC Auxiliary Enable register (RTC_AUXEN - 0x4002 4058) . . . . .	714
<b>29.5 Pin description . . . . .</b>	<b>710</b>	29.6.3	Consolidated time registers . . . . .	715
<b>29.6 Register description . . . . .</b>	<b>710</b>	29.6.3.1	Consolidated Time Register 0 (CTIME0 - 0x4002 4014) . . . . .	715
29.6.1 RTC interrupts . . . . .	712	29.6.3.2	Consolidated Time Register 1 (CTIME1 - 0x4002 4018) . . . . .	715
29.6.2 Miscellaneous register group . . . . .	712			
29.6.2.1 Interrupt Location Register (ILR - 0x4002 4000) . . . . .	712			
29.6.2.2 Clock Control Register (CCR - 0x4002 4008) . . . . .	712			

29.6.3.3	Consolidated Time Register 2 (CTIME2 - 0x4002 401C) . . . . .	715	29.6.6	Forward calibration . . . . .	717
29.6.4	Time Counter Group . . . . .	716	29.6.6.1	General purpose registers . . . . .	718
29.6.4.1	Leap year calculation . . . . .	716		General purpose registers 0 to 4 (GPREG0 to GPREG4 - addresses 0x4002 4044 to 0x4002 4054) . . . . .	718
29.6.4.2	Calibration register (CALIBRATION - address 0x4002 4040) . . . . .	716	29.6.7	Alarm register group . . . . .	718
29.6.5	Calibration procedure . . . . .	717	<b>29.7</b>	<b>RTC usage notes . . . . .</b>	<b>718</b>
	Backward calibration . . . . .	717			

**Chapter 30: LPC178x/7x Windowed Watchdog Timer (WWDT)**

<b>30.1</b>	<b>Features . . . . .</b>	<b>719</b>	30.4.3	Watchdog Feed register (WDFEED - 0x4000 0008) . . . . .	723
<b>30.2</b>	<b>Applications . . . . .</b>	<b>719</b>	30.4.4	Watchdog Timer Value register (WDTV - 0x4000 000C) . . . . .	724
<b>30.3</b>	<b>Description . . . . .</b>	<b>720</b>	30.4.5	Watchdog Timer Warning Interrupt register (WDWARNINT - 0x4000 0014) . . . . .	724
<b>30.4</b>	<b>Register description . . . . .</b>	<b>722</b>	30.4.6	Watchdog Timer Window register (WDWINDOW - 0x4000 0018) . . . . .	724
30.4.1	Watchdog Mode register (WDMOD - 0x4000 0000) . . . . .	722	<b>30.5</b>	<b>Watchdog timing examples . . . . .</b>	<b>725</b>
30.4.2	Watchdog Timer Constant register (WDTC - base + -x04) . . . . .	723			

**Chapter 31: LPC178x/7x Analog-to-Digital Converter (ADC)**

<b>31.1</b>	<b>Basic configuration . . . . .</b>	<b>727</b>	31.5.3	A/D Interrupt Enable register (AD0INTEN - 0x4003 400C) . . . . .	731
<b>31.2</b>	<b>Features . . . . .</b>	<b>727</b>	31.5.4	A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C) . . . . .	732
<b>31.3</b>	<b>Description . . . . .</b>	<b>728</b>	31.5.5	A/D Status register (ADSTAT - 0x4003 4030) . . . . .	732
<b>31.4</b>	<b>Pin description . . . . .</b>	<b>728</b>	31.5.6	A/D Trim register (ADTRIM - 0x4003 4034) . . . . .	733
<b>31.5</b>	<b>Register description . . . . .</b>	<b>729</b>	<b>31.6</b>	<b>Operation . . . . .</b>	<b>733</b>
31.5.1	A/D Control Register (AD0CR - 0x4003 4000) . . . . .	730	31.6.1	Hardware-triggered conversion . . . . .	734
31.5.2	A/D Global Data Register (AD0GDR - 0x4003 4004) . . . . .	730	31.6.2	Interrupts . . . . .	734
			31.6.3	Accuracy vs. digital receiver . . . . .	734
			31.6.4	DMA control . . . . .	734

**Chapter 32: LPC178x/7x Digital-to-Analog Converter (DAC)**

<b>32.1</b>	<b>Basic configuration . . . . .</b>	<b>735</b>	32.5.2	D/A Converter Control register (DACCTRL - 0x4008 C004) . . . . .	737
<b>32.2</b>	<b>Features . . . . .</b>	<b>735</b>	32.5.3	D/A Converter Counter Value register (DACCNTVAL - 0x4008 C008) . . . . .	738
<b>32.3</b>	<b>Architecture . . . . .</b>	<b>736</b>	<b>32.6</b>	<b>Operation . . . . .</b>	<b>739</b>
<b>32.4</b>	<b>Pin description . . . . .</b>	<b>736</b>	32.6.1	DMA counter . . . . .	739
<b>32.5</b>	<b>Register description . . . . .</b>	<b>737</b>	32.6.2	Double buffering . . . . .	739
32.5.1	D/A Converter Register (DACR - 0x4008 C000) . . . . .	737			

**Chapter 33: LPC178x/7x General Purpose DMA controller**

<b>33.1</b>	<b>Basic configuration . . . . .</b>	<b>740</b>	33.4.1.6.1	Bus and transfer widths . . . . .	742
<b>33.2</b>	<b>Introduction . . . . .</b>	<b>740</b>	33.4.1.6.2	Endian behavior . . . . .	742
<b>33.3</b>	<b>Features . . . . .</b>	<b>740</b>	33.4.1.6.3	Error conditions . . . . .	744
<b>33.4</b>	<b>Functional description . . . . .</b>	<b>741</b>	33.4.1.7	Channel hardware . . . . .	745
33.4.1	DMA controller functional description . . . . .	741	33.4.1.8	DMA request priority . . . . .	745
33.4.1.1	AHB slave interface . . . . .	741	33.4.1.9	Interrupt generation . . . . .	745
33.4.1.2	Control logic and register bank . . . . .	742	33.4.2	DMA system connections . . . . .	745
33.4.1.3	DMA request and response interface . . . . .	742	33.4.2.1	DMA request signals . . . . .	745
33.4.1.4	Channel logic and channel register bank . . . . .	742	33.4.2.2	DMA response signals . . . . .	745
33.4.1.5	Interrupt request . . . . .	742	33.4.2.3	DMA request connections . . . . .	746
33.4.1.6	AHB master interface . . . . .	742	<b>33.5</b>	<b>Register description . . . . .</b>	<b>747</b>

33.5.1	DMA Interrupt Status register (DMACIntStat - 0x2008 0000) . . . . .	749	33.5.19	DMA Channel Linked List Item registers (DMACCxLLI - 0x2008 01x8) . . . . .	756
33.5.2	DMA Interrupt Terminal Count Request Status register (DMACIntTCStat - 0x2008 0004) . . . . .	749	33.5.20	DMA channel control registers (DMACCxControl - 0x2008 01xC) . . . . .	756
33.5.3	DMA Interrupt Terminal Count Request Clear register (DMACIntTCClear - 0x2008 0008) . . . . .	749	33.5.20.1	Protection and access information . . . . .	756
33.5.4	DMA Interrupt Error Status register (DMACIntErrStat - 0x2008 000C) . . . . .	749	33.5.21	DMA Channel Configuration registers (DMACCxConfig - 0x2008 01x0) . . . . .	758
33.5.5	DMA Interrupt Error Clear register (DMACIntErrClr - 0x2008 0010) . . . . .	750	33.5.21.1	Lock control . . . . .	759
33.5.6	DMA Raw Interrupt Terminal Count Status register (DMACRawIntTCStat - 0x2008 0014) . . . . .	750	33.5.21.2	Transfer type . . . . .	760
33.5.7	DMA Raw Error Interrupt Status register (DMACRawIntErrStat - 0x2008 0018) . . . . .	750	<b>33.6</b>	<b>Using the DMA controller . . . . .</b>	<b>761</b>
33.5.8	DMA Enabled Channel register (DMACEnbldChns - 0x2008 001C) . . . . .	751	33.6.1	Programming the DMA controller . . . . .	761
33.5.9	DMA Software Burst Request register (DMACSoftBReq - 0x2008 0020) . . . . .	751	33.6.1.1	Enabling the DMA controller . . . . .	761
33.5.10	DMA Software Single Request register (DMACSoftSReq - 0x2008 0024) . . . . .	751	33.6.1.2	Disabling the DMA controller . . . . .	761
33.5.11	DMA Software Last Burst Request register (DMACSoftLBReq - 0x2008 0028) . . . . .	752	33.6.1.3	Enabling a DMA channel . . . . .	761
33.5.12	DMA Software Last Single Request register (DMACSoftLSReq - 0x2008 002C) . . . . .	752	33.6.1.4	Disabling a DMA channel . . . . .	761
33.5.13	DMA Configuration register (DMACConfig - 0x2008 0030) . . . . .	753		Disabling a DMA channel and losing data in the FIFO . . . . .	761
33.5.14	DMA Synchronization register (DMACSync - 0x2008 0034) . . . . .	753		Disabling the DMA channel without losing data in the FIFO . . . . .	761
33.5.15	DMA Request Select register (DMACReqSel - 0x400F C1C4) . . . . .	754	33.6.1.5	Setting up a new DMA transfer . . . . .	761
33.5.15.1	Timer DMA requests . . . . .	754	33.6.1.6	Halting a DMA channel . . . . .	762
33.5.16	DMA Channel registers . . . . .	755	33.6.1.7	Programming a DMA channel . . . . .	762
33.5.17	DMA Channel Source Address registers (DMACCxSrcAddr - 0x2008 01x0) . . . . .	755	33.6.2	Flow control . . . . .	762
33.5.18	DMA Channel Destination Address registers (DMACCxDestAddr - 0x2008 01x4) . . . . .	755	33.6.2.1	Peripheral-to-memory or memory-to-peripheral DMA flow . . . . .	763
			33.6.2.2	Peripheral-to-peripheral DMA flow . . . . .	763
			33.6.2.3	Memory-to-memory DMA flow . . . . .	764
			33.6.3	Interrupt requests . . . . .	765
			33.6.3.1	Hardware interrupt sequence flow . . . . .	765
			33.6.4	Address generation . . . . .	765
			33.6.4.1	Word-aligned transfers across a boundary . . . . .	765
			33.6.5	Scatter/gather . . . . .	766
			33.6.5.1	Linked list items . . . . .	766
			33.6.5.1.1	Programming the DMA controller for scatter/gather DMA . . . . .	766
			33.6.5.1.2	Example of scatter/gather DMA . . . . .	767

**Chapter 34: LPC178x/7x CRC engine**

<b>34.1</b>	<b>Introduction . . . . .</b>	<b>769</b>	34.4.3	CRC checksum register (CRC_SUM - 0x2009 0008) . . . . .	771
<b>34.2</b>	<b>Features . . . . .</b>	<b>769</b>	34.4.4	CRC data register (CRC_DATA - 0x2009 0008) . . . . .	772
<b>34.3</b>	<b>Description . . . . .</b>	<b>770</b>	<b>34.5</b>	<b>Functional description . . . . .</b>	<b>773</b>
<b>34.4</b>	<b>Register description . . . . .</b>	<b>771</b>		CRC-CCITT set-up . . . . .	773
34.4.1	CRC mode register (CRC_MODE - 0x2009 0000) . . . . .	771		CRC-16 set-up . . . . .	773
34.4.2	CRC seed register (CRC_SEED - 0x2009 0004) . . . . .	771		CRC-32 set-up . . . . .	773

**Chapter 35: LPC178x/7x EEPROM memory**

<b>35.1</b>	<b>Basic configuration . . . . .</b>	<b>774</b>	35.4.1.2	EEPROM address register (EEADDR - 0x0020 0084) . . . . .	776
<b>35.2</b>	<b>Description . . . . .</b>	<b>774</b>	35.4.1.3	EEPROM write data register (EEWDATA - 0x0020 0088) . . . . .	776
<b>35.3</b>	<b>Features . . . . .</b>	<b>774</b>	35.4.1.4	EEPROM read data register (EERDATA - 0x0020 008C) . . . . .	777
<b>35.4</b>	<b>Register description . . . . .</b>	<b>775</b>	35.4.1.5	EEPROM wait state register (EEWSTATE - 0x0020 0090) . . . . .	777
35.4.1	EEPROM control registers . . . . .	776			
35.4.1.1	EEPROM command register (EECMD - 0x0020 0080) . . . . .	776			

35.4.1.6	EEPROM clock divider register (EECLKDIV - 0x0020 0094) . . . . .	778	35.4.2.4	Interrupt status register (EEINTSTAT - 0x0020 0FE0) . . . . .	780
35.4.1.7	EEPROM power down register (EEPWRDWN - 0x0020 0098) . . . . .	778	35.4.2.5	Interrupt status clear register (0x0020 0FE8) . . . . .	781
35.4.2	Interrupt registers . . . . .	779	35.4.2.6	Interrupt status set (EEINTSTATSET - 0x0020 0FEC) . . . . .	781
35.4.2.1	Interrupt enable register (EEINTEN - 0x0020 0FE4) . . . . .	779	<b>35.5</b>	<b>EEPROM operation . . . . .</b>	<b>782</b>
35.4.2.2	Interrupt enable clear register (EEINTENCLR, 0x0020 0FD8) . . . . .	779	35.5.1	EEPROM device description . . . . .	782
35.4.2.3	Interrupt enable set register (EEINTENSET - 0x0020 0FDC) . . . . .	780	35.5.2	EEPROM operations . . . . .	782
			35.5.2.1	Writing . . . . .	782
			35.5.2.2	Programming . . . . .	784
			35.5.2.3	Reading . . . . .	784
			35.5.2.4	Error responses . . . . .	785

**Chapter 36: LPC178x/7x Flash memory**

<b>36.1</b>	<b>Introduction . . . . .</b>	<b>786</b>	36.7.10	Blank check sector(s) <sector number> <end sector number> . . . . .	799
<b>36.2</b>	<b>Features . . . . .</b>	<b>786</b>	36.7.11	Read Part Identification number . . . . .	799
<b>36.3</b>	<b>Description . . . . .</b>	<b>786</b>	36.7.12	Read Boot Code version number. . . . .	800
36.3.1	Memory map after any reset. . . . .	787	36.7.13	Read device serial number . . . . .	800
36.3.1.1	Criterion for Valid User Code . . . . .	787	36.7.14	Compare <address1> <address2> <no of bytes> . . . . .	800
36.3.2	Communication protocol. . . . .	788	36.7.15	ISP Return Codes . . . . .	801
36.3.2.1	ISP command format . . . . .	788	<b>36.8</b>	<b>IAP commands . . . . .</b>	<b>802</b>
36.3.2.2	ISP response format. . . . .	788	36.8.1	Prepare sector(s) for write operation . . . . .	804
36.3.2.3	ISP data format. . . . .	788	36.8.2	Copy RAM to Flash . . . . .	804
36.3.2.4	ISP flow control. . . . .	789	36.8.3	Erase Sector(s) . . . . .	805
36.3.2.5	ISP command abort . . . . .	789	36.8.4	Blank check sector(s). . . . .	805
36.3.2.6	Interrupts during IAP. . . . .	789	36.8.5	Read part identification number. . . . .	805
36.3.2.7	RAM used by ISP command handler . . . . .	789	36.8.6	Read Boot Code version number. . . . .	806
36.3.2.8	RAM used by Boot process prior to entering user program . . . . .	789	36.8.7	Read device serial number . . . . .	806
36.3.2.9	RAM used by IAP command handler . . . . .	789	36.8.8	Compare <address1> <address2> <no of bytes> . . . . .	806
<b>36.4</b>	<b>Boot process flowchart. . . . .</b>	<b>790</b>	36.8.9	Re-invoke ISP . . . . .	807
<b>36.5</b>	<b>Sector numbers . . . . .</b>	<b>791</b>	36.8.10	IAP Status Codes. . . . .	807
<b>36.6</b>	<b>Code Read Protection (CRP) . . . . .</b>	<b>792</b>	<b>36.9</b>	<b>JTAG flash programming interface . . . . .</b>	<b>807</b>
<b>36.7</b>	<b>ISP commands . . . . .</b>	<b>794</b>	<b>36.10</b>	<b>Flash signature generation . . . . .</b>	<b>808</b>
36.7.1	Unlock <Unlock code> . . . . .	794	36.10.1	Register description for signature generation . . . . .	808
36.7.2	Set Baud Rate <Baud Rate> <stop bit> . . . . .	795	36.10.1.1	Signature generation address and control registers . . . . .	809
36.7.3	Echo <setting> . . . . .	796	36.10.1.2	Signature generation result registers. . . . .	809
36.7.4	Write to RAM <start address> <number of bytes> . . . . .	796	36.10.1.3	Flash Module Status register (FMSTAT - 0x0x0020 0FE0). . . . .	810
36.7.5	Read Memory <address> <no. of bytes> . . . . .	796	36.10.1.4	Flash Module Status Clear register (FMSTATCLR - 0x0x0020 0FE8) . . . . .	810
36.7.6	Prepare sector(s) for write operation <start sector number> <end sector number> . . . . .	797	36.10.2	Algorithm and procedure for signature generation . . . . .	811
36.7.7	Copy RAM to Flash <flash address> <RAM address> <no of bytes> . . . . .	798		Signature generation . . . . .	811
36.7.8	Go <address> <mode>. . . . .	798		Content verification . . . . .	811
36.7.9	Erase sector(s) <start sector number> <end sector number>. . . . .	799			

**Chapter 37: LPC178x/7x JTAG, Serial Wire Debug, and Trace**

<b>37.1</b>	<b>Features . . . . .</b>	<b>812</b>	<b>37.6</b>	<b>JTAG TAP Identification . . . . .</b>	<b>815</b>
<b>37.2</b>	<b>Introduction . . . . .</b>	<b>812</b>	<b>37.7</b>	<b>Debug Notes . . . . .</b>	<b>816</b>
<b>37.3</b>	<b>Description . . . . .</b>	<b>812</b>	<b>37.8</b>	<b>Debug memory re-mapping . . . . .</b>	<b>816</b>
<b>37.4</b>	<b>Pin description. . . . .</b>	<b>813</b>	37.8.1	Memory Mapping Control register (MEMMAP - 0x400F C040) . . . . .	816
<b>37.5</b>	<b>Debug connections . . . . .</b>	<b>814</b>			

Chapter 38: ARM Cortex-M3 Appendix

<b>38.1</b>	<b>ARM Cortex-M3 User Guide: Introduction . . .</b>	<b>817</b>		
38.1.1	About the processor and core peripherals . . .	817	38.2.4.4.2	Operation . . . . .
38.1.1.1	System level interface . . . . .	818	38.2.4.4.3	Restrictions . . . . .
38.1.1.2	Integrated configurable debug . . . . .	818	38.2.4.4.4	Condition flags . . . . .
38.1.1.3	Cortex-M3 processor features and benefits summary . . . . .	819	38.2.4.4.5	Examples . . . . .
38.1.1.4	Cortex-M3 core peripherals . . . . .	819	38.2.4.5	LDR, PC-relative . . . . .
<b>38.2</b>	<b>ARM Cortex-M3 User Guide: Instruction Set</b>	<b>820</b>	38.2.4.5.1	Syntax . . . . .
38.2.1	Instruction set summary . . . . .	820	38.2.4.5.2	Operation . . . . .
	Note . . . . .	820	38.2.4.5.3	Restrictions . . . . .
38.2.2	Intrinsic functions . . . . .	823	38.2.4.5.4	Condition flags . . . . .
38.2.3	About the instruction descriptions . . . . .	823	38.2.4.5.5	Examples . . . . .
38.2.3.1	Operands . . . . .	824	38.2.4.6	LDM and STM . . . . .
38.2.3.2	Restrictions when using PC or SP . . . . .	824	38.2.4.6.1	Syntax . . . . .
38.2.3.3	Flexible second operand . . . . .	824	38.2.4.6.2	Operation . . . . .
38.2.3.3.1	Constant . . . . .	824	38.2.4.6.3	Restrictions . . . . .
38.2.3.3.2	Register with optional shift . . . . .	825	38.2.4.6.4	Condition flags . . . . .
38.2.3.4	Shift Operations . . . . .	825	38.2.4.6.5	Examples . . . . .
38.2.3.4.1	ASR . . . . .	826	38.2.4.6.6	Incorrect examples . . . . .
	Note . . . . .	826	38.2.4.7	PUSH and POP . . . . .
38.2.3.4.2	LSR . . . . .	826	38.2.4.7.1	Syntax . . . . .
	Note . . . . .	826	38.2.4.7.2	Operation . . . . .
38.2.3.4.3	LSL . . . . .	827	38.2.4.7.3	Restrictions . . . . .
	Note . . . . .	827	38.2.4.7.4	Condition flags . . . . .
38.2.3.4.4	ROR . . . . .	827	38.2.4.7.5	Examples . . . . .
	Note . . . . .	827	38.2.4.8	LDREX and STREX . . . . .
38.2.3.4.5	RRX . . . . .	828	38.2.4.8.1	Syntax . . . . .
38.2.3.5	Address alignment . . . . .	828	38.2.4.8.2	Operation . . . . .
38.2.3.6	PC-relative expressions . . . . .	829	38.2.4.8.3	Restrictions . . . . .
	Note . . . . .	829	38.2.4.8.4	Condition flags . . . . .
38.2.3.7	Conditional execution . . . . .	829	38.2.4.8.5	Examples . . . . .
38.2.3.7.1	The condition flags . . . . .	830	38.2.4.9	CLREX . . . . .
38.2.3.7.2	Condition code suffixes . . . . .	830	38.2.4.9.1	Syntax . . . . .
38.2.3.8	Instruction width selection . . . . .	831	38.2.4.9.2	Operation . . . . .
38.2.3.8.1	Example: Instruction width selection . . . . .	832	38.2.4.9.3	Condition flags . . . . .
38.2.4	Memory access instructions . . . . .	833	38.2.4.9.4	Examples . . . . .
38.2.4.1	ADR . . . . .	834	38.2.5	General data processing instructions . . . . .
38.2.4.1.1	Syntax . . . . .	834	38.2.5.1	ADD, ADC, SUB, SBC, and RSB . . . . .
38.2.4.1.2	Operation . . . . .	834	38.2.5.1.1	Syntax . . . . .
38.2.4.1.3	Restrictions . . . . .	834	38.2.5.1.2	Operation . . . . .
38.2.4.1.4	Condition flags . . . . .	834	38.2.5.1.3	Restrictions . . . . .
38.2.4.1.5	Examples . . . . .	834		Note . . . . .
38.2.4.2	LDR and STR, immediate offset . . . . .	835	38.2.5.1.4	Condition flags . . . . .
38.2.4.2.1	Syntax . . . . .	835	38.2.5.1.5	Examples . . . . .
38.2.4.2.2	Operation . . . . .	835	38.2.5.1.6	Multiword arithmetic examples . . . . .
38.2.4.2.3	Restrictions . . . . .	836	38.2.5.2	AND, ORR, EOR, BIC, and ORN . . . . .
38.2.4.2.4	Condition flags . . . . .	836	38.2.5.2.1	Syntax . . . . .
38.2.4.2.5	Examples . . . . .	837	38.2.5.2.2	Operation . . . . .
38.2.4.3	LDR and STR, register offset . . . . .	838	38.2.5.2.3	Restrictions . . . . .
38.2.4.3.1	Syntax . . . . .	838	38.2.5.2.4	Condition flags . . . . .
38.2.4.3.2	Operation . . . . .	838	38.2.5.2.5	Examples . . . . .
38.2.4.3.3	Restrictions . . . . .	838	38.2.5.3	ASR, LSL, LSR, ROR, and RRX . . . . .
38.2.4.3.4	Condition flags . . . . .	839	38.2.5.3.1	Syntax . . . . .
38.2.4.3.5	Examples . . . . .	839	38.2.5.3.2	Operation . . . . .
38.2.4.4	LDR and STR, unprivileged . . . . .	840	38.2.5.3.3	Restrictions . . . . .
38.2.4.4.1	Syntax . . . . .	840	38.2.5.3.4	Condition flags . . . . .
			38.2.5.3.5	Examples . . . . .
			38.2.5.4	CLZ . . . . .

38.2.5.4.1 Syntax . . . . .	858	38.2.7.1.3 Restrictions . . . . .	872
38.2.5.4.2 Operation . . . . .	858	38.2.7.1.4 Condition flags . . . . .	872
38.2.5.4.3 Restrictions . . . . .	858	38.2.7.1.5 Examples . . . . .	872
38.2.5.4.4 Condition flags . . . . .	858	38.2.8 Bitfield instructions . . . . .	873
38.2.5.4.5 Examples . . . . .	858	38.2.8.1 BFC and BFI . . . . .	874
38.2.5.5 CMP and CMN . . . . .	859	38.2.8.1.1 Syntax . . . . .	874
38.2.5.5.1 Syntax . . . . .	859	38.2.8.1.2 Operation . . . . .	874
38.2.5.5.2 Operation . . . . .	859	38.2.8.1.3 Restrictions . . . . .	874
38.2.5.5.3 Restrictions . . . . .	859	38.2.8.1.4 Condition flags . . . . .	874
38.2.5.5.4 Condition flags . . . . .	859	38.2.8.1.5 Examples . . . . .	874
38.2.5.5.5 Examples . . . . .	859	38.2.8.2 SBFX and UBFX . . . . .	875
38.2.5.6 MOV and MVN . . . . .	860	38.2.8.2.1 Syntax . . . . .	875
38.2.5.6.1 Syntax . . . . .	860	38.2.8.2.2 Operation . . . . .	875
38.2.5.6.2 Operation . . . . .	860	38.2.8.2.3 Restrictions . . . . .	875
38.2.5.6.3 Restrictions . . . . .	861	38.2.8.2.4 Condition flags . . . . .	875
38.2.5.6.4 Condition flags . . . . .	861	38.2.8.2.5 Examples . . . . .	875
38.2.5.6.5 Example . . . . .	861	38.2.8.3 SXT and UXT . . . . .	876
38.2.5.7 MOVT . . . . .	862	38.2.8.3.1 Syntax . . . . .	876
38.2.5.7.1 Syntax . . . . .	862	38.2.8.3.2 Operation . . . . .	876
38.2.5.7.2 Operation . . . . .	862	38.2.8.3.3 Restrictions . . . . .	876
38.2.5.7.3 Restrictions . . . . .	862	38.2.8.3.4 Condition flags . . . . .	876
38.2.5.7.4 Condition flags . . . . .	862	38.2.8.3.5 Examples . . . . .	877
38.2.5.7.5 Examples . . . . .	862	38.2.9 Branch and control instructions . . . . .	878
38.2.5.8 REV, REV16, REVSH, and RBIT . . . . .	863	38.2.9.1 B, BL, BX, and BLX . . . . .	879
38.2.5.8.1 Syntax . . . . .	863	38.2.9.1.1 Syntax . . . . .	879
38.2.5.8.2 Operation . . . . .	863	38.2.9.1.2 Operation . . . . .	879
38.2.5.8.3 Restrictions . . . . .	863	38.2.9.1.3 Restrictions . . . . .	880
38.2.5.8.4 Condition flags . . . . .	863	38.2.9.1.4 Condition flags . . . . .	880
38.2.5.8.5 Examples . . . . .	863	38.2.9.1.5 Examples . . . . .	880
38.2.5.9 TST and TEQ . . . . .	864	38.2.9.2 CBZ and CBNZ . . . . .	881
38.2.5.9.1 Syntax . . . . .	864	38.2.9.2.1 Syntax . . . . .	881
38.2.5.9.2 Operation . . . . .	864	38.2.9.2.2 Operation . . . . .	881
38.2.5.9.3 Restrictions . . . . .	864	38.2.9.2.3 Restrictions . . . . .	881
38.2.5.9.4 Condition flags . . . . .	864	38.2.9.2.4 Condition flags . . . . .	881
38.2.5.9.5 Examples . . . . .	864	38.2.9.2.5 Examples . . . . .	881
38.2.6 Multiply and divide instructions . . . . .	865	38.2.9.3 IT . . . . .	882
38.2.6.1 MUL, MLA, and MLS . . . . .	866	38.2.9.3.1 Syntax . . . . .	882
38.2.6.1.1 Syntax . . . . .	866	38.2.9.3.2 Operation . . . . .	882
38.2.6.1.2 Operation . . . . .	866	38.2.9.3.3 Restrictions . . . . .	882
38.2.6.1.3 Restrictions . . . . .	866	38.2.9.3.4 Condition flags . . . . .	883
38.2.6.1.4 Condition flags . . . . .	866	38.2.9.3.5 Example . . . . .	883
38.2.6.1.5 Examples . . . . .	867	38.2.9.4 TBB and TBH . . . . .	885
38.2.6.2 UMULL, UMLAL, SMULL, and SMLAL . . . . .	868	38.2.9.4.1 Syntax . . . . .	885
38.2.6.2.1 Syntax . . . . .	868	38.2.9.4.2 Operation . . . . .	885
38.2.6.2.2 Operation . . . . .	868	38.2.9.4.3 Restrictions . . . . .	885
38.2.6.2.3 Restrictions . . . . .	868	38.2.9.4.4 Condition flags . . . . .	885
38.2.6.2.4 Condition flags . . . . .	868	38.2.9.4.5 Examples . . . . .	885
38.2.6.2.5 Examples . . . . .	869	38.2.10 Miscellaneous instructions . . . . .	887
38.2.6.3 SDIV and UDIV . . . . .	870	38.2.10.1 BKPT . . . . .	888
38.2.6.3.1 Syntax . . . . .	870	38.2.10.1.1 Syntax . . . . .	888
38.2.6.3.2 Operation . . . . .	870	38.2.10.1.2 Operation . . . . .	888
38.2.6.3.3 Restrictions . . . . .	870	38.2.10.1.3 Condition flags . . . . .	888
38.2.6.3.4 Condition flags . . . . .	870	38.2.10.1.4 Examples . . . . .	888
38.2.6.3.5 Examples . . . . .	870	38.2.10.2 CPS . . . . .	889
38.2.7 Saturating instructions . . . . .	871	38.2.10.2.1 Syntax . . . . .	889
38.2.7.1 SSAT and USAT . . . . .	871	38.2.10.2.2 Operation . . . . .	889
38.2.7.1.1 Syntax . . . . .	871	38.2.10.2.3 Restrictions . . . . .	889
38.2.7.1.2 Operation . . . . .	871	38.2.10.2.4 Condition flags . . . . .	889

38.2.10.2.5 Examples . . . . .	889	38.3.1.2 Stacks . . . . .	900
38.2.10.3 DMB . . . . .	890	38.3.1.3 Core registers . . . . .	901
38.2.10.3.1 Syntax. . . . .	890	38.3.1.3.1 General-purpose registers . . . . .	902
38.2.10.3.2 Operation . . . . .	890	38.3.1.3.2 Stack Pointer . . . . .	902
38.2.10.3.3 Condition flags . . . . .	890	38.3.1.3.3 . . . . . Link Register	902
38.2.10.3.4 Examples . . . . .	890	38.3.1.3.4 Program Counter . . . . .	902
38.2.10.4 DSB . . . . .	891	38.3.1.3.5 Program Status Register . . . . .	902
38.2.10.4.1 Syntax. . . . .	891	38.3.1.3.6 Exception mask registers. . . . .	906
38.2.10.4.2 Operation . . . . .	891	38.3.1.3.7 CONTROL register . . . . .	907
38.2.10.4.3 Condition flags . . . . .	891	38.3.1.4 Exceptions and interrupts . . . . .	907
38.2.10.4.4 Examples . . . . .	891	38.3.1.5 Data types . . . . .	908
38.2.10.5 ISB . . . . .	892	38.3.1.6 The Cortex Microcontroller Software Interface Standard. . . . .	908
38.2.10.5.1 Syntax. . . . .	892	38.3.2 Memory model . . . . .	910
38.2.10.5.2 Operation . . . . .	892	38.3.2.1 Memory regions, types and attributes . . . . .	910
38.2.10.5.3 Condition flags . . . . .	892	38.3.2.2 Memory system ordering of memory accesses . . . . .	911
38.2.10.5.4 Examples . . . . .	892	38.3.2.3 Behavior of memory accesses . . . . .	912
38.2.10.6 MRS . . . . .	893	38.3.2.4 Software ordering of memory accesses. . . . .	912
38.2.10.6.1 Syntax. . . . .	893	38.3.2.5 Bit-banding. . . . .	913
38.2.10.6.2 Operation . . . . .	893	38.3.2.5.1 Directly accessing an alias region . . . . .	915
38.2.10.6.3 Restrictions. . . . .	893	38.3.2.5.2 Directly accessing a bit-band region . . . . .	915
38.2.10.6.4 Condition flags . . . . .	893	38.3.2.6 Memory endianness. . . . .	916
38.2.10.6.5 Examples . . . . .	893	38.3.2.6.1 Little-endian format . . . . .	916
38.2.10.7 MSR . . . . .	894	38.3.2.7 Synchronization primitives . . . . .	916
38.2.10.7.1 Syntax. . . . .	894	38.3.2.8 Programming hints for the synchronization primitives . . . . .	917
38.2.10.7.2 Operation . . . . .	894	38.3.3 Exception model . . . . .	918
Note . . . . .	894	38.3.3.1 Exception states. . . . .	918
38.2.10.7.3 Restrictions. . . . .	894	38.3.3.2 Exception types . . . . .	918
38.2.10.7.4 Condition flags . . . . .	894	38.3.3.3 Exception handlers . . . . .	920
38.2.10.7.5 Examples . . . . .	894	38.3.3.4 Vector table . . . . .	921
38.2.10.8 NOP . . . . .	895	38.3.3.5 Exception priorities. . . . .	921
38.2.10.8.1 Syntax. . . . .	895	38.3.3.6 Interrupt priority grouping. . . . .	922
38.2.10.8.2 Operation . . . . .	895	38.3.3.7 Exception entry and return. . . . .	922
38.2.10.8.3 Condition flags . . . . .	895	38.3.3.7.1 Exception entry . . . . .	923
38.2.10.8.4 Examples . . . . .	895	38.3.3.7.2 Exception return. . . . .	924
38.2.10.9 SEV . . . . .	896	38.3.4 Fault handling . . . . .	926
38.2.10.9.1 Syntax. . . . .	896	38.3.4.1 Fault types . . . . .	926
38.2.10.9.2 Operation . . . . .	896	38.3.4.2 Fault escalation and hard faults . . . . .	927
38.2.10.9.3 Condition flags . . . . .	896	38.3.4.3 Fault status registers and fault address registers 927	
38.2.10.9.4 Examples . . . . .	896	38.3.4.4 Lockup . . . . .	928
38.2.10.10 SVC . . . . .	897	38.3.5 Power management. . . . .	929
38.2.10.10.1 Syntax. . . . .	897	38.3.5.1 Entering sleep mode . . . . .	929
38.2.10.10.2 Operation . . . . .	897	38.3.5.1.1 Wait for interrupt . . . . .	929
38.2.10.10.3 Condition flags . . . . .	897	38.3.5.1.2 Wait for event. . . . .	929
38.2.10.10.4 Examples . . . . .	897	38.3.5.1.3 Sleep-on-exit . . . . .	930
38.2.10.11 WFE. . . . .	898	38.3.5.2 Wakeup from sleep mode . . . . .	930
38.2.10.11.1 Syntax. . . . .	898	38.3.5.2.1 Wakeup from WFI or sleep-on-exit . . . . .	930
38.2.10.11.2 Operation . . . . .	898	38.3.5.2.2 Wakeup from WFE. . . . .	930
38.2.10.11.3 Condition flags . . . . .	898	38.3.5.3 The Wakeup Interrupt Controller . . . . .	930
38.2.10.11.4 Examples . . . . .	898	38.3.5.4 Power management programming hints . . . . .	931
38.2.10.12 WFI . . . . .	899	<b>38.4 ARM Cortex-M3 User Guide: Peripherals . . . . .</b>	<b>932</b>
38.2.10.12.1 Syntax. . . . .	899	38.4.1 About the Cortex-M3 peripherals. . . . .	932
38.2.10.12.2 Operation . . . . .	899	38.4.2 Nested Vectored Interrupt Controller . . . . .	933
38.2.10.12.3 Condition flags . . . . .	899	38.4.2.1 The CMSIS mapping of the Cortex-M3 NVIC registers . . . . .	933
38.2.10.12.4 Examples . . . . .	899		
<b>38.3 ARM Cortex-M3 User Guide: Processor . . . . .</b>	<b>900</b>		
38.3.1 Programmers model . . . . .	900		
38.3.1.1 Processor mode and privilege levels for software execution . . . . .	900		

38.4.2.2	Interrupt Set-enable Registers . . . . .	934	38.4.3.11.3	Usage Fault Status Register . . . . .	952
38.4.2.3	Interrupt Clear-enable Registers. . . . .	934	38.4.3.12	Hard Fault Status Register. . . . .	954
38.4.2.4	Interrupt Set-pending Registers . . . . .	935	38.4.3.13	Memory Management Fault Address Register . . . . .	954
38.4.2.5	Interrupt Clear-pending Registers. . . . .	935	38.4.3.14	Bus Fault Address Register . . . . .	954
38.4.2.6	Interrupt Active Bit Registers . . . . .	936	38.4.3.15	System control block design hints and tips . . . . .	955
38.4.2.7	Interrupt Priority Registers . . . . .	936	38.4.4	System timer, SysTick . . . . .	956
38.4.2.8	Software Trigger Interrupt Register . . . . .	937	38.4.4.1	SysTick Control and Status Register . . . . .	956
38.4.2.9	Level-sensitive and pulse interrupts . . . . .	937	38.4.4.2	SysTick Reload Value Register . . . . .	957
38.4.2.9.1	Hardware and software control of interrupts. . . . .	938	38.4.4.2.1	Calculating the RELOAD value . . . . .	957
38.4.2.10	NVIC design hints and tips . . . . .	938	38.4.4.3	SysTick Current Value Register . . . . .	957
38.4.2.10.1	NVIC programming hints . . . . .	939	38.4.4.4	SysTick Calibration Value Register . . . . .	957
38.4.3	System control block . . . . .	940	38.4.4.5	SysTick design hints and tips. . . . .	958
38.4.3.1	The CMSIS mapping of the Cortex-M3 SCB registers . . . . .	940	38.4.5	Memory protection unit . . . . .	959
38.4.3.2	Auxiliary Control Register. . . . .	940	38.4.5.1	MPU Type Register . . . . .	960
38.4.3.2.1	About IT folding . . . . .	941	38.4.5.2	MPU Control Register . . . . .	961
38.4.3.3	CPUID Base Register. . . . .	941	38.4.5.3	MPU Region Number Register . . . . .	962
38.4.3.4	Interrupt Control and State Register. . . . .	941	38.4.5.4	MPU Region Base Address Register. . . . .	962
38.4.3.5	Vector Table Offset Register. . . . .	943	38.4.5.4.1	The ADDR field . . . . .	963
38.4.3.6	Application Interrupt and Reset Control Register . . . . .	944	38.4.5.5	MPU Region Attribute and Size Register. . . . .	963
38.4.3.6.1	Binary point . . . . .	945	38.4.5.5.1	SIZE field values . . . . .	964
38.4.3.7	System Control Register . . . . .	945	38.4.5.6	MPU access permission attributes. . . . .	965
38.4.3.8	Configuration and Control Register . . . . .	946	38.4.5.7	MPU mismatch. . . . .	966
38.4.3.9	System Handler Priority Registers . . . . .	947	38.4.5.8	Updating an MPU region . . . . .	966
38.4.3.9.1	System Handler Priority Register 1 . . . . .	948	38.4.5.8.1	Updating an MPU region using separate words . . . . .	966
38.4.3.9.2	System Handler Priority Register 2 . . . . .	948	38.4.5.8.2	Updating an MPU region using multi-word writes . . . . .	967
38.4.3.9.3	System Handler Priority Register 3 . . . . .	948	38.4.5.8.3	Subregions. . . . .	968
38.4.3.10	System Handler Control and State Register. . . . .	948	38.4.5.9	MPU design hints and tips . . . . .	969
	Caution . . . . .	949	38.4.5.9.1	MPU configuration for a microcontroller . . . . .	969
38.4.3.11	Configurable Fault Status Register. . . . .	950	<b>38.5</b>	<b>ARM Cortex-M3 User Guide: Glossary . . . . .</b>	<b>970</b>
38.4.3.11.1	Memory Management Fault Status Register . . . . .	950			
38.4.3.11.2	Bus Fault Status Register. . . . .	951			

**Chapter 39: Supplementary information**

<b>39.1</b>	<b>Abbreviations. . . . .</b>	<b>974</b>	<b>39.3</b>	<b>Tables. . . . .</b>	<b>976</b>
<b>39.2</b>	<b>Legal information. . . . .</b>	<b>975</b>	<b>39.4</b>	<b>Figures . . . . .</b>	<b>990</b>
39.2.1	Definitions. . . . .	975	<b>39.5</b>	<b>Contents. . . . .</b>	<b>993</b>
39.2.2	Disclaimers. . . . .	975			
39.2.3	Trademarks. . . . .	975			

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© NXP B.V. 2011. All rights reserved.

For more information, please visit: <http://www.nxp.com>  
 For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

Date of release: 7 March 2011

Document identifier: <Document ID>