UM10462

LPC11Uxx User manual

Rev. 2.1 — 13 January 2012 **User manual**

Document information

Info	Content
Keywords	LPC11Uxx, ARM Cortex-M0, microcontroller, LPC11U12, LPC11U14, LPC11U13, USB, LPC11U23, LPC11U24
Abstract	LPC11Uxx User manual

Revision history

Rev	Date	Description
2.1	20120113	LPC11Uxx User manual
		Modifications: • Description of PIOPOR1CAP register updated (see Table 34). • LPM register added (Table 201).
2	20111214	LPC11Uxx User manual
		Modifications: • Parts LPC11U2x added. • Chapter 22 added. • Part LPC11U14FHI33/201 added. • Bit 10 (TD) changed to reserved for PIO0_4 and PIO0_5 registers (Table 65, Table 66). • Register PIO1_29, bit description of FUNC bit updated: 0x0 = PIO1_29 (Table 114). • Section 20.16.4.7 “Algorithm and procedure for signature generation” updated. • Update description of ISP command Blank check sectors (Table 346). • Description of ISP Go command updated in Table 344. • Description of SYSMEMREMAP register updated in Table 7. • Requirement to enable the USART clock before the USART pins removed (see Chapter 3 and Section 12.2. • Register bit description of PINTSEL registers corrected (see Section 3.5.33). • Update the description of the address offset in for USB endpoint commands in Table 210. • Smart card application example updated (Section 12.6.2). • Chapter 10 added. • Explanation of clock synchronization steps expanded in Section 17.6. • Use of pins PIO0_3 and PIO0_1 clarified in Table 116. • Description of power profiles updated (Section 5.3). • LPM mode added in Chapter 11. • Description of LOCK bit corrected in Table 315. • Description of SYSRSTSTAT register updated (Table 15).
1	20110414	Initial version

Contact information

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

For sales office addresses, please send an email to: **salesaddresses@nxp.com**

1.1 Introduction

The LPC11Uxx are an ARM Cortex-M0 based, low-cost 32-bit MCU family, designed for 8/16-bit microcontroller applications, offering performance, low power, simple instruction set and memory addressing together with reduced code size compared to existing 8/16-bit architectures.

The LPC11Uxx operate at CPU frequencies of up to 50 MHz. Equipped with a highly flexible and configurable full-speed USB 2.0 device controller, the LPC11Uxx bring unparalleled design flexibility and seamless integration to today's demanding connectivity solutions.

The peripheral complement of the LPC11Uxx includes up to 32 kB of flash memory, up to 8 kB of SRAM data memory, one Fast-mode Plus I^2C-bus interface, one RS-485/EIA-485 USART with support for synchronous mode and smart card interface, two SSP interfaces, four general purpose counter/timers, a 10-bit ADC, and up to 54 general purpose I/O pins.

1.2 Features

- System:
  - ARM Cortex-M0 processor, running at frequencies of up to 50 MHz.
  - ARM Cortex-M0 built-in Nested Vectored Interrupt Controller (NVIC).
  - Non Maskable Interrupt (NMI) input selectable from several input sources.
  - System tick timer.
- Memory:
  - Up to 32 kB on-chip flash program memory.
  - In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software.
  - Total of 6 kB (4 kB main SRAM and 2 kB USB SRAM) SRAM data memory (LPC11U1x) or up to 10 kB (8 kB main SRAM and 2 kB USB SRAM) SRAM data memory (LPC11U2x).
  - 16 kB boot ROM.
  - LPC11U2x only: Up to 4 kB on-chip EEPROM data memory; byte erasable and byte programmable; on-chip API support.
- ROM based drivers:
  - Power profiles.
  - 32-bit integer division routines.
  - LPC11U2x only: ROM-based USB drivers. Flash updates via USB supported. Supports Human-Interface Device (HID) class, Mass Storage Device Class (MSC), and Communication Device Class (CDC).
  - LPC11U2x only: IAP EEPROM drivers.

- Debug options:
  - Standard JTAG test interface for BSDL.
  - Serial Wire Debug.
- Digital peripherals:
  - Up to 54 General Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors, repeater mode, and open-drain mode.
  - Up to eight GPIO pins can be selected as edge and level sensitive interrupt sources.
  - Two GPIO grouped interrupt modules enables an interrupt based on a programmable pattern of input states of a group of GPIO pins.
  - High-current source output driver (20 mA) on one pin (P0_7).
  - High-current sink driver (20 mA) on true open-drain pins (P0_4 and P0_5).
  - Four general purpose counter/timers with a total of 4 capture inputs and 13 match outputs.
  - Programmable windowed WatchDog Timer (WDT) with a dedicated, internal low-power WatchDog Oscillator (WDO).
- Analog peripherals:
  - 10-bit ADC with input multiplexing among eight pins.
- Serial interfaces:
  - USB 2.0 full-speed device controller.
  - USART with fractional baud rate generation, internal FIFO, a full modem control handshake interface, and support for RS-485/9-bit mode and synchronous mode. USART supports an asynchronous smart card interface (ISO 7816-3).
  - Two SSP interfaces with FIFO and multi-protocol capabilities.
  - I^2C-bus interface supporting the full I^2C-bus specification and Fast-mode Plus with a data rate of up to 1 Mbit/s with multiple address recognition and monitor mode.
- Clock generation:
  - Crystal Oscillator with an operating range of 1 MHz to 25 MHz (system oscillator).
  - 12 MHz Internal high-frequency RC oscillator (IRC) that can optionally be used as a system clock.
  - Internal low-power, low-frequency WatchDog Oscillator (WDO) with programmable frequency output.
  - PLL allows CPU operation up to the maximum CPU rate with the system oscillator or the IRC as clock sources.
  - A second, dedicated PLL is provided for USB.
  - Clock output function with divider that can reflect the crystal oscillator, the main clock, the IRC, or the watchdog oscillator.
- Power control:
  - Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.
  - Power profiles residing in boot ROM allow optimized performance and minimized power consumption for any given application through one simple function call.

- Processor wake-up from Deep-sleep and Power-down modes via reset, selectable GPIO pins, watchdog interrupt, BOD interrupt, or USB port activity.
- Processor wake-up from Deep power-down mode using one special function pin.
- Integrated PMU (Power Management Unit) to minimize power consumption during Sleep, Deep-sleep, Power-down, and Deep power-down modes.
- Power-On Reset (POR).
- Brownout detect with four separate thresholds for interrupt and forced reset.

- Unique device serial number for identification.
- Single 3.3 V power supply (1.8 V to 3.6 V).
- Temperature range −40 °C to +85 °C.
- Available as LQFP64, LQFP48, TFBGA48 packages, and as HVQFN33 in two package sizes: 5 x 5 x 0.85 mm and 7 x 7 x 0.85 mm.
- Pin-compatible to the ARM Cortex-M3 based LPC134x series.

1.3 Ordering information

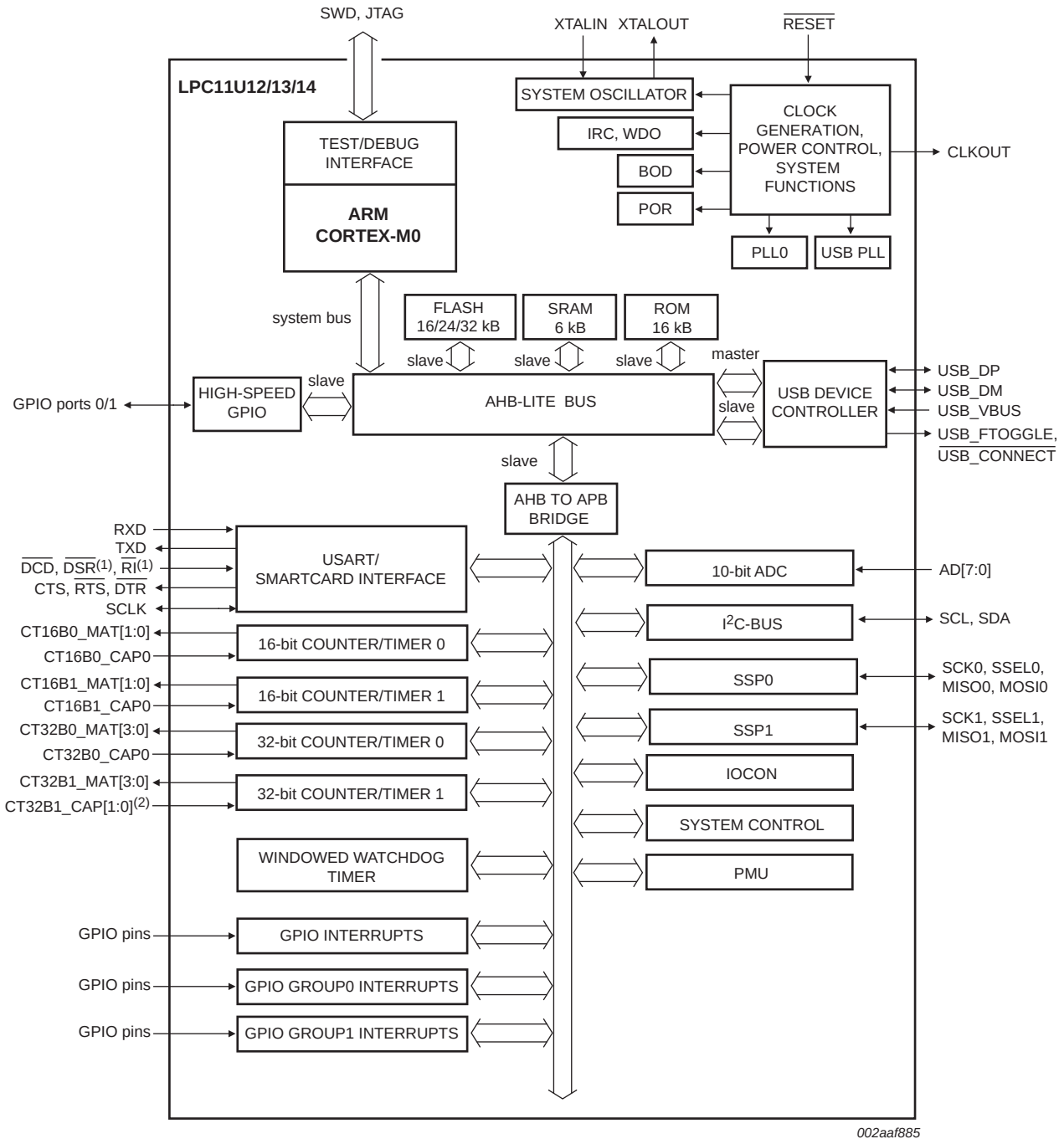
Table 1. Ordering information

Type number	Package		
	Name	Description	Version
LPC11U12FHN33/201	HVQFN33	plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
LPC11U12FBD48/201	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U13FBD48/201	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U14FHN33/201	HVQFN33	plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
LPC11U14FHI33/201	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 5 × 5 × 0.85 mm	n/a
LPC11U14FBD48/201	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U14FET48/201	TFBGA48	plastic thin fine-pitch ball grid array package; 48 balls; body 4.5 × 4.5 × 0.7 mm	SOT1155-2
LPC11U23FBD48/301	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U24FHI33/301	HVQFN33	plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 5 × 5 × 0.85 mm	n/a
LPC11U24FBD48/301	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U24FET48/301	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U24FHN33/401	HVQFN33	plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
LPC11U24FBD48/401	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC11U24FBD64/401	LQFP64	plastic low profile quad flat package; 64 leads; body 10 × 10 × 1.4 mm	SOT314-2

Table 2. Part ordering options

Part Number	FLASH (kB)	MAIN SRAM (kB)	USB SRAM (kB)	EEPROM (kB)	USB	I2C/ Fast+	SSP	ADC Channels	GPIO	Package
LPC11U12FHN33/201	16	4	2	N/A	1	1	2	8	26	7x7 HVQFN33
LPC11U12FBD48/201	16	4	2	N/A	1	1	2	8	40	LQFP48
LPC11U13FBD48/201	24	4	2	N/A	1	1	2	8	40	LQFP48
LPC11U14FHI33/201	32	4	2	N/A	1	1	2	8	26	5x5 HVQFN33
LPC11U14FHN33/201	32	4	2	N/A	1	1	2	8	26	7x7 HVQFN33
LPC11U14FBD48/201	32	4	2	N/A	1	1	2	8	40	LQFP48
LPC11U14FET48/201	32	4	2	N/A	1	1	2	8	40	TFBGA48
LPC11U23FBD48/301	24	6	2	1	1	1	2	8	40	LQFP48
LPC11U24FHI33/301	32	6	2	2	1	1	2	8	26	5x5 HVQFN33
LPC11U24FBD48/301	32	6	2	2	1	1	2	8	40	LQFP48
LPC11U24FET48/301	32	6	2	2	1	1	2	8	40	TFBGA48
LPC11U24FHN33/401	32	8	2	4	1	1	2	8	26	7x7 HVQFN33
LPC11U24FBD48/401	32	8	2	4	1	1	2	8	40	LQFP48
LPC11U24FBD64/401	32	8	2	4	1	1	2	8	54	LQFP64

1.4 Block diagram



(1) Function not available on the HVQFN33 package.

(2) CT32B1_CAP1 is only available on the TFBGA48 package.

Fig 1. Block diagram (LPC11U1x)

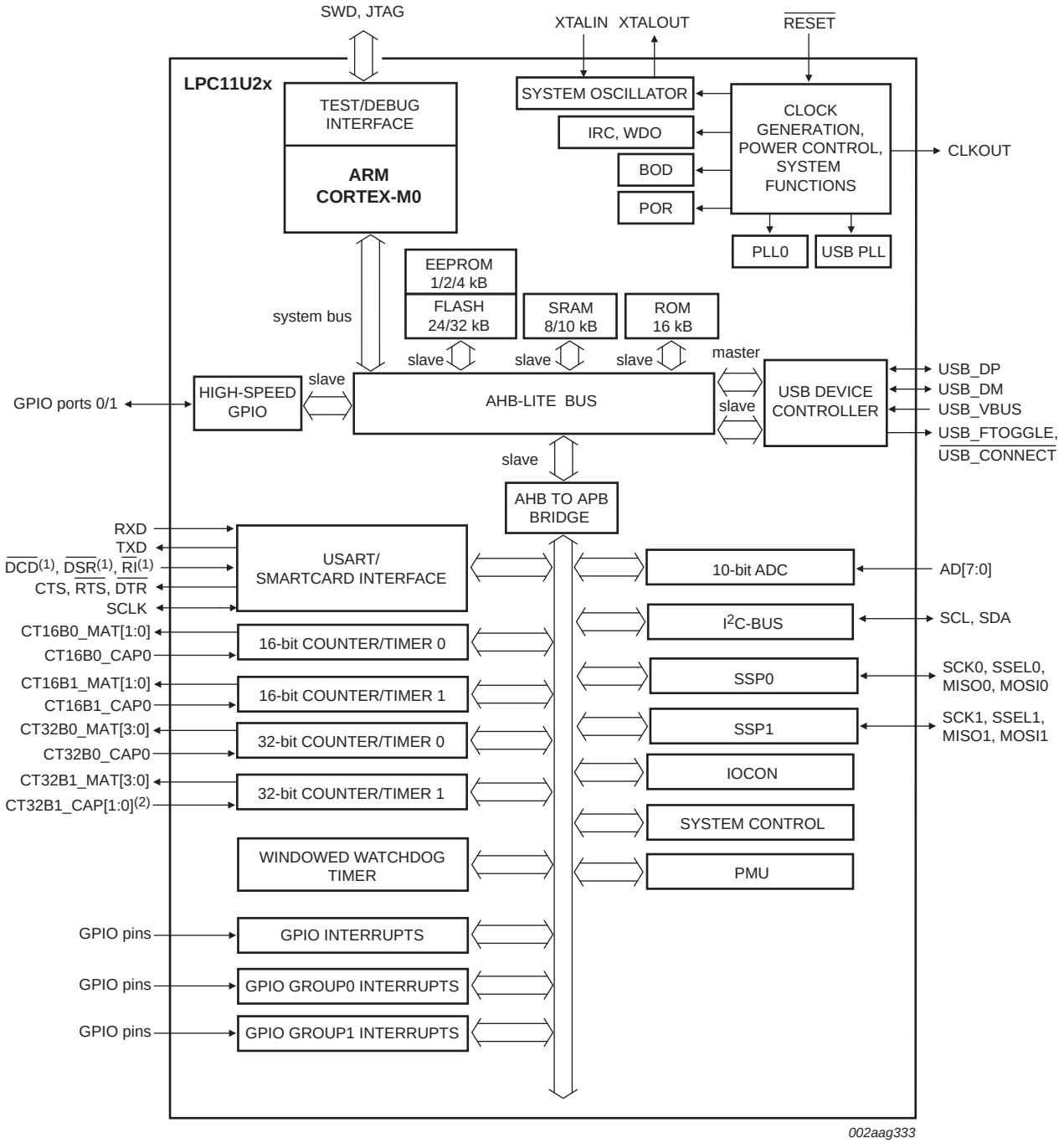


Fig 2. Block diagram (LPC11U2x)

2.1 How to read this chapter

See Table 3 for the memory configuration of the LPC11Uxx parts.

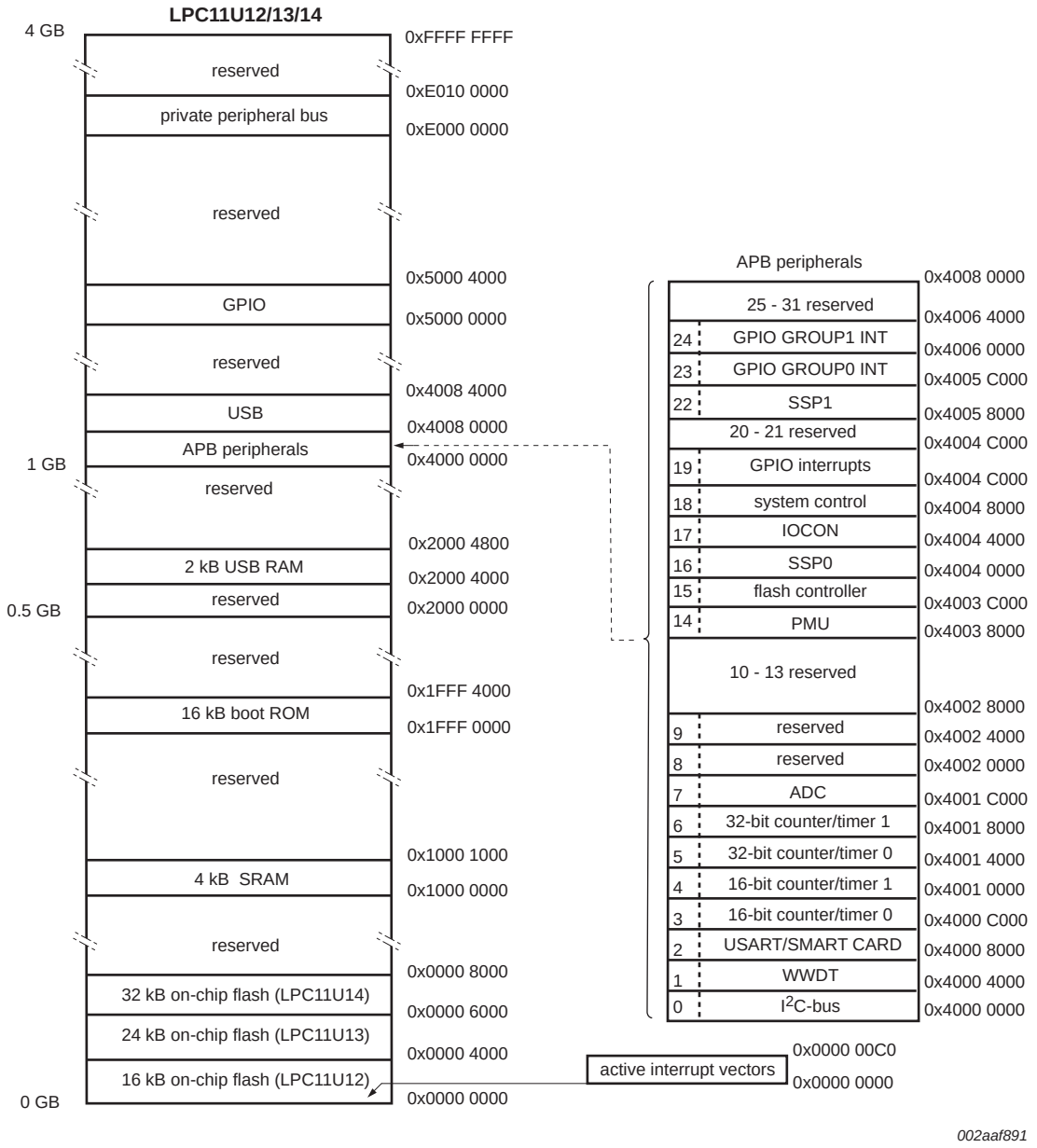
Table 3. LPC11Uxx memory configuration

Part	Flash	Main SRAM	USB SRAM	EEPROM	Reference
LPC11U12FHN33/201	16 kB	4 kB	2 kB	n/a	Figure 3
LPC11U12FBD48/201	16 kB	4 kB	2 kB	n/a	Figure 3
LPC11U13FBD48/201	24 kB	4 kB	2 kB	n/a	Figure 3
LPC11U14FHN33/201	32 kB	4 kB	2 kB	n/a	Figure 3
LPC11U14FHI33/201	32 kB	4 kB	2 kB	n/a	Figure 3
LPC11U14FBD48/201	32 kB	4 kB	2 kB	n/a	Figure 3
LPC11U14FET48/201	32 kB	4 kB	2 kB	n/a	Figure 3
LPC11U23FBD48/301	24 kB	6 kB	2 kB	1 kB	Figure 4
LPC11U24FHI33/301	32 kB	6 kB	2 kB	2 kB	Figure 4
LPC11U24FBD48/301	32 kB	6 kB	2 kB	2 kB	Figure 4
LPC11U24FET48/301	32 kB	6 kB	2 kB	2 kB	Figure 4
LPC11U24FHN33/401	32 kB	8 kB	2 kB	4 kB	Figure 4
LPC11U24FBD48/401	32 kB	8 kB	2 kB	4 kB	Figure 4
LPC11U24FBD64/401	32 kB	8 kB	2 kB	4 kB	Figure 4

2.2 Memory map

The LPC11Uxx 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 AHB peripheral area is 2 MB in size and is divided to allow for up to 128 peripherals. The APB peripheral area is 512 kB in size and is divided to allow for up to 32 peripherals. Each peripheral of either type is allocated 16 kB of space. This allows simplifying the address decoding for each peripheral.



SSP1 available on 48-pin packages only.

Fig 3. LPC11U1x memory map

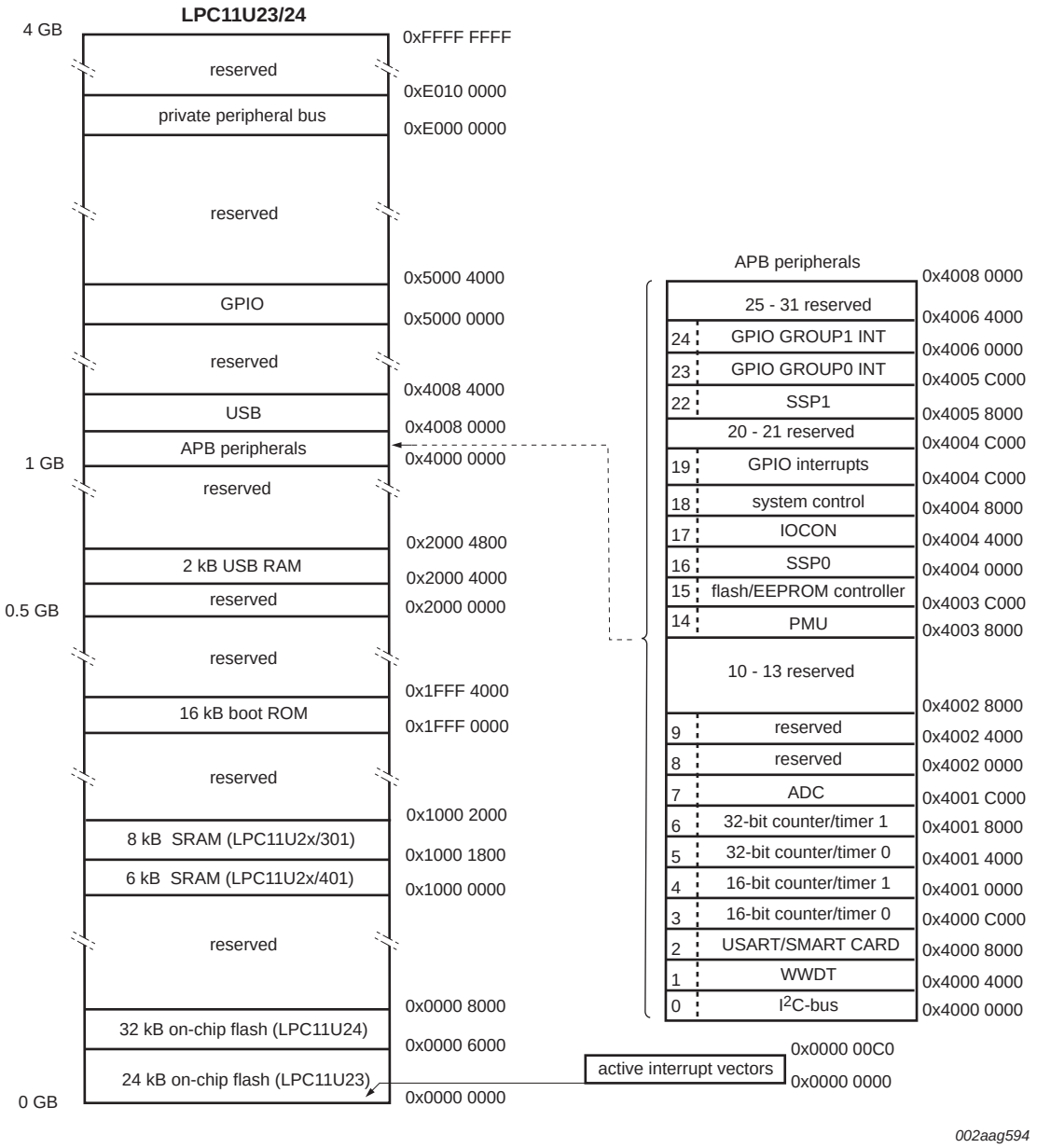


Fig 4. LPC11U2x memory map

3.1 How to read this chapter

The system control block is identical for all LPC11Uxx parts.

3.2 Introduction

The system configuration block controls oscillators, some aspects of the power management, and the clock generation of the LPC11Uxx. Also included in this block is a register for remapping flash, SRAM, and ROM memory areas.

3.3 Pin description

Table 4 shows pins that are associated with system control block functions.

Table 4. Pin summary

Pin name	Pin direction	Pin description
CLKOUT	O	Clockout pin
PIO0 and PIO1 pins	I	Eight pins can be selected as external interrupt pins from all available GPIO pins (see Table 39).

3.4 Clocking and power control

See Figure 5 for an overview of the LPC11Uxx Clock Generation Unit (CGU).

The LPC11Uxx include three independent oscillators. These are the system oscillator, the Internal RC oscillator (IRC), and the Watchdog oscillator. Each oscillator can be used for more than one purpose as required in a particular application.

Following reset, the LPC11Uxx will operate from the Internal RC oscillator until switched by software. This allows systems to operate without an external crystal and the bootloader code to operate at a known frequency.

The SYSAHBCLKCTRL register gates the system clock to the various peripherals and memories. USART and SSP have individual clock dividers to derive peripheral clocks from the main clock.

The main clock, and the clock outputs from the IRC, the system oscillator, and the watchdog oscillator can be observed directly on the CLKOUT pin.

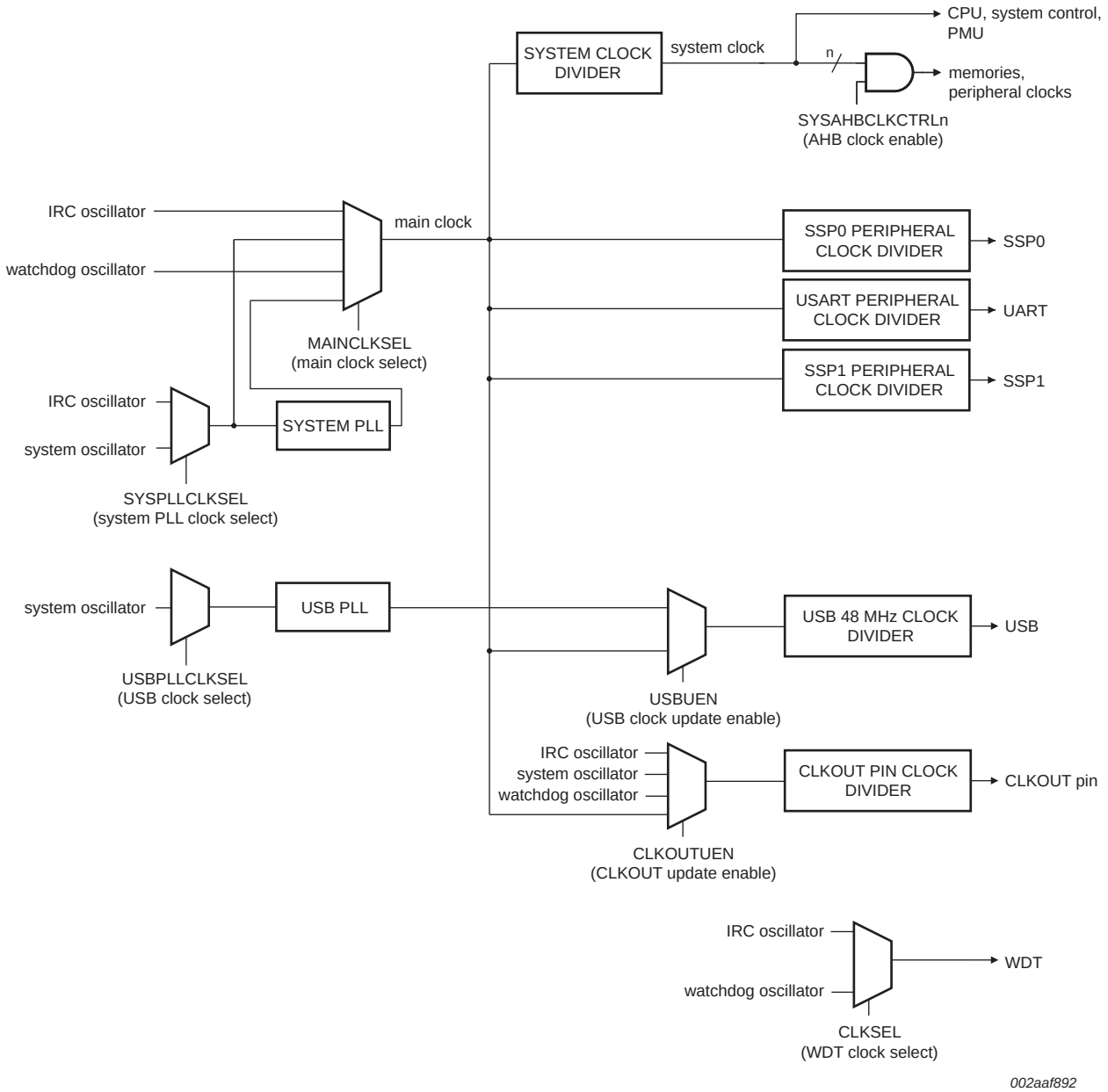


Fig 5. LPC11Uxx CGU block diagram

3.5 Register description

All system control block registers are on word address boundaries. Details of the registers appear in the description of each function.

In addition to the system control block registers described in Table 5, the flash access timing register, which can be re-configured as part the system setup, is described in Table 6. This register is not part of the system configuration block.

All address offsets not shown in Table 5 and Table 6 are reserved and should not be written.

Table 5. Register overview: system control block (base address 0x4004 8000)

Name	Access	Offset	Description	Reset value	Reference
SYSMEMREMAP	R/W	0x000	System memory remap	0x02	Table 7
PRESETCTRL	R/W	0x004	Peripheral reset control	0	Table 8
SYSPLLCTRL	R/W	0x008	System PLL control	0	Table 9
SYSPLLSTAT	R	0x00C	System PLL status	0	Table 10
USBPLLCTRL	R/W	0x010	USB PLL control	0	Table 11
USBPLLSTAT	R	0x014	USB PLL status	0	Table 12
SYSOSCCTRL	R/W	0x020	System oscillator control	0x000	Table 13
WDTOSCCTRL	R/W	0x024	Watchdog oscillator control	0x0A0	Table 14
-	-	0x028	Reserved	-	-
-	-	0x02C	Reserved	-	-
SYSRSTSTAT	R/W	0x030	System reset status register	0	Table 15
SYSPLLCLKSEL	R/W	0x040	System PLL clock source select	0	Table 16
SYSPLLCLKUEN	R/W	0x044	System PLL clock source update enable	0	Table 17
USBPLLCLKSEL	R/W	0x048	USB PLL clock source select	0	Table 18
USBPLLCLKUEN	R/W	0x04C	USB PLL clock source update enable	0	Table 19
MAINCLKSEL	R/W	0x070	Main clock source select	0	Table 20
MAINCLKUEN	R/W	0x074	Main clock source update enable	0	Table 21
SYSAHBCLKDIV	R/W	0x078	System clock divider	0x001	Table 22
SYSAHBCLKCTRL	R/W	0x080	System clock control	0x3F	Table 23
SSP0CLKDIV	R/W	0x094	SSP0 clock divider	0	Table 24
UARTCLKDIV	R/W	0x098	UART clock divider	0	Table 25
SSP1CLKDIV	R/W	0x09C	SSP1 clock divider	0	Table 26
-	-	0x0A0 - 0x0BC	Reserved	-	-
USBCLKSEL	R/W	0x0C0	USB clock source select	0	Table 27
USBCLKUEN	R/W	0x0C4	USB clock source update enable	0	Table 28
USBCLKDIV	R/W	0x0C8	USB clock source divider	0	Table 29
-	-	0x0CC	Reserved	-	-
CLKOUTSEL	R/W	0x0E0	CLKOUT clock source select	0	Table 30
CLKOUTUEN	R/W	0x0E4	CLKOUT clock source update enable	0	Table 31
CLKOUTDIV	R/W	0x0E8	CLKOUT clock divider	0	Table 32
PIOPORCAP0	R	0x100	POR captured PIO status 0	user dependent	Table 33
PIOPORCAP1	R	0x104	POR captured PIO status 1	user dependent	Table 34
BODCTRL	R/W	0x150	Brown-Out Detect	0	Table 35
SYSTCKCAL	R/W	0x154	System tick counter calibration	0x0	Table 36

Table 5. Register overview: system control block (base address 0x4004 8000) *…continued*

Name	Access	Offset	Description	Reset value	Reference
IRQLATENCY	R/W	0x170	IQR delay. Allows trade-off between interrupt latency and determinism.	0x0000 0010	
NMISRC	R/W	0x174	NMI Source Control	0	Table 38
PINTSEL0	R/W	0x178	GPIO Pin Interrupt Select register 0	0	Table 39
PINTSEL1	R/W	0x17C	GPIO Pin Interrupt Select register 1	0	Table 39
PINTSEL2	R/W	0x180	GPIO Pin Interrupt Select register 2	0	Table 39
PINTSEL3	R/W	0x184	GPIO Pin Interrupt Select register 3	0	Table 39
PINTSEL4	R/W	0x188	GPIO Pin Interrupt Select register 4	0	Table 39
PINTSEL5	R/W	0x18C	GPIO Pin Interrupt Select register 5	0	Table 39
PINTSEL6	R/W	0x190	GPIO Pin Interrupt Select register 6	0	Table 39
PINTSEL7	R/W	0x194	GPIO Pin Interrupt Select register 7	0	Table 39
USBCLKCTRL	R/W	0x198	USB clock control	0x0	Table 40
USBCLKST	R	0x19C	USB clock status	0x1	Table 41
STARTERP0	R/W	0x204	Start logic 0 interrupt wake-up enable register 0	0	Table 42
STARTERP1	R/W	0x214	Start logic 1 interrupt wake-up enable register 1	0	Table 43
PDSLEEPCFG	R/W	0x230	Power-down states in deep-sleep mode	0xFFFF	Table 44
PDAWAKECFG	R/W	0x234	Power-down states for wake-up from deep-sleep	0xEDF0	Table 45
PDRUNCFG	R/W	0x238	Power configuration register	0xEDF0	Table 46
DEVICE_ID	R	0x3F4	Device ID	part dependent	Table 47

Table 6. Register overview: flash control block (base address 0x4003 C000)

Name	Access	Offset	Description	Reset value	Reference
FLASHCFG	R/W	0x010	Flash read access configuration	-	Table 48

3.5.1 System memory remap register

The system memory remap register selects whether the exception vectors are read from boot ROM, flash, or SRAM. By default, the flash memory is mapped to address 0x0000 0000. When the MAP bits in the SYSMEMREMAP register are set to 0x0 or 0x1, the boot ROM or RAM respectively are mapped to the bottom 512 bytes of the memory map (addresses 0x0000 0000 to 0x0000 0200).

Table 7. System memory remap register (SYSMEMREMAP, address 0x4004 8000) bit description

Bit	Symbol	Value	Description	Reset value
1:0	MAP		System memory remap. Value 0x3 is reserved.	0x2
		0x0	Boot Loader Mode. Interrupt vectors are re-mapped to Boot ROM.	
		0x1	User RAM Mode. Interrupt vectors are re-mapped to Static RAM.	
		0x2	User Flash Mode. Interrupt vectors are not re-mapped and reside in Flash.	
31:2	-	-	Reserved	-

3.5.2 Peripheral reset control register

This register allows software to reset specific peripherals. A 0 in an assigned bit in this register resets the specified peripheral. A 1 negates the reset and allows peripheral operation.

Remark: Before accessing the SSP and I2C peripherals, write a 1 to this register to ensure that the reset signals to the SSP and I2C are de-asserted.

Table 8. Peripheral reset control register (PRESETCTRL, address 0x4004 8004) bit description

Bit	Symbol	Value	Description	Reset value
0	SSP0_RST_N		SSP0 reset control	0
		0	Resets the SSP0 peripheral.	
		1	SSP0 reset de-asserted.	
1	I2C_RST_N		I2C reset control	0
		0	Resets the I2C peripheral.	
		1	I2C reset de-asserted.	
2	SSP1_RST_N		SSP1 reset control	0
		0	Resets the SSP1 peripheral.	
		1	SSP1 reset de-asserted.	
3	-		Reserved	-
31:4	-	-	Reserved	-

3.5.3 System PLL control register

This register connects and enables the system PLL and configures the PLL multiplier and divider values. The PLL accepts an input frequency from 10 MHz to 25 MHz from various clock sources. The input frequency is multiplied to a higher frequency and then divided down to provide the actual clock used by the CPU, peripherals, and memories. The PLL can produce a clock up to the maximum allowed for the CPU.

Table 9. System PLL control register (SYSPLLCTRL, address 0x4004 8008) bit description

Bit	Symbol	Value	Description	Reset value
4:0	MSEL		Feedback divider value. The division value M is the programmed MSEL value + 1. 00000: Division ratio M = 1 to 11111: Division ratio M = 32	0
6:5	PSEL		Post divider ratio P. The division ratio is 2 × P.	0
		0x0	P = 1	
		0x1	P = 2	
		0x2	P = 4	
		0x3	P = 8	
31:7	-	-	Reserved. Do not write ones to reserved bits.	-

3.5.4 System PLL status register

This register is a Read-only register and supplies the PLL lock status (see Section 3.10.1).

Table 10. System PLL status register (SYSPLLSTAT, address 0x4004 800C) bit description

Bit	Symbol	Value	Description	Reset value
0	LOCK		PLL lock status	0
		0	PLL not locked	
		1	PLL locked	
31:1	-	-	Reserved	-

3.5.5 USB PLL control register

The USB PLL is identical to the system PLL and is used to provide a dedicated clock to the USB block if available (see Section 3.1).

This register connects and enables the USB PLL and configures the PLL multiplier and divider values. The PLL accepts an input frequency from 10 MHz to 25 MHz from various clock sources. The input frequency is multiplied up to a high frequency, then divided down to provide the actual clock 48 MHz clock used by the USB subsystem.

Remark: The USB PLL must be connected to the system oscillator for correct USB operation (see Table 18).

Table 11. USB PLL control register (USBPLLCTRL, address 0x4004 8010) bit description

Bit	Symbol	Value	Description	Reset value
4:0	MSEL		Feedback divider value. The division value M is the programmed MSEL value + 1. 00000: Division ratio M = 1 to 11111: Division ratio M = 32	0x000
6:5	PSEL		Post divider ratio P. The division ratio is 2 × P.	0x00
		0x0	P = 1	
		0x1	P = 2	
		0x2	P = 4	
		0x3	P = 8	
31:7	-	-	Reserved. Do not write ones to reserved bits.	0x00

3.5.6 USB PLL status register

This register is a Read-only register and supplies the PLL lock status (see Section 3.10.1).

Table 12. USB PLL status register (USBPLLSTAT, address 0x4004 8014) bit description

Bit	Symbol	Value	Description	Reset value
0	LOCK		PLL lock status	0x0
		0	PLL not locked	
		1	PLL locked	
31:1	-	-	Reserved	0x00

3.5.7 System oscillator control register

This register configures the frequency range for the system oscillator.

Table 13. System oscillator control register (SYSOSCCTRL, address 0x4004 8020) bit description

Bit	Symbol	Value	Description	Reset value
0	BYPASS		Bypass system oscillator	0x0
		0	Oscillator is not bypassed.	
		1	Bypass enabled. PLL input (sys_osc_clk) is fed directly from the XTALIN and XTALOUT pins.	
1	FREQRANGE		Determines frequency range for Low-power oscillator.	0x0
		0	1 - 20 MHz frequency range.	
		1	15 - 25 MHz frequency range	
31:2	-	-	Reserved	0x00

3.5.8 Watchdog oscillator control register

This register configures the watchdog oscillator. The oscillator consists of an analog and a digital part. The analog part contains the oscillator function and generates an analog clock (F_{clkana}). The FREQSEL field selects F_{clkana} between 0.5 and 3.4 MHz. In the digital part, F_{clkana} is divided to produce the oscillator output clock under the control of the DIVSEL field.

The output clock frequency can be calculated as
wdt_osc_clk = Fclkana/(2 × (1 + DIVSEL).

Remark: Any non-zero setting of the FREQSEL field will yield a F_{clkana} value within ±40% of the listed frequency value.

Table 14. Watchdog oscillator control register (WDTOSCCTRL, address 0x4004 8024) bit description

Bit	Symbol	Value	Description	Reset value
4:0	DIVSEL		Select divider for Fclkana. wdt_osc_clk = Fclkana/ (2 × (1 + DIVSEL)) 00000: 2 × (1 + DIVSEL) = 2 00001: 2 × (1 + DIVSEL) = 4 to 11111: 2 × (1 + DIVSEL) = 64	0
8:5	FREQSEL		Select watchdog oscillator analog output frequency (Fclkana). Value 0x0 is reserved. Operation is undefined for this value. Startup code should program a non-zero value in this field as soon after reset as possible.	0
		0x1	0.5 MHz	
		0x2	0.8 MHz	
		0x3	1.1 MHz	
		0x4	1.4 MHz	
		0x5	1.6 MHz	
		0x6	1.8 MHz	
		0x7	2.0 MHz	
		0x8	2.2 MHz	
		0x9	2.4 MHz	
		0xA	2.6 MHz	
		0xB	2.7 MHz	
		0xC	2.9 MHz	
		0xD	3.1 MHz	
		0xE	3.2 MHz	
		0xF	3.4 MHz	
31:9	-	-	Reserved	-

3.5.9 System reset status register

If another reset signal - for example the external $\overline{\text{RESET}}$ pin - remains asserted after the POR signal is negated, then its bit is set to detected. Write a one to clear the reset.

The reset value given in Table 15 applies to the POR reset.

Table 15. System reset status register (SYSRSTSTAT, address 0x4004 8030) bit description

Bit	Symbol	Value	Description	Reset value
0	POR		POR reset status	0
		0	No POR detected	
		1	POR detected. Writing a one clears this reset.	
1	EXTRST		External reset status	0
		0	No reset event detected.	
		1	Reset detected. Writing a one clears this reset.	
2	WDT		Status of the Watchdog reset	0
		0	No WDT reset detected	
		1	WDT reset detected. Writing a one clears this reset.	
3	BOD		Status of the Brown-out detect reset	0
		0	No BOD reset detected	
		1	BOD reset detected. Writing a one clears this reset.	
4	SYSRST		Status of the software system reset	0
		0	No System reset detected	
		1	System reset detected. Writing a one clears this reset.	
31:5	-	-	Reserved	-

3.5.10 System PLL clock source select register

This register selects the clock source for the system PLL. The SYSPLLCLKUEN register (see Section 3.5.11) must be toggled from LOW to HIGH for the update to take effect.

Table 16. System PLL clock source select register (SYSPLLCLKSEL, address 0x4004 8040) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		System PLL clock source	0
		0x0	IRC	
		0x1	Crystal Oscillator (SYSOSC)	
		0x2	Reserved	
		0x3	Reserved	
31:2	-	-	Reserved	-

3.5.11 System PLL clock source update register

This register updates the clock source of the system PLL with the new input clock after the SYSPLLCLKSEL register has been written to. In order for the update to take effect, first write a zero to the SYSPLLUEN register and then write a one to SYSPLLUEN.

Table 17. System PLL clock source update enable register (SYSPLLCLKUEN, address 0x4004 8044) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA		Enable system PLL clock source update	0
		0	No change	
		1	Update clock source	
31:1	-	-	Reserved	-

3.5.12 USB PLL clock source select register

This register selects the clock source for the dedicated USB PLL. The USBPLLCLKUEN register (see Section 3.5.13) must be toggled from LOW to HIGH for the update to take effect.

Remark: When switching clock sources, both clocks must be running before the clock source is updated in the USBPLLCLKUEN register. For USB operation, the clock source must be switched from IRC to system oscillator with both the IRC and the system oscillator running. After the switch, the IRC can be turned off.

Table 18. USB PLL clock source select register (USBPLLCLKSEL, address 0x4004 8048) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		USB PLL clock source	0x00
		0x0	IRC. The USB PLL clock source must be switched to system oscillator for correct USB operation.	
		0x1	System oscillator	
		0x2	Reserved	
		0x3	Reserved	
31:2	-	-	Reserved	0x00

3.5.13 USB PLL clock source update enable register

This register updates the clock source of the USB PLL with the new input clock after the USBPLLCLKSEL register has been written to. In order for the update to take effect at the USB PLL input, first write a zero to the USBPLLUEN register and then write a one to USBPLLUEN.

Remark: The system oscillator must be selected in the USBPLLCLKSEL register in order to use the USB PLL, and this register must be toggled to update the USB PLL clock with the system oscillator.

Remark: When switching clock sources, both clocks must be running before the clock source is updated.

Table 19. USB PLL clock source update enable register (USBPLLCLKUEN, address 0x4004 804C) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA		Enable USB PLL clock source update	0x0
		0	No change	
		1	Update clock source	
31:1	-	-	Reserved	0x00

3.5.14 Main clock source select register

This register selects the main system clock, which can be the system PLL (sys_pllclkout), or the watchdog oscillator, or the IRC oscillator. The main system clock clocks the core, the peripherals, and the memories.

Bit 0 of the MAINCLKUEN register (see Section 3.5.15) must be toggled from 0 to 1 for the update to take effect.

Table 20. Main clock source select register (MAINCLKSEL, address 0x4004 8070) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		Clock source for main clock	0
		0x0	IRC Oscillator	
		0x1	PLL input	
		0x2	Watchdog oscillator	
		0x3	PLL output	
31:2	-	-	Reserved	-

3.5.15 Main clock source update enable register

This register updates the clock source of the main clock with the new input clock after the MAINCLKSEL register has been written to. In order for the update to take effect, first write a zero to bit 0 of this register, then write a one.

Table 21. Main clock source update enable register (MAINCLKUEN, address 0x4004 8074) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA		Enable main clock source update	0
		0	No change	
		1	Update clock source	
31:1	-	-	Reserved	-

3.5.16 System clock divider register

This register controls how the main clock is divided to provide the system clock to the core, memories, and the peripherals. The system clock can be shut down completely by setting the DIV field to zero.

Table 22. System clock divider register (SYSAHBCLKDIV, address 0x4004 8078) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	System AHB clock divider values 0: System clock disabled. 1: Divide by 1. to 255: Divide by 255.	0x01
31:8	-	Reserved	-

3.5.17 System clock control register

The SYSAHBCLKCTRL register enables the clocks to individual system and peripheral blocks. The system clock (bit 0) provides the clock for the AHB, the APB bridge, the ARM Cortex-M0, the Syscon block, and the PMU. This clock cannot be disabled.

Table 23. System clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description

Bit	Symbol	Value	Description	Reset value
0	SYS		Enables the clock for the AHB, the APB bridge, the Cortex-M0 FCLK and HCLK, SysCon, and the PMU. This bit is read only and always reads as 1.	1
		0	Reserved	
		1	Enable	
1	ROM		Enables clock for ROM.	1
		0	Disable	
		1	Enable	
2	RAM		Enables clock for RAM.	1
		0	Disable	
		1	Enable	
3	FLASHREG		Enables clock for flash register interface.	1
		0	Disable	
		1	Enable	
4	FLASHARRAY		Enables clock for flash array access.	1
		0	Disable	
		1	Enable	
5	I2C		Enables clock for I2C.	1
		0	Disable	
		1	Enable	
6	GPIO		Enables clock for GPIO port registers.	0
		0	Disable	
		1	Enable	
7	CT16B0		Enables clock for 16-bit counter/timer 0.	0
		0	Disable	
		1	Enable	

Table 23. System clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
8	CT16B1		Enables clock for 16-bit counter/timer 1.	0
		0	Disable	
		1	Enable	
9	CT32B0		Enables clock for 32-bit counter/timer 0.	0
		0	Disable	
		1	Enable	
10	CT32B1		Enables clock for 32-bit counter/timer 1.	
		0	Disable	
		1	Enable	
11	SSP0		Enables clock for SSP0.	0
		0	Disable	
		1	Enable	
12	USART		Enables clock for UART.	
		0	Disable	
		1	Enable	
13	ADC		Enables clock for ADC.	0
		0	Disable	
		1	Enable	
14	USB		Enables clock to the USB register interface.	0
		0	Disable	
		1	Enable	
15	WWDT		Enables clock for WWDT.	0
		0	Disable	
		1	Enable	
16	IOCON		Enables clock for I/O configuration block.	0
		0	Disable	
		1	Enable	
17	-		Reserved	0
18	SSP1		Enables clock for SSP1.	0
		0	Disable	
		1	Enable	
19	PINT		Enables clock to GPIO Pin interrupts register interface.	0
		0	Disable	
		1	Enable	
22:20	-		Reserved	-
23	GROUP0INT		Enables clock to GPIO GROUP0 interrupt register interface.	0
		0	Disable	
		1	Enable	

Table 23. System clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
24	GROUP1INT		Enables clock to GPIO GROUP1 interrupt register interface.	0
		0	Disable	
		1	Enable	
26:25	-	-	Reserved	-
27	USBRAM		Enables USB SRAM block.	0
		0	Disable	
		1	Enable	
31:28	-	-	Reserved	-

3.5.18 SSP0 clock divider register

This register configures the SSP0 peripheral clock SPI0_PCLK. SPI0_PCLK can be shut down by setting the DIV field to zero.

Table 24. SSP0 clock divider register (SSP0CLKDIV, address 0x4004 8094) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	SPI0_PCLK clock divider values. 0: System clock disabled. 1: Divide by 1. to 255: Divide by 255.	0
31:8	-	Reserved	-

3.5.19 USART clock divider register

This register configures the USART peripheral clock UART_PCLK. The UART_PCLK can be shut down by setting the DIV field to zero.

Table 25. USART clock divider register (UARTCLKDIV, address 0x4004 8098) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	UART_PCLK clock divider values 0: Disable UART_PCLK. 1: Divide by 1. to 255: Divide by 255.	0
31:8	-	Reserved	-

3.5.20 SSP1 clock divider register

This register configures the SSP1 peripheral clock SSP1_PCLK. The SSP1_PCLK can be shut down by setting the DIV bits to 0x0.

Table 26. SPI1 clock divider register (SSP1CLKDIV, address 0x4004 809C) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	SSP1_PCLK clock divider values 0: Disable SSP1_PCLK. 1: Divide by 1. to 255: Divide by 255.	0x00
31:8	-	Reserved	0x00

3.5.21 USB clock source select register

This register selects the clock source for the USB usb_clk. The clock source can be either the USB PLL output or the main clock, and the clock can be further divided by the USBCLKDIV register (see Table 29) to obtain a 48 MHz clock.

The USBCLKUEN register (see Section 3.5.22) must be toggled from LOW to HIGH for the update to take effect.

Remark: When switching clock sources, both clocks must be running before the clock source is updated. The default clock source for the USB controller is the USB PLL output. For switching the clock source to the main clock, ensure that the system PLL and the USB PLL are running to make both clock sources available for switching. The main clock must be set to 48 MHz and configured with the main PLL and the system oscillator. After the switch, the USB PLL can be turned off.

Table 27. USB clock source select register (USBCLKSEL, address 0x4004 80C0) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		USB clock source. Values 0x2 and 0x3 are reserved.	0x00
		0x0	USB PLL out	
		0x1	Main clock	
31:2	-	-	Reserved	0x00

3.5.22 USB clock source update enable register

This register updates the clock source of the USB with the new input clock after the USBCLKSEL register has been written to. In order for the update to take effect, first write a zero to the USBCLKUEN register and then write a one to USBCLKUEN.

Remark: When switching clock sources, both clocks must be running before the clock source is updated.

Table 28. USB clock source update enable register (USBCLKUEN, address 0x4004 80C4) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA		Enable USB clock source update	0x0
		0	No change	
		1	Update clock source	
31:1	-	-	Reserved	0x00

3.5.23 USB clock divider register

This register allows the USB clock usb_clk to be divided to 48 MHz. The usb_clk can be shut down by setting the DIV bits to 0x0.

Table 29. USB clock divider register (USBCLKDIV, address 0x4004 80C8) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	USB clock divider values 0: Disable USB clock. 1: Divide by 1. to 255: Divide by 255.	0x01
31:8	-	Reserved	0x00

3.5.24 CLKOUT clock source select register

This register selects the signal visible on the CLKOUT pin. Any oscillator or the main clock can be selected.

Bit 0 of the CLKOUTUEN register (see Section 3.5.25) must be toggled from 0 to 1 for the update to take effect.

Table 30. CLKOUT clock source select register (CLKOUTSEL, address 0x4004 80E0) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		CLKOUT clock source	0
		0x0	IRC oscillator	
		0x1	Crystal oscillator (SYSOSC)	
		0x2	LF oscillator	
		0x3	Main clock	
31:2	-	-	Reserved	0

3.5.25 CLKOUT clock source update enable register

This register updates the clock source of the CLKOUT pin with the new clock after the CLKOUTSEL register has been written to. In order for the update to take effect at the input of the CLKOUT pin, first write a zero to bit 0 of this register, then write a one.

Table 31. CLKOUT clock source update enable register (CLKOUTUEN, address 0x4004 80E4) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA		Enable CLKOUT clock source update	0
		0	No change	
		1	Update clock source	
31:1	-	-	Reserved	-

3.5.26 CLKOUT clock divider register

This register determines the divider value for the signal on the CLKOUT pin.

Table 32. CLKOUT clock divider registers (CLKOUTDIV, address 0x4004 80E8) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	CLKOUT clock divider values 0: Disable CLKOUT clock divider. 1: Divide by 1. to 255: Divide by 255.	0
31:8	-	Reserved	-

3.5.27 POR captured PIO status register 0

The PIOPORCAP0 register captures the state of GPIO port 0 at power-on-reset. Each bit represents the reset state of one GPIO pin. This register is a read-only status register.

Table 33. POR captured PIO status register 0 (PIOPORCAP0, address 0x4004 8100) bit description

Bit	Symbol	Description	Reset value
31:0	PIOSTAT	State of P0_31 through P0_0 at power-on reset	Implementation dependent

3.5.28 POR captured PIO status register 1

The PIOPORCAP1 register captures the state of GPIO port 1 at power-on-reset. Each bit represents the reset state of one GPIO pin. This register is a read-only status register.

Table 34. POR captured PIO status register 1 (PIOPORCAP1, address 0x4004 8104) bit description

Bit	Symbol	Description	Reset value
31:0	-	State of P1_31 through P1_0 at power-on reset	Implementation dependent

3.5.29 BOD control register

The BOD control register selects four separate threshold values for sending a BOD interrupt to the NVIC and for forced reset. Reset and interrupt threshold values listed in Table 35 are typical values.

Both the BOD interrupt and the BOD reset, depending on the value of bit BODRSTENA in this register, can wake-up the chip from Sleep, Deep-sleep, and Power-down modes. See Section 3.9.

Table 35. BOD control register (BODCTRL, address 0x4004 8150) bit description

Bit	Symbol	Value	Description	Reset value
1:0	BODRSTLEV		BOD reset level	0
		0x0	Level 0: The reset assertion threshold voltage is 1.46 V; the reset de-assertion threshold voltage is 1.63 V.	
		0x1	Level 1: The reset assertion threshold voltage is 2.06 V; the reset de-assertion threshold voltage is 2.15 V.	
		0x2	Level 2: The reset assertion threshold voltage is 2.35 V; the reset de-assertion threshold voltage is 2.43 V.	
		0x3	Level 3: The reset assertion threshold voltage is 2.63 V; the reset de-assertion threshold voltage is 2.71 V.	
3:2	BODINTVAL		BOD interrupt level	0
		0x0	Level 0: The interrupt assertion threshold voltage is 1.65 V; the interrupt de-assertion threshold voltage is 1.80 V.	
		0x1	Level 1:The interrupt assertion threshold voltage is 2.22 V; the interrupt de-assertion threshold voltage is 2.35 V.	
		0x2	Level 2: The interrupt assertion threshold voltage is 2.52 V; the interrupt de-assertion threshold voltage is 2.66 V.	
		0x3	Level 3: The interrupt assertion threshold voltage is 2.80 V; the interrupt de-assertion threshold voltage is 2.90 V.	
4	BODRSTENA		BOD reset enable	0
		0	Disable reset function.	
		1	Enable reset function.	
31:5	-	-	Reserved	0x00

3.5.30 System tick counter calibration register

This register determines the value of the SYST_CALIB register (see Table 322).

Table 36. System tick timer calibration register (SYSTCKCAL, address 0x4004 8154) bit description

Bit	Symbol	Description	Reset value
25:0	CAL	System tick timer calibration value	0
31:26	-	Reserved	-

3.5.31 IRQ latency register

The IRQLATENCY register is an eight-bit register which specifies the minimum number of cycles (0-255) permitted for the system to respond to an interrupt request. The intent of this register is to allow the user to select a trade-off between interrupt response time and determinism.

Setting this parameter to a very low value (e.g. zero) will guarantee the best possible interrupt performance but will also introduce a significant degree of uncertainty and jitter. Requiring the system to always take a larger number of cycles (whether it needs it or not) will reduce the amount of uncertainty but may not necessarily eliminate it.

Theoretically, the ARM Cortex-M0 core should always be able to service an interrupt request within 15 cycles. System factors external to the cpu, however, bus latencies, peripheral response times, etc. can increase the time required to complete a previous instruction before an interrupt can be serviced. Therefore, accurately specifying a minimum number of cycles that will ensure determinism will depend on the application.

The default setting for this register is 0x010.

Table 37. IRQ latency register (IRQLATENCY, address 0x4004 8170) bit description

Bit	Symbol	Description	Reset value
7:0	LATENCY	8-bit latency value	0x010
31:8	-	Reserved	-

3.5.32 NMI source selection register

The NMI source selection register selects a peripheral interrupts as source for the NMI interrupt of the ARM Cortex-M0 core. For a list of all peripheral interrupts and their IRQ numbers see Table 58. For a description of the NMI functionality, see Section 23.3.3.2.

Table 38. NMI source selection register (NMISRC, address 0x4004 8174) bit description

Bit	Symbol	Description	Reset value
4:0	IRQNO	The IRQ number of the interrupt that acts as the Non-Maskable Interrupt (NMI) if bit 31 is 1. See Table 58 for the list of interrupt sources and their IRQ numbers.	0
30:5	-	Reserved	-
31	NMIEN	Write a 1 to this bit to enable the Non-Maskable Interrupt (NMI) source selected by bits 4:0.	0

Note: If the NMISRC register is used to select an interrupt as the source of Non-Maskable interrupts, and the selected interrupt is enabled, one interrupt request can result in both a Non-Maskable and a normal interrupt. This can be avoided by disabling the normal interrupt in the NVIC, as described in Section 23.5.2.

3.5.33 Pin interrupt select registers

Each of these 8 registers selects one GPIO pin from all GPIO pins on both ports as the source of a pin interrupt. To select a pin for any of the eight pin interrupts, write the pin number as 0 to 23 for pins PIO0_0 to PIO0_23 and 24 to 55 for pins PIO1_0 to PIO1_31 to the INTPIN bits. For example, setting INTPIN to 0x5 in PINTSEL0 selects pin PIO0_5 for pin interrupt 0. Setting INTPIN in PINTSEL7 to 0x32 (pin 50) selects pin PIO1_26 for pin interrupt 7.

Each of the 8 pin interrupts must be enabled in the NVIC using interrupt slots # 0 to 7 (see Table 58).

To enable each pin interrupt and configure its edge or level sensitivity, use the GPIO pin interrupt registers (see Section 9.5.1).

Table 39. Pin interrupt select registers (PINTSEL0 to 7, address 0x4004 8178 to 0x4004 8194) bit description

Bit	Symbol	Description	Reset value
5:0	INTPIN	Pin number select for pin interrupt. (P0_0 to P0_23 correspond to numbers 0 to 23 and PIO1_0 to PIO1_31 correspond to numbers 24 to 55).	0
31:6	-	Reserved	-

3.5.34 USB clock control register

This register controls the use of the USB need_clock signal and the polarity of the need_clock signal for triggering the USB wake-up interrupt. For details of how to use the USB need_clock signal for waking up the part from Deep-sleep or Power-down modes, see Section 11.7.6.

Table 40. USB clock control register (USBCLKCTRL, address 0x4004 8198) bit description

Bit	Symbol	Value	Description	Reset value
0	AP_CLK		USB need_clock signal control	0
		0	Under hardware control.	
		1	Forced HIGH.	
1	POL_CLK		USB need_clock polarity for triggering the USB wake-up interrupt	0
		0	Falling edge of the USB need_clock triggers the USB wake-up (default).	
		1	Rising edge of the USB need_clock triggers the USB wake-up.	
31:2	-	-	Reserved	0x00

3.5.35 USB clock status register

This register is read-only and returns the status of the USB need_clock signal. For details of how to use the USB need_clock signal for waking up the part from Deep-sleep or Power-down modes, see Section 11.7.6.

Table 41. USB clock status register (USBCLKST, address 0x4004 819C) bit description

Bit	Symbol	Value	Description	Reset value
0	NEED_CLKST		USB need_clock signal status	0
		0	LOW	
		1	HIGH	
31:1	-	-	Reserved	0x00

3.5.36 Interrupt wake-up enable register 0

LPC11Uxx

The STARTERP0 register enables the individual GPIO pins selected through the Pin interrupt select registers (see Table 39) for wake-up. The pin interrupts must also be enabled in the NVIC (interrupts 0 to 8 in Table 58).

Table 42. Interrupt wake-up enable register 0 (STARTERP0, address 0x4004 8204) bit description

Bit	Symbol	Value	Description	Reset value
0	PINT0		Pin interrupt 0 wake-up	0
		0	Disabled	
		1	Enabled	
1	PINT1		Pin interrupt 1 wake-up	0
		0	Disabled	
		1	Enabled	
2	PINT2		Pin interrupt 2 wake-up	0
		0	Disabled	
		1	Enabled	
3	PINT3		Pin interrupt 3 wake-up	0
		0	Disabled	
		1	Enabled	
4	PINT4		Pin interrupt 4 wake-up	0
		0	Disabled	
		1	Enabled	
5	PINT5		Pin interrupt 5 wake-up	0
		0	Disabled	
		1	Enabled	
6	PINT6		Pin interrupt 6 wake-up	0
		0	Disabled	
		1	Enabled	
7	PINT7		Pin interrupt 7 wake-up	0
		0	Disabled	
		1	Enabled	
31:8	-		Reserved	-

3.5.37 Interrupt wake-up enable register 1

This register selects which interrupts will wake the LPC11Uxx from deep-sleep and power-down modes. Interrupts selected by a one in these registers must be enabled in the NVIC (Table 58) in order to successfully wake the LPC11Uxx from deep-sleep or power-down mode.

The STARTERP1 register enables the WWDT interrupt, the BOD interrupt, the USB wake-up interrupt and the two GPIO group interrupts for wake-up.

Table 43. Interrupt wake-up enable register 1 (STARTERP1, address 0x4004 8214) bit description

Bit	Symbol	Value	Description	Reset value
11:0			Reserved.	-
12	WWDTINT		WWDT interrupt wake-up	0
		0	Disabled	
		1	Enabled	
13	BODINT		Brown Out Detect (BOD) interrupt wake-up	0
		0	Disabled	
		1	Enabled	
18:14	-		Reserved	-
19	USB_WAKEUP		USB need_clock signal wake-up	0
		0	Disabled	
		1	Enabled	
20	GPIOINT0		GPIO GROUP0 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
21	GPIOINT1		GPIO GROUP1 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
31:22			Reserved.	-

3.5.38 Deep-sleep mode configuration register

The bits in this register (BOD_PD and WDTOSC_OD) can be programmed to control aspects of Deep-sleep and Power-down modes. The bits are loaded into corresponding bits of the PDRUNCFG register when Deep-sleep mode or Power-down mode is entered.

Remark: Hardware forces the analog blocks to be powered down in Deep-sleep and Power-down modes according to the power configuration described in Section 3.9.4.1 and Section 3.9.5.1.An exception are the exception of BOD and watchdog oscillator, which can be configured to remain running through this register. The WDTOSC_PD value written to the PDSLEEPCFG register is overwritten if the LOCK bit in the WWDT MOD register (see Table 310) is set. See Section 17.7 for details.

Table 44. Deep-sleep configuration register (PDSLEEPCFG, address 0x4004 8230) bit description

Bit	Symbol	Value	Description	Reset value
2:0			Reserved.	111
3	BOD_PD		BOD power-down control for Deep-sleep and Power-down mode	1
		0	Powered	
		1	Powered down	
5:4			Reserved.	11

Table 44. Deep-sleep configuration register (PDSLEEPCFG, address 0x4004 8230) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
6	WDTOSC_PD		Watchdog oscillator power-down control for Deep-sleep and Power-down mode	1
		0	Powered	
		1	Powered down	
31:7	-	-	Reserved	-

3.5.39 Wake-up configuration register

This register controls the power configuration of the device when waking up from Deep-sleep or Power-down mode.

Table 45. Wake-up configuration register (PDAWAKECFG, address 0x4004 8234) bit description

Bit	Symbol	Value	Description	Reset value
0	IRCOUT_PD		IRC oscillator output wake-up configuration	0
		0	Powered	
		1	Powered down	
1	IRC_PD		IRC oscillator power-down wake-up configuration	0
		0	Powered	
		1	Powered down	
2	FLASH_PD		Flash wake-up configuration	0
		0	Powered	
		1	Powered down	
3	BOD_PD		BOD wake-up configuration	0
		0	Powered	
		1	Powered down	
4	ADC_PD		ADC wake-up configuration	1
		0	Powered	
		1	Powered down	
5	SYSOSC_PD		Crystal oscillator wake-up configuration	1
		0	Powered	
		1	Powered down	
6	WDTOSC_PD		Watchdog oscillator wake-up configuration	1
		0	Powered	
		1	Powered down	
7	SYSPLL_PD		System PLL wake-up configuration	1
		0	Powered	
		1	Powered down	
8	USBPLL_PD		USB PLL wake-up configuration	1
		0	Powered	
		1	Powered down	

Table 45. Wake-up configuration register (PDAWAKECFG, address 0x4004 8234) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
9	-	0	Reserved.	0
10	USBPAD_PD		USB transceiver wake-up configuration	1
		0	USB transceiver powered	
		1	USB transceiver powered down	
11	-		Reserved. **Always write this bit as 1.**	1
12	-		Reserved. **Always write this bit as 0.**	0
15:13	-		Reserved. **Always write these bits as 111.**	111
31:16	-	-	Reserved	-

3.5.40 Power configuration register

The PDRUNCFG register controls the power to the various analog blocks. This register can be written to at any time while the chip is running, and a write will take effect immediately with the exception of the power-down signal to the IRC.

To avoid glitches when powering down the IRC, the IRC clock is automatically switched off at a clean point. Therefore, for the IRC a delay is possible before the power-down state takes effect.

Table 46. Power configuration register (PDRUNCFG, address 0x4004 8238) bit description

Bit	Symbol	Value	Description	Reset value
0	IRCOUT_PD		IRC oscillator output power-down	0
		0	Powered	
		1	Powered down	
1	IRC_PD		IRC oscillator power-down	0
		0	Powered	
		1	Powered down	
2	FLASH_PD		Flash power-down	0
		0	Powered	
		1	Powered down	
3	BOD_PD		BOD power-down	0
		0	Powered	
		1	Powered down	
4	ADC_PD		ADC power-down	1
		0	Powered	
		1	Powered down	
5	SYSOSC_PD		Crystal oscillator power-down	1
		0	Powered	
		1	Powered down	

Table 46. Power configuration register (PDRUNCFG, address 0x4004 8238) bit description

Bit	Symbol	Value	Description	Reset value
6	WDTOSC_PD		Watchdog oscillator power-down	1
		0	Powered	
		1	Powered down	
7	SYSPLL_PD		System PLL power-down	1
		0	Powered	
		1	Powered down	
8	USBPLL_PD		USB PLL power-down	1
		0	Powered	
		1	Powered down	
9	-	0	Reserved. **Always write this bit as 0.**	0
10	USBPAD_PD		USB transceiver power-down configuration	1
		0	USB transceiver powered	
		1	USB transceiver powered down (suspend mode)	
11	-		Reserved. **Always write this bit as 1.**	1
12	-		Reserved. **Always write this bit as 0.**	0
15:13	-		Reserved. **Always write these bits as 111.**	111
31:16	-	-	Reserved	-

3.5.41 Device ID register

This device ID register is a read-only register and contains the part ID for each LPC11Uxx part. This register is also read by the ISP/IAP commands (see Table 347).

Table 47. Device ID register (DEVICE_ID, address 0x4004 83F4) bit description

Bit	Symbol	Description	Reset value
31:0	DEVICEID	Device ID numbers for LPC11Uxx parts LPC11U12FHN33/201 = 0x095C 802B/0x295C 802B LPC11U12FBD48/201 = 0x095C 802B/0x295C 802B LPC11U13FBD48/201 = 0x097A 802B/0x297A 802B LPC11U14FHN33/201 = 0x0998 802B/0x2998 802B LPC11U14FHI33/201 = 0x2998 802B LPC11U14FBD48/201 = 0x0998 802B/0x2998 802B LPC11U14FET48/201 = 0x0998 802B/0x2998 802B LPC11U23FBD48/301 = 0x2972 402B LPC11U24FHI33/301 = 0x2988 402B LPC11U24FBD48/301 = 0x2988 402B LPC11U24FET48/301 = 0x2988 402B LPC11U24FHN33/401 = 0x2980 002B LPC11U24FBD48/401 = 0x2980 002B LPC11U24FBD64/401 = 0x2980 002B	part-dependent

3.5.42 Flash memory access

Depending on the system clock frequency, access to the flash memory can be configured with various access times by writing to the FLASHCFG register at address 0x4003 C010. This register is part of the flash configuration block (see Figure 3).

Remark: Improper setting of this register may result in incorrect operation of the LPC11Uxx.

Table 48. Flash configuration register (FLASHCFG, address 0x4003 C010) bit description

Bit	Symbol	Value	Description	Reset value
1:0	FLASHTIM		Flash memory access time. FLASHTIM +1 is equal to the number of system clocks used for flash access.	0x2
		0x0	1 system clock flash access time (for system clock frequencies of up to 20 MHz).	
		0x1	2 system clocks flash access time (for system clock frequencies of up to 40 MHz).	
		0x2	3 system clocks flash access time (for system clock frequencies of up to 50 MHz).	
		0x3	Reserved.	
31:2	-	-	Reserved. **User software must not change the value of these bits. Bits 31:2 must be written back exactly as read**.	-

3.6 Reset

Reset has four sources on the LPC11Uxx: the $\overline{\text{RESET}}$ pin, Watchdog Reset, Power-On Reset (POR), and Brown Out Detect (BOD). In addition, there is an ARM software reset.

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 IRC causing reset to remain asserted until the external Reset is de-asserted, the oscillator is running, and the flash controller has completed its initialization.

On the assertion of any reset source (Arm software reset, POR, BOD reset, External reset, and Watchdog reset), the following processes are initiated:

1. The IRC starts up. After the IRC-start-up time (maximum of 6 μs on power-up), the IRC provides a stable clock output.
2. The boot code in the ROM starts. The boot code performs the boot tasks and may jump to the flash.
3. The flash is powered up. This takes approximately 100 μs. Then the flash initialization sequence is started, which takes about 250 cycles.

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.

3.7 Start-up behavior

See Figure 6 for the start-up timing after reset. The IRC is the default clock at Reset and provides a clean system clock shortly after the supply voltage reaches the threshold value of 1.8 V.

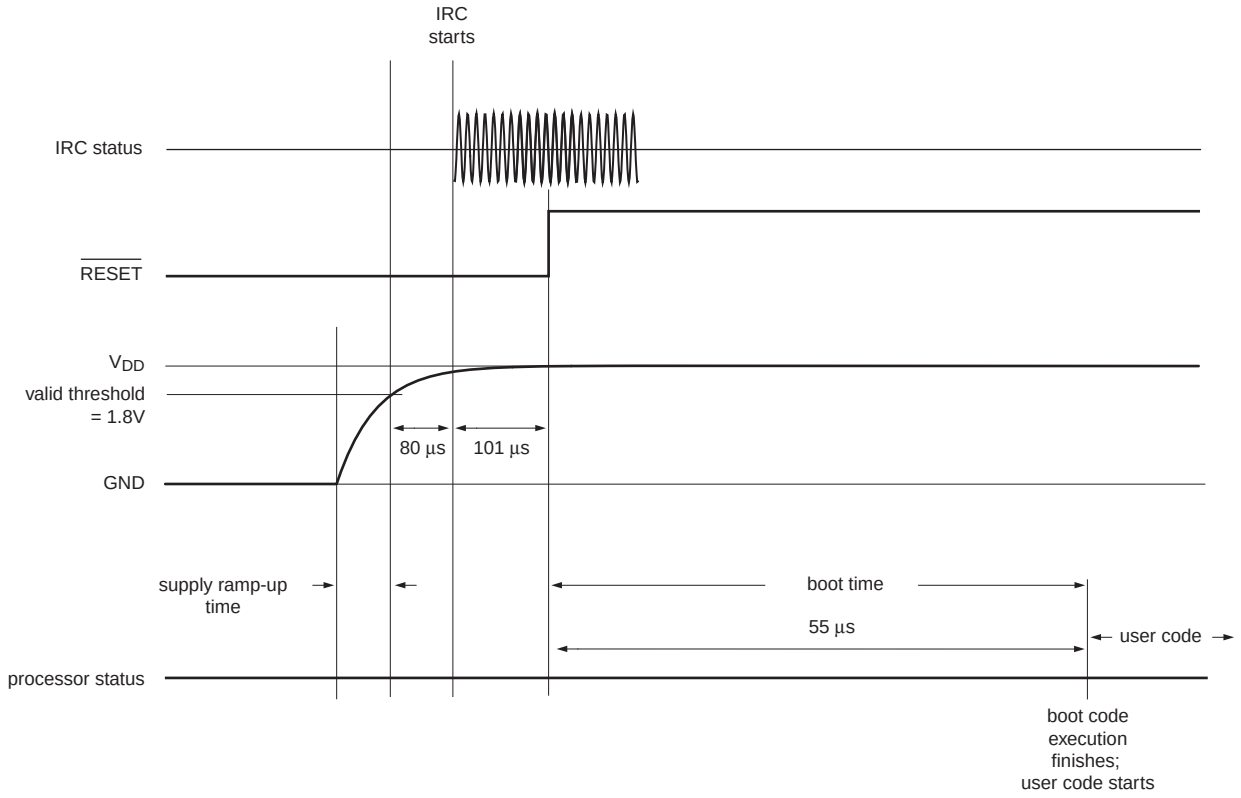


Fig 6. Start-up timing

3.8 Brown-out detection

The LPC11Uxx includes four levels for monitoring the voltage on the V_{DD} pin. If this voltage falls below one of the four selected levels, the BOD asserts an interrupt signal to the NVIC or issues a reset, depending on the value of the BODRSTENA bit in the BOD control register (Table 35).

The interrupt signal can be enabled for interrupt in the Interrupt Enable Register in the NVIC (see Table 406) in order to cause a CPU interrupt; if not, software can monitor the signal by reading a dedicated status register.

If the BOD interrupt is enabled in the STARTERP1 register (see Table 43) and in the NVIC, the BOD interrupt can wake up the chip from Deep-sleep and power-down mode.

If the BOD reset is enabled, the forced BOD reset can wake up the chip from Deep-sleep or Power-down mode.

3.9 Power management

The LPC11Uxx support a variety of power control features. In Active mode, when the chip is running, power and clocks to selected peripherals can be optimized for power consumption. In addition, there are four special modes of processor power reduction with different peripherals running: Sleep mode, Deep-sleep mode, Power-down mode, and Deep power-down mode.

Table 49. Peripheral configuration in reduced power modes

Peripheral	Sleep mode	Deep-sleep mode	Power-down mode	Deep power-down mode
IRC	software configurable	on	off[1]	off
IRC output	software configurable	off[1]	off[1]	off
Flash	software configurable	on	off	off
BOD	software configurable	software configurable	software configurable	off
PLL	software configurable	off	off	off
SysOsc	software configurable	off	off	off
WDosc/WWDT	software configurable	software configurable	software configurable	off
ADC	software configurable	off	off	off
Digital peripherals	software configurable	off	off	off
USB	software configurable	off	off	off

[1] If bit 5, the clock source lock bit, in the WWDT MOD register is set and the IRC is selected as the WWDT clock source, the IRC and the IRC output are forced on during this mode (Table 315). This increases power consumption and may cause the part not to enter Power-down mode correctly. For details see Section 17.7.

Remark: The Debug mode is not supported in Sleep, Deep-sleep, Power-down, or Deep power-down modes.

3.9.1 Reduced power modes and WWDT lock features

The WWDT clock select lock feature influences the power consumption in any of the power modes because locking the WWDT clock source forces the selected WWDT clock source to be on independently of the Deep-sleep and Power-down mode software configuration through the PDSLEEPCFG register. For details see Section 17.7.

If the part uses Deep-sleep mode with the WWDT running, the watchdog oscillator is the preferred clock source as it minimizes power consumption. If the clock source is not locked, the watchdog oscillator must be powered by using the PDSLEEPCFG register. Alternatively, the IRC may be selected and locked in WWDT MOD register, which forces the IRC on during Deep-sleep mode.

If the part uses Power-down mode with the WWDT running, the watchdog oscillator must be selected as the clock source. If the clock source is not locked, the watchdog oscillator must be powered by using the PDSLEEPCFG register. Do not lock the clock source with the IRC selected.

3.9.2 Active mode

In Active mode, the ARM Cortex-M0 core and memories are clocked by the system clock, and peripherals are clocked by the system clock or a dedicated peripheral clock.

The chip is in Active mode after reset and the default power configuration is determined by the reset values of the PDRUNCFG and SYSAHBCLKCTRL registers. The power configuration can be changed during run time.

3.9.2.1 Power configuration in Active mode

Power consumption in Active mode is determined by the following configuration choices:

- The SYSAHBCLKCTRL register controls which memories and peripherals are running (Table 23).
- The power to various analog blocks (PLL, oscillators, the ADC, the BOD circuit, and the flash block) can be controlled at any time individually through the PDRUNCFG register (Table 46).
- The clock source for the system clock can be selected from the IRC (default), the system oscillator, or the watchdog oscillator (see Figure 5 and related registers).
- The system clock frequency can be selected by the SYSPLLCTRL (Table 9) and the SYSAHBCLKDIV register (Table 22).
- Selected peripherals (USART, SSP0/1, USB, CLKOUT) use individual peripheral clocks with their own clock dividers. The peripheral clocks can be shut down through the corresponding clock divider registers (Table 24 to Table 32).

3.9.3 Sleep mode

In Sleep mode, the system clock to the ARM Cortex-M0 core is stopped, and execution of instructions is suspended until either a reset or an interrupt occurs.

Peripheral functions, if selected to be clocked in the SYSAHBCLKCTRL register, 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 processor state and registers, peripheral registers, and internal SRAM values are maintained, and the logic levels of the pins remain static.

3.9.3.1 Power configuration in Sleep mode

Power consumption in Sleep mode is configured by the same settings as in Active mode:

- The clock remains running.
- The system clock frequency remains the same as in Active mode, but the processor is not clocked.
- Analog and digital peripherals are selected as in Active mode.

3.9.3.2 Programming Sleep mode

The following steps must be performed to enter Sleep mode:

1. The PD bits in the PCON register must be set to the default value 0x0.
2. The SLEEPDEEP bit in the ARM Cortex-M0 SCR register must be set to zero.

3. Use the ARM Cortex-M0 Wait-For-Interrupt (WFI) instruction.

3.9.3.3 Wake-up from Sleep mode

Sleep mode is exited automatically when an interrupt enabled by the NVIC arrives at the processor or a reset occurs. After wake-up due to an interrupt, the microcontroller returns to its original power configuration defined by the contents of the PDRUNCFG and the SYSAHBCLKDIV registers. If a reset occurs, the microcontroller enters the default configuration in Active mode.

3.9.4 Deep-sleep mode

In Deep-sleep mode, the system clock to the processor is disabled as in Sleep mode. All analog blocks are powered down, except for the BOD circuit and the watchdog oscillator, which must be selected or deselected during Deep-sleep mode in the PDSLEEPCFG register. The main clock, and therefore all peripheral clocks, are disabled except for the clock to the watchdog timer if the watchdog oscillator is selected. The IRC is running, but its output is disabled. The flash is in stand-by mode.

Remark: If the LOCK bit is set in the WWDT MOD register (Table 310) and the IRC is selected as a clock source for the WWDT, the IRC continues to clock the WWDT in Deep-sleep mode.

Deep-sleep mode eliminates all power used by analog peripherals and all dynamic power used by the processor itself, memory systems and related controllers, and internal buses. The processor state and registers, peripheral registers, and internal SRAM values are maintained, and the logic levels of the pins remain static.

3.9.4.1 Power configuration in Deep-sleep mode

Power consumption in Deep-sleep mode is determined by the Deep-sleep power configuration setting in the PDSLEEPCFG (Table 44) register:

- The watchdog oscillator can be left running in Deep-sleep mode if required for the WWDT.
- If the IRC is locked as the WWDT clock source (see Section 17.7), the IRC continues to run and clock the WWDT in Deep-sleep mode independently of the setting in the PDSLEEPCFG register.
- The BOD circuit can be left running in Deep-sleep mode if required by the application.

3.9.4.2 Programming Deep-sleep mode

The following steps must be performed to enter Deep-sleep mode:

1. The PD bits in the PCON register must be set to 0x1 (Table 53).
2. Select the power configuration in Deep-sleep mode in the PDSLEEPCFG (Table 44) register.
3. Determine if the WWDT clock source must be locked to override the power configuration if the IRC is selected (see Section 17.7).
4. If the watchdog oscillator is shut down, ensure that the IRC is powered in the PDRUNCFG register and switch the clock source to IRC in the MAINCLKSEL register (Table 20). This ensures that the system clock is shut down glitch-free.

5. Select the power configuration after wake-up in the PDAWAKECFG (Table 45) register.
6. If any of the available wake-up interrupts are needed for wake-up, enable the interrupts in the interrupt wake-up registers (Table 42, Table 43) and in the NVIC.
7. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register.
8. Use the ARM WFI instruction.

3.9.4.3 Wake-up from Deep-sleep mode

The microcontroller can wake up from Deep-sleep mode in the following ways:

- Signal on one of the eight pin interrupts selected in Table 39. Each pin interrupt must also be enabled in the STARTERP0 register (Table 42) and in the NVIC.
- BOD signal, if the BOD is enabled in the PDSLEEPCFG register:
  - BOD interrupt using the deep-sleep interrupt wake-up register 1 (Table 43). The BOD interrupt must be enabled in the NVIC. The BOD interrupt must be selected in the BODCTRL register.
  - Reset from the BOD circuit. In this case, the BOD circuit must be enabled in the PDSLEEPCFG register, and the BOD reset must be enabled in the BODCTRL register (Table 35).
- WWDT signal, if the watchdog oscillator is enabled in the PDSLEEPCFG register:
  - WWDT interrupt using the interrupt wake-up register 1 (Table 43). The WWDT interrupt must be enabled in the NVIC. The WWDT interrupt must be set in the WWDT MOD register.
  - Reset from the watchdog timer. The WWDT reset must be set in the WWDT MOD register. In this case, the watchdog oscillator must be running in Deep-sleep mode (see PDSLEEPCFG register), and the WDT must be enabled in the SYSAHBCLKCTRL register.
- USB wake-up signal using the interrupt wake-up register 1 (Table 43). For details, see Section 11.7.6.
- GPIO group interrupt signal (see Table 43).

Remark: If the watchdog oscillator is running in Deep-sleep mode, its frequency determines the wake-up time.

3.9.5 Power-down mode

In Power-down mode, the system clock to the processor is disabled as in Sleep mode. All analog blocks are powered down, except for the BOD circuit and the watchdog oscillator, which must be selected or deselected during Power-down mode in the PDSLEEPCFG register. The main clock and therefore all peripheral clocks are disabled except for the clock to the watchdog timer if the watchdog oscillator is selected. The IRC itself and the flash are powered down, decreasing power consumption compared to Deep-sleep mode.

Remark: Do not set the LOCK bit in the WWDT MOD register (Table 310) when the IRC is selected as a clock source for the WWDT. This prevents the part from entering the Power-down mode correctly.

Power-down mode eliminates all power used by analog peripherals and all dynamic power used by the processor itself, memory systems and related controllers, and internal buses. The processor state and registers, peripheral registers, and internal SRAM values are maintained, and the logic levels of the pins remain static. Wake-up times are longer compared to the Deep-sleep mode.

3.9.5.1 Power configuration in Power-down mode

Power consumption in Power-down mode can be configured by the power configuration setting in the PDSLEEPCFG (Table 44) register in the same way as for Deep-sleep mode (see Section 3.9.4.1):

- The watchdog oscillator can be left running in Deep-sleep mode if required for the WWDT.
- The BOD circuit can be left running in Deep-sleep mode if required by the application.

3.9.5.2 Programming Power-down mode

The following steps must be performed to enter Power-down mode:

1. The PD bits in the PCON register must be set to 0x2 (Table 53).
2. Select the power configuration in Power-down mode in the PDSLEEPCFG (Table 44) register.
3. If the lock bit 5 in the WWDT MOD register is set (Table 310) and the IRC is selected as the WWDT clock source, reset the part to clear the lock bit and then select the watchdog oscillator as the WWDT clock source.
4. If the watchdog oscillator is shut down, ensure that the IRC is powered in the PDRUNCFG register and switch the clock source to IRC in the MAINCLKSEL register (Table 20). This ensures that the system clock is shut down glitch-free.
5. Select the power configuration after wake-up in the PDAWAKECFG (Table 45) register.
6. If any of the available wake-up interrupts are used for wake-up, enable the interrupts in the interrupt wake-up registers (Table 42, Table 43) and in the NVIC.
7. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register.
8. Use the ARM WFI instruction.

3.9.5.3 Wake-up from Power-down mode

The microcontroller can wake up from Power-down mode in the same way as from Deep-sleep mode:

- Signal on one of the eight pin interrupts selected in Table 39. Each pin interrupt must also be enabled in the STARTERP0 register (Table 42) and in the NVIC.
- BOD signal, if the BOD is enabled in the PDSLEEPCFG register:
  - BOD interrupt using the interrupt wake-up register 1 (Table 43). The BOD interrupt must be enabled in the NVIC. The BOD interrupt must be selected in the BODCTRL register.
  - Reset from the BOD circuit. In this case, the BOD reset must be enabled in the BODCTRL register (Table 35).
- WWDT signal, if the watchdog oscillator is enabled in the PDSLEEPCFG register:

- WWDT interrupt using the interrupt wake-up register 1 (Table 43). The WWDT interrupt must be enabled in the NVIC. The WWDT interrupt must be set in the WWDT MOD register.
- Reset from the watchdog timer.The WWDT reset must be set in the WWDT MOD register.

- USB wake-up signal interrupt wake-up register 1 (Table 43). For details, see Section 11.7.6.
- GPIO group interrupt signal (see Table 43).

3.9.6 Deep power-down mode

In Deep power-down mode, power and clocks are shut off to the entire chip with the exception of the WAKEUP pin. The Deep power-down mode is controlled by the PMU (see Chapter 4).

During Deep power-down mode, the contents of the SRAM and registers are not retained except for a small amount of data which can be stored in the general purpose registers of the PMU block.

All functional pins are tri-stated in Deep power-down mode except for the WAKEUP pin.

Remark: Setting bit 3 in the PCON register (Section 4.3.1) prevents the part from entering Deep-power down mode.

3.9.6.1 Power configuration in Deep power-down mode

Deep power-down mode has no configuration options. All clocks, the core, and all peripherals are powered down. Only the WAKEUP pin is powered.

3.9.6.2 Programming Deep power-down mode

The following steps must be performed to enter Deep power-down mode:

1. Pull the WAKEUP pin externally HIGH.
2. Ensure that bit 3 in the PCON register (Table 53) is cleared.
3. Write 0x3 to the PD bits in the PCON register (see Table 53).
4. Store data to be retained in the general purpose registers (Section 4.3.2).
5. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register.
6. Ensure that the IRC is powered by setting bits IRCOUT_PD and IRC_PD to zero in the PDRUNCFG register before entering Deep power-down mode.
7. Use the ARM WFI instruction.

3.9.6.3 Wake-up from Deep power-down mode

Pulling the WAKEUP pin LOW wakes up the LPC11Uxx from Deep power-down, and the chip goes through the entire reset process (Section 3.6).

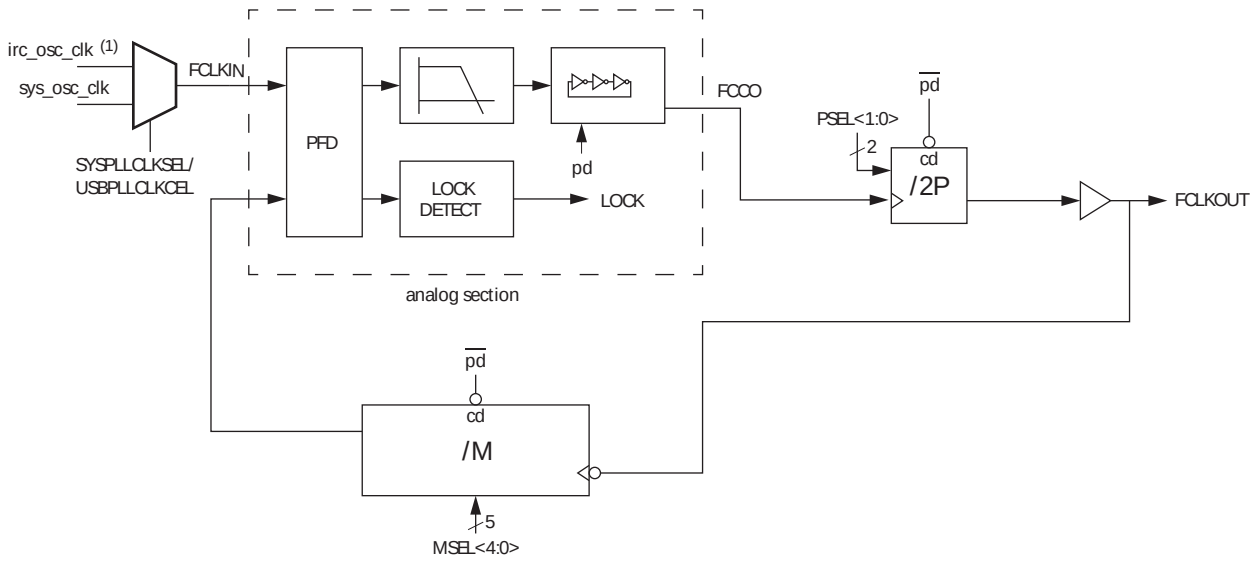
1. On the WAKEUP pin, transition from HIGH to LOW.
   - The PMU will turn on the on-chip voltage regulator. When the core voltage reaches the power-on-reset (POR) trip point, a system reset will be triggered and the chip re-boots.
   - All registers except the GPREG0 to GPREG4 will be in their reset state.

2. Once the chip has booted, read the deep power-down flag in the PCON register (Table 53) to verify that the reset was caused by a wake-up event from Deep power-down and was not a cold reset.
3. Clear the deep power-down flag in the PCON register (Table 53).
4. (Optional) Read the stored data in the general purpose registers (Section 4.3.2).
5. Set up the PMU for the next Deep power-down cycle.

Remark: The $\overline{\text{RESET}}$ pin has no functionality in Deep power-down mode.

3.10 System PLL/USB PLL functional description

The LPC11Uxx uses the system PLL to create the clocks for the core and peripherals. An identical PLL is available for the USB.



(1) System PLL only.

Fig 7. System PLL block diagram

The block diagram of this PLL is shown in Figure 7. The input frequency range is 10 MHz to 25 MHz. The input clock is fed directly to the Phase-Frequency Detector (PFD). This block compares the phase and frequency of its inputs, and generates a control signal when phase and/ or frequency do not match. The loop filter filters these control signals and drives the current controlled oscillator (CCO), which generates the main clock and optionally two additional phases. The CCO frequency range is 156 MHz to 320 MHz.These clocks are either divided by 2×P by the programmable post divider to create the output clocks, or are sent directly to the outputs. The main output clock is then divided by M by the programmable feedback divider to generate the feedback clock. The output signal of the phase-frequency detector is also monitored by the lock detector, to signal when the PLL has locked on to the input clock.

3.10.1 Lock detector

The lock detector measures the phase difference between the rising edges of the input and feedback clocks. Only when this difference is smaller than the so called “lock criterion” for more than eight consecutive input clock periods, the lock output switches from low to high. A single too large phase difference immediately resets the counter and causes the lock signal to drop (if it was high). Requiring eight phase measurements in a row to be below a certain figure ensures that the lock detector will not indicate lock until both the phase and frequency of the input and feedback clocks are very well aligned. This effectively prevents false lock indications, and thus ensures a glitch free lock signal.

3.10.2 Power-down control

To reduce the power consumption when the PLL clock is not needed, a Power-down mode has been incorporated. This mode is enabled by setting the SYSPLL_PD bit to one in the Power-down configuration register (Table 46). In this mode, the internal current reference will be turned off, the oscillator and the phase-frequency detector will be stopped and the dividers will enter a reset state. While in Power-down mode, the lock output will be low to indicate that the PLL is not in lock. When the Power-down mode is terminated by setting the SYSPLL_PD bit to zero, the PLL will resume its normal operation and will make the lock signal high once it has regained lock on the input clock.

3.10.3 Divider ratio programming

Post divider

The division ratio of the post divider is controlled by the PSEL bits. The division ratio is two times the value of P selected by PSEL bits as shown in Table 9 and Table 11. This guarantees an output clock with a 50% duty cycle.

Feedback divider

The feedback divider’s division ratio is controlled by the MSEL bits. The division ratio between the PLL’s output clock and the input clock is the decimal value on MSEL bits plus one, as specified in Table 9 and Table 11.

Changing the divider values

Changing the divider ratio while the PLL is running is not recommended. As there is no way to synchronize the change of the MSEL and PSEL values with the dividers, the risk exists that the counter will read in an undefined value, which could lead to unwanted spikes or drops in the frequency of the output clock. The recommended way of changing between divider settings is to power down the PLL, adjust the divider settings and then let the PLL start up again.

3.10.4 Frequency selection

The PLL frequency equations use the following parameters (also see Figure 5):

Table 50. PLL frequency parameters

Parameter	System PLL
FCLKIN	Frequency of sys_pllclkin (input clock to the system PLL) from the SYSPLLCLKSEL multiplexer (see Section 3.5.10).
FCCO	Frequency of the Current Controlled Oscillator (CCO); 156 to 320 MHz.
FCLKOUT	Frequency of sys_pllclkout
P	System PLL post divider ratio; PSEL bits in SYSPLLCTRL (see Section 3.5.3).
M	System PLL feedback divider register; MSEL bits in SYSPLLCTRL (see Section 3.5.3).

3.10.4.1 Normal mode

In this mode the post divider is enabled, giving a 50% duty cycle clock with the following frequency relations:

$$Fclkout = M \times Fclkin = (FCCO)/(2 \times P) \quad (1)$$

To select the appropriate values for M and P, it is recommended to follow these steps:

1. Specify the input clock frequency Fclkin.
2. Calculate M to obtain the desired output frequency Fclkout with M = F_{clkout} / F_{clkin}.
3. Find a value so that FCCO = 2 × P × F_{clkout}.
4. Verify that all frequencies and divider values conform to the limits specified in Table 9 and Table 11.

Table 51 shows how to configure the PLL for a 12 MHz crystal oscillator using the SYSPLLCTRL register (Table 9). The main clock is equivalent to the system clock if the system clock divider SYSAHBCLKDIV is set to one (see Table 22).

Table 51. PLL configuration examples

PLL input clock sys_pllclkin (Fclkin)	Main clock (Fclkout)	MSEL bits Table 9	M divider value	PSEL bits Table 9	P divider value	FCCO frequency
12 MHz	48 MHz	00011(binary)	4	01 (binary)	2	192 MHz
12 MHz	36 MHz	00010(binary)	3	10 (binary)	4	288 MHz
12 MHz	24 MHz	00001(binary)	2	10 (binary)	4	192 MHz

3.10.4.2 Power-down mode

In this mode, the internal current reference will be turned off, the oscillator and the phase-frequency detector will be stopped and the dividers will enter a reset state. While in Power-down mode, the lock output will be low, to indicate that the PLL is not in lock. When the Power-down mode is terminated by SYSPLL_PD bit to zero in the Power-down configuration register (Table 46), the PLL will resume its normal operation and will make the lock signal high once it has regained lock on the input clock.

4.1 How to read this chapter

The PMU is identical on all LPC11Uxx parts. Also refer to Chapter 5 for power control.

4.2 Introduction

The PMU controls the Deep power-down mode. Four general purpose register in the PMU can be used to retain data during Deep power-down mode.

4.3 Register description

Table 52. Register overview: PMU (base address 0x4003 8000)

Name	Access	Address offset	Description	Reset value	Reference
PCON	R/W	0x000	Power control register	0x0	Table 53
GPREG0	R/W	0x004	General purpose register 0	0x0	Table 54
GPREG1	R/W	0x008	General purpose register 1	0x0	Table 54
GPREG2	R/W	0x00C	General purpose register 2	0x0	Table 54
GPREG3	R/W	0x010	General purpose register 3	0x0	Table 54
GPREG4	R/W	0x014	General purpose register 4	0x0	Table 55

4.3.1 Power control register

The power control register selects whether one of the ARM Cortex-M0 controlled power-down modes (Sleep mode or Deep-sleep/Power-down mode) or the Deep power-down mode is entered and provides the flags for Sleep or Deep-sleep/Power-down modes and Deep power-down modes respectively. See Section 3.9 for details on how to enter the power-down modes.

Table 53. Power control register (PCON, address 0x4003 8000) bit description

Bit	Symbol	Value	Description	Reset value
2:0	PM		Power mode	000
		0x0	Default. The part is in active or sleep mode.	
		0x1	ARM WFI will enter Deep-sleep mode.	
		0x2	ARM WFI will enter Power-down mode.	
		0x3	ARM WFI will enter Deep-power down mode (ARM Cortex-M0 core powered-down).	
3	NODPD		A 1 in this bit prevents entry to Deep power-down mode when 0x3 is written to the PM field above, the SLEEPDEEP bit is set, and a WFI is executed. Execution continues after the WFI if this bit is 1. This bit is cleared only by power-on reset, so writing a one to this bit locks the part in a mode in which Deep power-down mode is blocked.	0

Table 53. Power control register (PCON, address 0x4003 8000) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
7:4	-	-	Reserved. Do not write ones to this bit.	0
8	SLEEPFLAG		Sleep mode flag	0
		0	Read: No power-down mode entered. LPC11Uxx is in Active mode. Write: No effect.	
		1	Read: Sleep/Deep-sleep or Deep power-down mode entered. Write: Writing a 1 clears the SLEEPFLAG bit to 0.	
10:9	-	-	Reserved. Do not write ones to this bit.	0
11	DPDFLAG		Deep power-down flag	0
		0	Read: Deep power-down mode **not** entered. Write: No effect.	0
		1	Read: Deep power-down mode entered. Write: Clear the Deep power-down flag.	
31:12	-	-	Reserved. Do not write ones to this bit.	0

4.3.2 General purpose registers 0 to 3

The general purpose registers retain data through the Deep power-down mode when power is still applied to the V_{DD} pin but the chip has entered Deep power-down mode. Only a “cold” boot when all power has been completely removed from the chip will reset the general purpose registers.

Table 54. General purpose registers 0 to 3 (GPREG0 - GPREG3, address 0x4003 8004 to 0x4003 8010) bit description

Bit	Symbol	Description	Reset value
31:0	GPDATA	Data retained during Deep power-down mode.	0x0

4.3.3 General purpose register 4

The general purpose register 4 retains data through the Deep power-down mode when power is still applied to the V_{DD} pin but the chip has entered Deep power-down mode. Only a “cold” boot, when all power has been completely removed from the chip, will reset the general purpose registers.

Remark: If there is a possibility that the external voltage applied on pin V_{DD} drops below 2.2 V during Deep power-down, the hysteresis of the WAKEUP input pin has to be disabled in this register before entering Deep power-down mode in order for the chip to wake up.

Table 55. General purpose register 4 (GPREG4, address 0x4003 8014) bit description

Bit	Symbol	Value	Description	Reset value
9:0	-	-	Reserved. Do not write ones to this bit.	0x0

Table 55. General purpose register 4 (GPREG4, address 0x4003 8014) bit description

Bit	Symbol	Value	Description	Reset value
10	WAKEUPHYS		WAKEUP pin hysteresis enable	0x0
		0	Hysteresis for WAKUP pin disabled.	
		1	Hysteresis for WAKEUP pin enabled.	
31:11	GPDATA		Data retained during Deep power-down mode.	0x0

4.4 Functional description

For details of entering and exiting reduced power modes, see Section 3.9.

5.1 How to read this chapter

The power profiles are available for all LPC11Uxx.

5.2 Features

- Includes ROM-based application services
- Power Management services
- Clocking services

5.3 Description

The power consumption in Active and Sleep modes can be optimized for the application through simple calls to the power profile. The power configuration routine configures the LPC11Uxx for one of the following power modes:

- Default mode corresponding to power configuration after reset.
- CPU performance mode corresponding to optimized processing capability.
- Efficiency mode corresponding to optimized balance of current consumption and CPU performance.
- Low-current mode corresponding to lowest power consumption.

In addition, the power profile includes routines to select the optimal PLL settings for a given system clock and PLL input clock.

Remark: When using the USB, configure the LPC11Uxx in Default mode.

The API calls to the ROM are performed by executing functions which are pointed by a pointer within the ROM Driver Table. Figure 8 shows the pointer structure used to call the Power Profiles API.

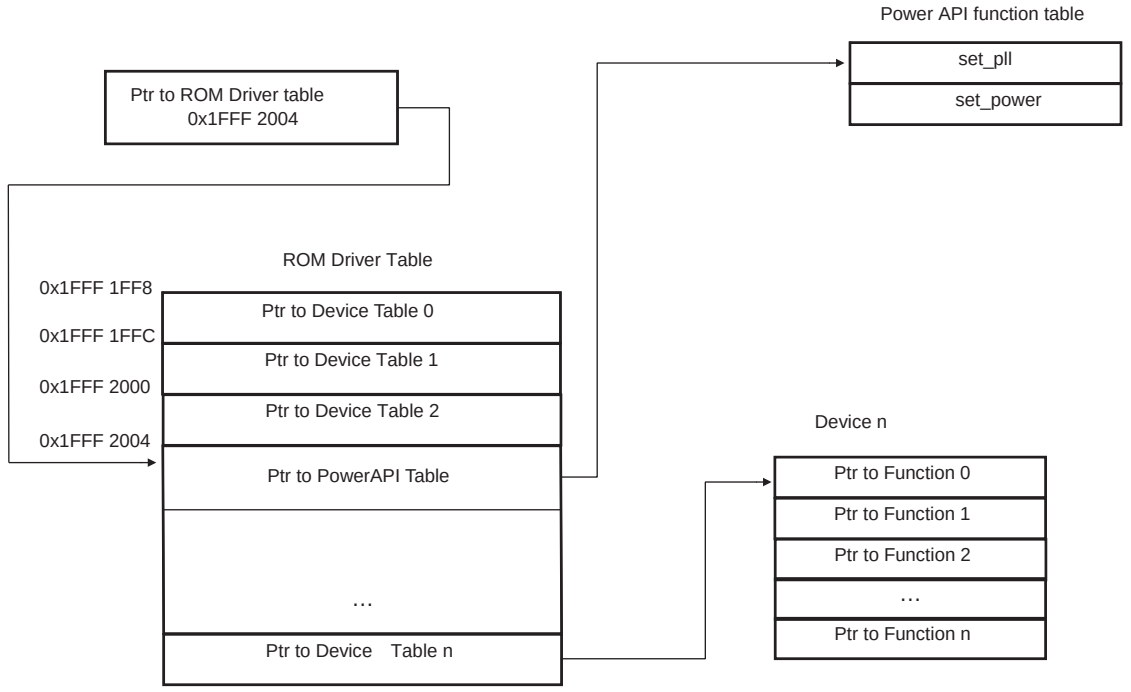


Fig 8. Power profiles pointer structure

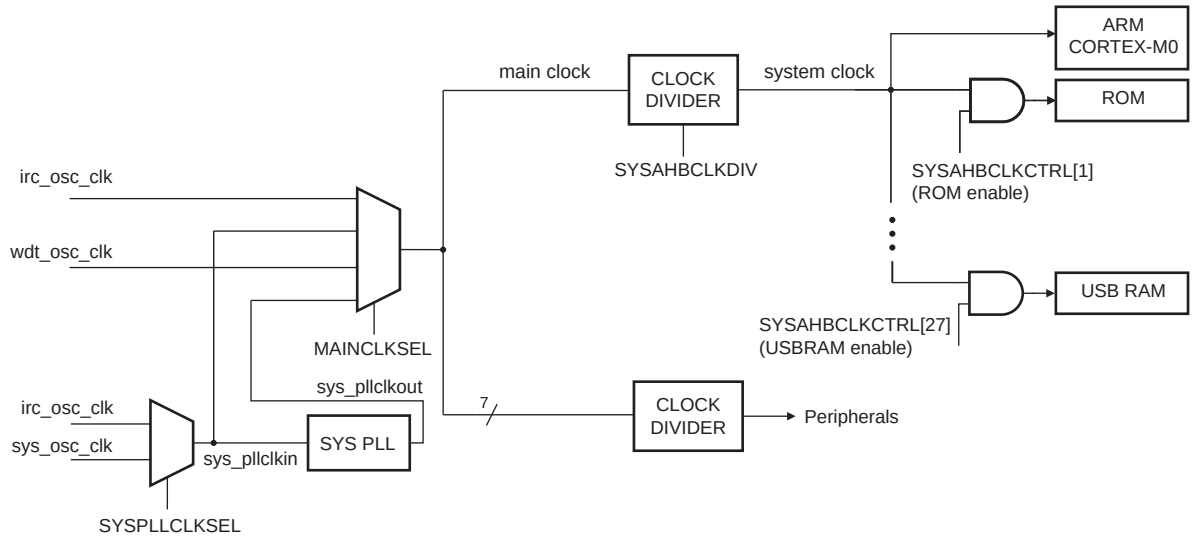


Fig 9. LPC11Uxx clock configuration for power API use

5.4 Definitions

The following elements have to be defined in an application that uses the power profiles:

```
typedef struct _PWRD {
        void (*set_pll)(unsigned int cmd[], unsigned int resp[]);
        void (*set_power)(unsigned int cmd[], unsigned int resp[]);
} PWRD;
typedef struct _ROM {
        const PWRD * pWRD;
} ROM;
ROM ** rom = (ROM **) (0x1FFF1FF8 + 3 * sizeof(ROM**));
unsigned int command[4], result[2];
```

5.5 Clocking routine

5.5.1 set_pll

This routine sets up the system PLL according to the calling arguments. If the expected clock can be obtained by simply dividing the system PLL input, *set_pll* bypasses the PLL to lower system power consumption.

Remark: Before this routine is invoked, the PLL clock source (IRC/system oscillator) must be selected (Table 16), the main clock source must be set to the input clock to the system PLL (Table 18) and the system/AHB clock divider must be set to 1 (Table 20).

set_pll attempts to find a PLL setup that matches the calling parameters. Once a combination of a feedback divider value (SYSPLLCTRL, M), a post divider ratio (SYSPLLCTRL, P) and the system/AHB clock divider (SYSAHBCLKDIV) is found, *set_pll* applies the selected values and switches the main clock source selection to the system PLL clock out (if necessary).

The routine returns a result code that indicates if the system PLL was successfully set (PLL_CMD_SUCCESS) or not (in which case the result code identifies what went wrong). The current system frequency value is also returned. The application should use this information to adjust other clocks in the device (the SSP, UART, and WDT clocks, and/or clockout).

Table 56. set_pll routine

Routine	set_pll
Input	**Param0:** system PLL input frequency (in kHz)
	Param1: expected system clock (in kHz)
	Param2: mode (CPU_FREQ_EQU, CPU_FREQ_LTE, CPU_FREQ_GTE, CPU_FREQ_APPROX)
	Param3: system PLL lock time-out
Result	**Result0:** PLL_CMD_SUCCESS | PLL_INVALID_FREQ | PLL_INVALID_MODE | PLL_FREQ_NOT_FOUND | PLL_NOT_LOCKED
	Result1: system clock (in kHz)

The following definitions are needed when making set_pll power routine calls:

```
/* set_pll mode options */
#define     CPU_FREQ_EQU        0
#define     CPU_FREQ_LTE        1
#define     CPU_FREQ_GTE        2
#define     CPU_FREQ_APPROX     3
```

```
/* set_pll result0 options */
#define     PLL_CMD_SUCCESS      0
#define     PLL_INVALID_FREQ     1
#define     PLL_INVALID_MODE     2
#define     PLL_FREQ_NOT_FOUND   3
#define     PLL_NOT_LOCKED       4
```

For a simplified clock configuration scheme see Figure 9. For more details see Figure 5.

5.5.1.1 Param0: system PLL input frequency and Param1: expected system clock

set_pll looks for a setup in which the system PLL clock does not exceed 50 MHz. It easily finds a solution when the ratio between the expected system clock and the system PLL input frequency is an integer value, but it can also find solutions in other cases.

The system PLL input frequency (*Param0*) must be between 10000 to 25000 kHz (10 MHz to 25 MHz) inclusive. The expected system clock (*Param1*) must be between 1 and 50000 kHz inclusive. If either of these requirements is not met, *set_pll* returns PLL_INVALID_FREQ and returns *Param0* as *Result1* since the PLL setting is unchanged.

5.5.1.2 Param2: mode

The first priority of *set_pll* is to find a setup that generates the system clock at exactly the rate specified in *Param1*. If it is unlikely that an exact match can be found, input parameter mode (*Param2*) should be used to specify if the actual system clock can be less than or equal, greater than or equal or approximately the value specified as the expected system clock (*Param1*).

A call specifying CPU_FREQ_EQU will only succeed if the PLL can output exactly the frequency requested in *Param1*.

CPU_FREQ_LTE can be used if the requested frequency should not be exceeded (such as overall current consumption and/or power budget reasons).

CPU_FREQ_GTE helps applications that need a minimum level of CPU processing capabilities.

CPU_FREQ_APPROX results in a system clock that is as close as possible to the requested value (it may be greater than or less than the requested value).

If an illegal mode is specified, *set_pll* returns PLL_INVALID_MODE. If the expected system clock is out of the range supported by this routine, *set_pll* returns PLL_FREQ_NOT_FOUND. In these cases the current PLL setting is not changed and *Param0* is returned as *Result1*.

5.5.1.3 Param3: system PLL lock time-out

It should take no more than 100 μs for the system PLL to lock if a valid configuration is selected. If *Param3* is zero, *set_pll* will wait indefinitely for the PLL to lock. A non-zero value indicates how many times the code will check for a successful PLL lock event before it returns PLL_NOT_LOCKED. In this case the PLL settings are unchanged and *Param0* is returned as *Result1*.

Remark: The time it takes the PLL to lock depends on the selected PLL input clock source (IRC/system oscillator) and its characteristics. The selected source can experience more or less jitter depending on the operating conditions such as power

supply and/or ambient temperature. This is why it is suggested that when a good known clock source is used and a PLL_NOT_LOCKED response is received, the set_pll routine should be invoked several times before declaring the selected PLL clock source invalid.

Hint: setting *Param3* equal to the system PLL frequency [Hz] divided by 10000 will provide more than enough PLL lock-polling cycles.

5.5.1.4 Code examples

The following examples illustrate some of the features of *set_pll* discussed above.

5.5.1.4.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 12000;
command[1] = 60000;
command[2] = CPU_FREQ_EQU;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock and a system clock of exactly 60 MHz. The application was ready to infinitely wait for the PLL to lock. But the expected system clock of 60 MHz exceeds the maximum of 50 MHz. Therefore *set_pll* returns PLL_INVALID_FREQ in *result[0]* and 12000 in *result[1]* without changing the PLL settings.

5.5.1.4.2 Invalid frequency selection (system clock divider restrictions)

```
command[0] = 12000;
command[1] = 40;
command[2] = CPU_FREQ_LTE;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock, a system clock of no more than 40 kHz and no time-out while waiting for the PLL to lock. Since the maximum divider value for the system clock is 255 and running at 40 kHz would need a divide by value of 300, *set_pll* returns PLL_INVALID_FREQ in *result[0]* and 12000 in *result[1]* without changing the PLL settings.

5.5.1.4.3 Exact solution cannot be found (PLL)

```
command[0] = 12000;
command[1] = 25000;
command[2] = CPU_FREQ_EQU;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock and a system clock of exactly 25 MHz. The application was ready to infinitely wait for the PLL to lock. Since there is no valid PLL setup within earlier mentioned restrictions, *set_pll* returns PLL_FREQ_NOT_FOUND in *result[0]* and 12000 in *result[1]* without changing the PLL settings.

5.5.1.4.4 System clock less than or equal to the expected value

```
command[0] = 12000;
command[1] = 25000;
command[2] = CPU_FREQ_LTE;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock, a system clock of no more than 25 MHz and no locking time-out. *set_pll* returns PLL_CMD_SUCCESS in *result[0]* and 24000 in *result[1]*. The new system clock is 24 MHz.

5.5.1.4.5 System clock greater than or equal to the expected value

```
command[0] = 12000;
command[1] = 25000;
command[2] = CPU_FREQ_GTE;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock, a system clock of at least 25 MHz and no locking time-out. *set_pll* returns PLL_CMD_SUCCESS in *result[0]* and 36000 in *result[1]*. The new system clock is 36 MHz.

5.5.1.4.6 System clock approximately equal to the expected value

```
command[0] = 12000;
command[1] = 16500;
command[2] = CPU_FREQ_APPROX;
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

The above code specifies a 12 MHz PLL input clock, a system clock of approximately 16.5 MHz and no locking time-out. *set_pll* returns PLL_CMD_SUCCESS in *result[0]* and 16000 in *result[1]*. The new system clock is 16 MHz.

5.6 Power routine

5.6.1 set_power

This routine configures the device's internal power control settings according to the calling arguments. The goal is to reduce active power consumption while maintaining the feature of interest to the application close to its optimum.

Remark: The set_power routine was designed for systems employing the configuration of SYSAHBCLKDIV = 1 (System clock divider register, see Table 22 and Figure 9). Using this routine in an application with the system clock divider not equal to 1 might not improve microcontroller's performance as much as in setups when the main clock and the system clock are running at the same rate.

set_power returns a result code that reports whether the power setting was successfully changed or not.

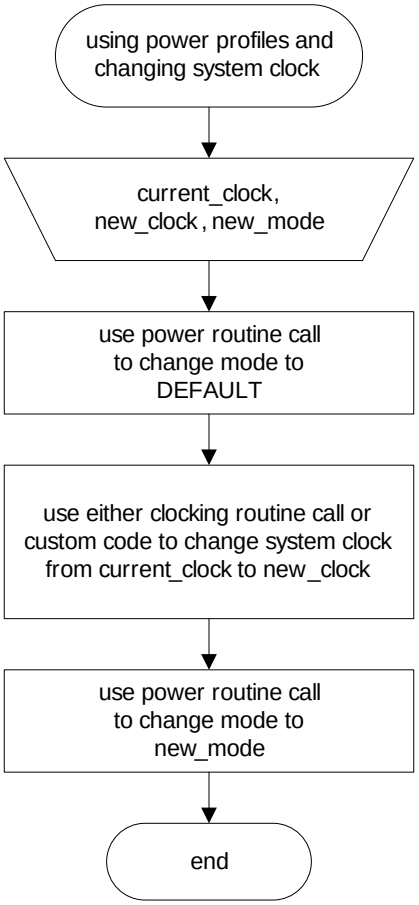


Fig 10. Power profiles usage

Table 57. set_power routine

Routine	set_power
Input	**Param0:** main clock (in MHz) **Param1:** mode (PWR_DEFAULT, PWR_CPU_PERFORMANCE, PWR_EFFICIENCY, PWR_LOW_CURRENT) **Param2:** system clock (in MHz)
Result	**Result0:** PWR_CMD_SUCCESS | PWR_INVALID_FREQ | PWR_INVALID_MODE

The following definitions are needed for set_power routine calls:

```
/* set_power mode options */
#define     PWR_DEFAULT          0
#define     PWR_CPU_PERFORMANCE  1
#define     PWR_EFFICIENCY       2
#define     PWR_LOW_CURRENT      3
/* set_power result0 options */
#define     PWR_CMD_SUCCESS      0
#define     PWR_INVALID_FREQ     1
#define     PWR_INVALID_MODE     2
```

For a simplified clock configuration scheme see Figure 9. For more details see Figure 5.

5.6.1.1 Param0: main clock

The main clock is the clock rate the microcontroller uses to source the system's and the peripherals' clock. It is configured by either a successful execution of the clocking routine call or a similar code provided by the user. This operand must be an integer between 1 to 50 MHz inclusive. If a value out of this range is supplied, *set_power* returns PWR_INVALID_FREQ and does not change the power control system.

5.6.1.2 Param1: mode

The input parameter mode (*Param1*) specifies one of four available power settings. If an illegal selection is provided, *set_power* returns PWR_INVALID_MODE and does not change the power control system.

PWR_DEFAULT keeps the device in a baseline power setting similar to its reset state.

PWR_CPU_PERFORMANCE configures the microcontroller so that it can provide more processing capability to the application. CPU performance is 30% better than the default option.

PWR_EFFICIENCY setting was designed to find a balance between active current and the CPU's ability to execute code and process data. In this mode the device outperforms the default mode both in terms of providing higher CPU performance and lowering active current.

PWR_LOW_CURRENT is intended for those solutions that focus on lowering power consumption rather than CPU performance.

5.6.1.3 Param2: system clock

The system clock is the clock rate at which the microcontroller core is running when *set_power* is called. This parameter is an integer between from 1 and 50 MHz inclusive.

5.6.1.4 Code examples

The following examples illustrate some of the *set_power* features discussed above.

5.6.1.4.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 60;
command[1] = PWR_CPU_PERFORMANCE;
command[2] = 60;
(*rom)->pWRD->set_power(command, result);
```

The above setup would be used in a system running at the main and system clock of 60 MHz, with a need for maximum CPU processing power. Since the specified 60 MHz clock is above the 50 MHz maximum, *set_power* returns PWR_INVALID_FREQ in *result[0]* without changing anything in the existing power setup.

5.6.1.4.2 An applicable power setup

```
command[0] = 24;
command[1] = PWR_CPU_EFFICIENCY;
command[2] = 24;
```

```
(*rom)->pWRD->set_power(command, result);
```

The above code specifies that an application is running at the main and system clock of 24 MHz with emphasis on efficiency. *set_power* returns PWR_CMD_SUCCESS in *result[0]* after configuring the microcontroller's internal power control features.

6.1 How to read this chapter

The NVIC is identical for all LPC11Uxx parts. See Section 23.5.2 for details.

6.2 Introduction

The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M0. The tight coupling to the CPU allows for low interrupt latency and efficient processing of late arriving interrupts.

6.3 Features

- Nested Vectored Interrupt Controller that is an integral part of the ARM Cortex-M0
- Tightly coupled interrupt controller provides low interrupt latency
- Controls system exceptions and peripheral interrupts
- The NVIC supports 32 vectored interrupts
- 4 programmable interrupt priority levels with hardware priority level masking
- Software interrupt generation
- Support for NMI

6.4 Interrupt sources

Table 58 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. There is no significance or priority about what line is connected where, except for certain standards from ARM.

See Section 23.5.2 for the NVIC register bit descriptions.

Table 58. Connection of interrupt sources to the Vectored Interrupt Controller

Exception Number	Name	Description	Flags
0	PIN_INT0	GPIO pin interrupt 0	-
1	PIN_INT1	GPIO pin interrupt 1	-
2	PIN_INT2	GPIO pin interrupt 2	-
3	PIN_INT3	GPIO pin interrupt 3	-
4	PIN_INT4	GPIO pin interrupt 4	-
5	PIN_INT5	GPIO pin interrupt 5	-
6	PIN_INT6	GPIO pin interrupt 6	-
7	PIN_INT7	GPIO pin interrupt 7	-
8	GINT0	GPIO GROUP0 interrupt	-

Table 58. Connection of interrupt sources to the Vectored Interrupt Controller

Exception Number	Name	Description	Flags
9	GINT1	GPIO GROUP1 interrupt	-
13 to 10		-	Reserved
14	SSP1	SSP1 interrupt	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
15	I2C	I2C interrupt	SI (state change)
16	CT16B0	CT16B0 interrupt	Match 0 - 2 Capture 0
17	CT16B1	CT16B1 interrupt	Match 0 - 1 Capture 0
18	CT32B0	CT32B0 interrupt	Match 0 - 3 Capture 0
19	CT32B1	CT32B1 interrupt	Match 0 - 3 Capture 0
20	SSP0	SSP0 interrupt	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
21	USART	USART interrupt	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) Modem control interrupt
22	USB_IRQ	USB_IRQ interrupt	USB IRQ interrupt
23	USB_FIQ	USB_FIQ interrupt	USB FIQ interrupt
24	ADC	ADC interrupt	A/D Converter end of conversion
25	WWDT	WWDT interrupt	Windowed Watchdog interrupt (WDINT)
26	BOD	BOD interrupt	Brown-out detect
27	FLASH	Flash/EEPROM interface interrupt	-
28	-	-	Reserved
29	-	-	Reserved
30	USB_WAKEUP	USB_WAKEUP interrupt	USB wake-up interrupt
31	-	-	Reserved

7.1 How to read this chapter

The IOCON register map depends on the package type (see Table 59). Registers for pins that are not pinned out are reserved.

Table 59. IOCON registers available

Package	Port 0	Port 1
HVQFN33	PIO0_0 to PIO0_23	PIO1_15; PIO1_19
LQFPN48	PIO0_0 to PIO0_23	PIO1_13 to PIO1_16; PIO1_19 to PIO1_29; PIO1_31
TFBGA48	PIO0_0 to PIO0_23	PIO1_5; PIO1_13 to PIO1_16; PIO1_19 to PIO1_29
LQFP64	PIO0_0 to PIO0_23	PIO1_0 to PIO1_29

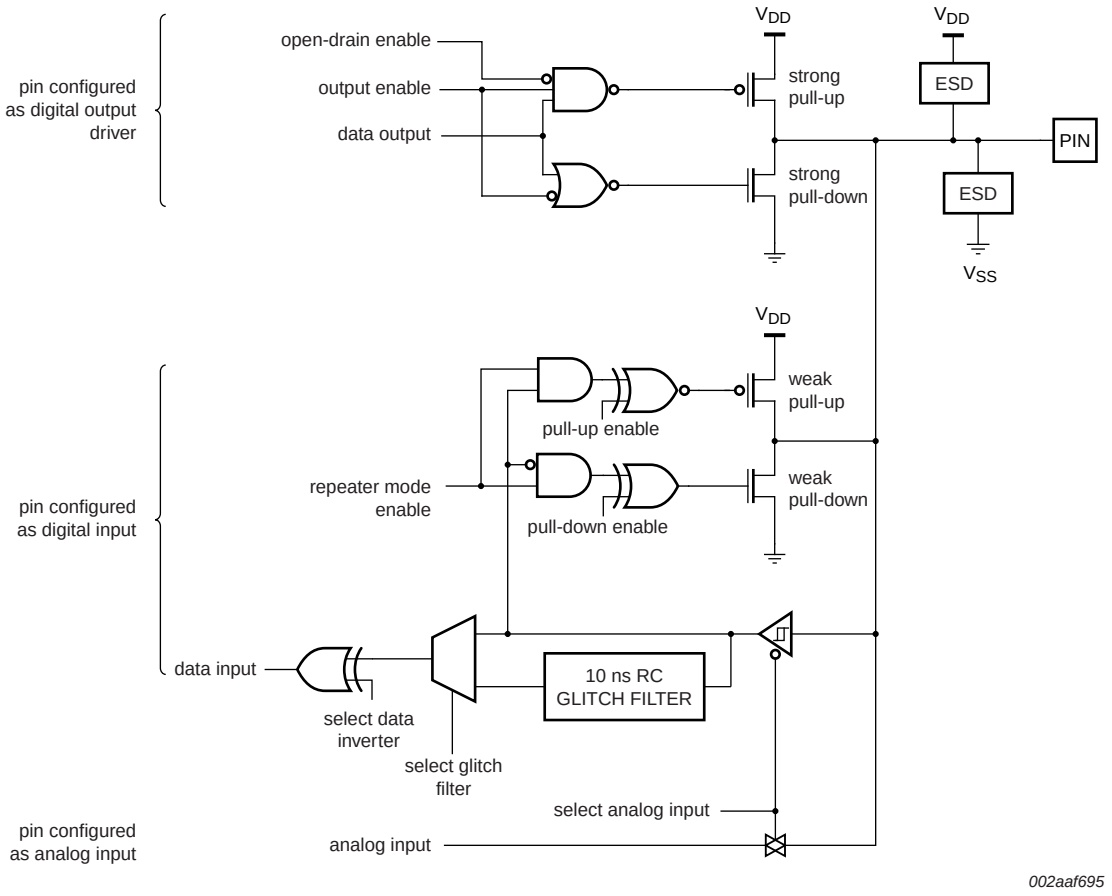
7.2 Introduction

The I/O configuration registers control the electrical characteristics of the pads. The following features are programmable:

- Pin function
- Internal pull-up/pull-down resistor or bus keeper function (repeater mode)
- Open-drain mode for standard I/O pins
- Hysteresis
- Input inverter
- Glitch filter on selected pins
- Analog input or digital mode for pads hosting the ADC inputs
- I^2C mode for pads hosting the I^2C-bus function

7.3 General description

The IOCON registers control the function (GPIO or peripheral function) and the electrical characteristics of the port pins (see Figure 11).



The 10 ns glitch filter is available on selected pins only.

Fig 11. Standard I/O pin configuration

7.3.1 Pin function

The FUNC bits in the IOCON registers can be set to GPIO (FUNC = 000) or to a peripheral function. If the pins are GPIO pins, the DIR registers determine whether the pin is configured as an input or output (see Section 9.5.3.3). For any peripheral function, the pin direction is controlled automatically depending on the pin's functionality. The DIR registers have no effect for peripheral functions.

7.3.2 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 at a logic HIGH and enables the pull-down resistor if the pin is at a logic LOW. This causes the pin to retain its last known state if it is configured as an input and is not driven externally. The 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.

7.3.3 Hysteresis

The input buffer for digital functions can be configured with hysteresis or as plain buffer through the IOCON registers.

If the external pad supply voltage V_{DD} is between 2.5 V and 3.6 V, the hysteresis buffer can be enabled or disabled. If V_{DD} is below 2.5 V, the hysteresis buffer must be **disabled** to use the pin in input mode.

7.3.4 Input inverter

If the input inverter is enabled, a HIGH pin level is inverted to 0 and a LOW pin level is inverted to 1.

7.3.5 Input glitch filter

Selected pins (pins PIO0_22, PIO0_23, and PIO0_11 to PIO0_16) provide the option of turning on or off a 10 ns input glitch filter. The glitch filter is turned off by default. The $\overline{\text{RESET}}$ pin has a 20 ns glitch filter (not configurable).

7.3.6 Open-drain mode

A pseudo open-drain mode can be supported for all digital pins. Note that except for the I^2C-bus pins, this is not a true open-drain mode.

7.3.7 Analog mode

In analog mode, the digital receiver is disconnected to obtain an accurate input voltage for analog-to-digital conversions. This mode can be selected in those IOCON registers that control pins with an analog function. If analog mode is selected, hysteresis, pin mode, inverter, glitch filter, and open-drain settings have no effect.

For pins without analog functions, the analog mode setting has no effect.

7.3.8 I^2C mode

If the I^2C function is selected by the FUNC bits of registers PIO0_4 (Table 65) and PIO0_5 (Table 66), then the I^2C-bus pins can be configured for different I^2C-modes:

- Standard mode/Fast-mode I^2C with 50 ns input glitch filter. An open-drain output according to the I^2C-bus specification can be configured separately.
- Fast-mode Plus I^2C with 50 ns input glitch filter. In this mode, the pins function as high-current sinks. An open-drain output according to the I^2C-bus specification can be configured separately.
- Standard functionality without input filter.

Remark: Either Standard mode/Fast-mode I^2C or Standard I/O functionality should be selected if the pin is used as GPIO pin.

7.3.9 $\overline{\text{RESET}}$ pin (pin RESET_PIO0_0)

See Figure 12 for the reset pad configuration. $\overline{\text{RESET}}$ functionality is not available in Deep power-down mode. Use the WAKEUP pin to reset the chip and wake up from Deep power-down mode. An external pull-up resistor is required on this pin for the Deep power-down mode. The reset pin includes a fixed 20 ns glitch filter.

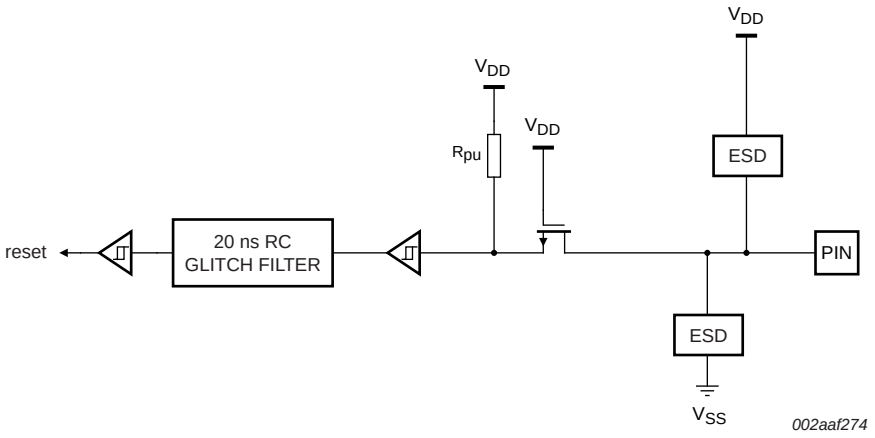


Fig 12. Reset pad configuration

7.3.10 WAKEUP pin (pin PIO0_16)

The WAKEUP pin is combined with pin PIO0_16 and includes a 20 ns fixed glitch filter. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. A LOW-going pulse as short as 50 ns wakes up the part.

7.4 Register description

The I/O configuration registers control the PIO port pins, the inputs and outputs of all peripherals and functional blocks, the I^2C-bus pins, and the ADC input pins.

Each port pin PIOn_m has one IOCON register assigned to control the pin's function and electrical characteristics.

Table 60. Register overview: I/O configuration (base address 0x4004 4000)

Name	Access	Address offset	Description	Reset value	Reference
RESET_PIO0_0	R/W	0x000	I/O configuration for pin RESET/PIO0_0	0x0000 0090	Table 61
PIO0_1	R/W	0x004	I/O configuration for pin PIO0_1/CLKOUT/CT32B0_MAT2/USB_FTOGGLE	0x0000 0090	Table 62
PIO0_2	R/W	0x008	I/O configuration for pin PIO0_2/SSEL0/CT16B0_CAP0	0x0000 0090	Table 63
PIO0_3	R/W	0x00C	I/O configuration for pin PIO0_3/USB_VBUS	0x0000 0090	Table 64
PIO0_4	R/W	0x010	I/O configuration for pin PIO0_4/SCL	0x0000 0080	Table 65
PIO0_5	R/W	0x014	I/O configuration for pin PIO0_5/SDA	0x0000 0080	Table 66
PIO0_6	R/W	0x018	I/O configuration for pin PIO0_6/USB_CONNECT/SCK0	0x0000 0090	Table 67
PIO0_7	R/W	0x01C	I/O configuration for pin PIO0_7/CTS	0x0000 0090	Table 68
PIO0_8	R/W	0x020	I/O configuration for pin PIO0_8/MISO0/CT16B0_MAT0	0x0000 0090	Table 69
PIO0_9	R/W	0x024	I/O configuration for pin PIO0_9/MOSI0/CT16B0_MAT1	0x0000 0090	Table 70
SWCLK_PIO0_10	R/W	0x028	I/O configuration for pin SWCLK/PIO0_10/SCK0/CT16B0_MAT2	0x0000 0090	Table 71
TDI_PIO0_11	R/W	0x02C	I/O configuration for pin TDI/PIO0_11/AD0/CT32B0_MAT3	0x0000 0090	Table 72
TMS_PIO0_12	R/W	0x030	I/O configuration for pin TMS/PIO0_12/AD1/CT32B1_CAP0	0x0000 0090	Table 73
TDO_PIO0_13	R/W	0x034	I/O configuration for pin TDO/PIO0_13/AD2/CT32B1_MAT0	0x0000 0090	Table 74
TRST_PIO0_14	R/W	0x038	I/O configuration for pin TRST/PIO0_14/AD3/CT32B1_MAT1	0x0000 0090	Table 75
SWDIO_PIO0_15	R/W	0x03C	I/O configuration for pin SWDIO/PIO0_15/AD4/CT32B1_MAT2	0x0000 0090	Table 76
PIO0_16	R/W	0x040	I/O configuration for pin PIO0_16/AD5/CT32B1_MAT3/WAKEUP	0x0000 0090	Table 77
PIO0_17	R/W	0x044	I/O configuration for pin PIO0_17/RTS/CT32B0_CAP0/SCLK	0x0000 0090	Table 78
PIO0_18	R/W	0x048	I/O configuration for pin PIO0_18/RXD/CT32B0_MAT0	0x0000 0090	Table 79
PIO0_19	R/W	0x04C	I/O configuration for pin PIO0_19/TXD/CT32B0_MAT1	0x0000 0090	Table 80

Table 60. Register overview: I/O configuration (base address 0x4004 4000)

Name	Access	Address offset	Description	Reset value	Reference
PIO0_20	R/W	0x050	I/O configuration for pin PIO0_20/CT16B1_CAP0	0x0000 0090	Table 81
PIO0_21	R/W	0x054	I/O configuration for pin PIO0_21/CT16B1_MAT0/MOSI1	0x0000 0090	Table 82
PIO0_22	R/W	0x058	I/O configuration for pin PIO0_22/AD6/CT16B1_MAT1/MISO1	0x0000 0090	Table 83
PIO0_23	R/W	0x05C	I/O configuration for pin PIO0_23/AD7	0x0000 0090	Table 84
PIO1_0	R/W	0x060	I/O configuration for pin PIO1_0/CT32B1_MAT0	0x0000 0090	Table 85
PIO1_1	R/W	0x064	I/O configuration for pin PIO1_1/CT32B1_MAT1	0x0000 0090	Table 86
PIO1_2	R/W	0x068	I/O configuration for pin PIO1_2/CT32B1_MAT2	0x0000 0090	Table 87
PIO1_3	R/W	0x06C	I/O configuration for pin PIO1_3/CT32B1_MAT3	0x0000 0090	Table 88
PIO1_4	R/W	0x070	I/O configuration for pin PIO1_4/CT32B1_CAP0	0x0000 0090	Table 89
PIO1_5	R/W	0x074	I/O configuration for pin PIO1_5/CT32B1_CAP1	0x0000 0090	Table 90
PIO1_6	R/W	0x078	I/O configuration for pin PIO1_6	0x0000 0090	Table 91
PIO1_7	R/W	0x07C	I/O configuration for pin PIO1_7	0x0000 0090	Table 92
PIO1_8	R/W	0x080	I/O configuration for pin PIO1_8	0x0000 0090	Table 93
PIO1_9	R/W	0x084	I/O configuration for pin PIO1_9	0x0000 0090	Table 94
PIO1_10	R/W	0x088	I/O configuration for pin PIO1_10	0x0000 0090	Table 95
PIO1_11	R/W	0x08C	I/O configuration for pin PIO1_11	0x0000 0090	Table 96
PIO1_12	R/W	0x090	I/O configuration for pin PIO1_12	0x0000 0090	Table 97
PIO1_13	R/W	0x094	I/O configuration for pin PIO1_13/$\overline{\text{DTR}}$/CT16B0_MAT0/TXD	0x0000 0090	Table 98
PIO1_14	R/W	0x098	I/O configuration for pin PIO1_14/$\overline{\text{DSR}}$/CT16B0_MAT1/RXD	0x0000 0090	Table 99
PIO1_15	R/W	0x09C	I/O configuration for pin PIO1_15/$\overline{\text{DCD}}$/CT16B0_MAT2/SCK1	0x0000 0090	Table 100
PIO1_16	R/W	0x0A0	I/O configuration for pin PIO1_16/$\overline{\text{RI}}$/CT16B0_CAP0	0x0000 0090	Table 101
PIO1_17	R/W	0x0A4	I/O configuration for PIO1_17/CT16B0_CAP1/RXD	0x0000 0090	Table 102
PIO1_18	R/W	0x0A8	I/O configuration for PIO1_18/CT16B1_CAP1/TXD	0x0000 0090	Table 103
PIO1_19	R/W	0x0AC	I/O configuration for pin PIO1_19/$\overline{\text{DTR}}$/SSEL1	0x0000 0090	Table 104
PIO1_20	R/W	0x0B0	I/O configuration for pin PIO1_20/$\overline{\text{DSR}}$/SCK1	0x0000 0090	Table 105
PIO1_21	R/W	0x0B4	I/O configuration for pin PIO1_21/$\overline{\text{DCD}}$/MISO1	0x0000 0090	Table 106

Table 60. Register overview: I/O configuration (base address 0x4004 4000)

Name	Access	Address offset	Description	Reset value	Reference
PIO1_22	R/W	0x0B8	I/O configuration for pin PIO1_22/RI/MOSI1	0x0000 0090	Table 107
PIO1_23	R/W	0x0BC	I/O configuration for pin PIO1_23/CT16B1_MAT1/SSEL1	0x0000 0090	Table 108
PIO1_24	R/W	0x0C0	I/O configuration for pin PIO1_24/ CT32B0_MAT0	0x0000 0090	Table 109
PIO1_25	R/W	0x0C4	I/O configuration for pin PIO1_25/CT32B0_MAT1	0x0000 0090	Table 110
PIO1_26	R/W	0x0C8	I/O configuration for pin PIO1_26/CT32B0_MAT2/RXD	0x0000 0090	Table 111
PIO1_27	R/W	0x0CC	I/O configuration for pin PIO1_27/CT32B0_MAT3/TXD	0x0000 0090	Table 112
PIO1_28	R/W	0x0D0	I/O configuration for pin PIO1_28/CT32B0_CAP0/SCLK	0x0000 0090	Table 113
PIO1_29	R/W	0x0D4	I/O configuration for pin PIO1_29/SCK0/ CT32B0_CAP1	0x0000 0090	Table 114
-	R/W	0x0D8	Reserved	-	-
PIO1_31	R/W	0x0DC	I/O configuration for pin PIO1_31	0x0000 0090	Table 115

7.4.1 I/O configuration registers

7.4.1.1 RESET_PIO0_0 register

Table 61. RESET_PIO0_0 register (RESET_PIO0_0, address 0x4004 4000) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	RESET.	
		0x1	PIO0_0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001

Table 61. RESET_PIO0_0 register (RESET_PIO0_0, address 0x4004 4000) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.2 PIO0_1 register

Table 62. PIO0_1 register (PIO0_1, address 0x4004 4004) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO0_1.	
		0x1	CLKOUT.	
		0x2	CT32B0_MAT2.	
		0x3	USB_FTOGGLE.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.3 PIO0_2 register

Table 63. PIO0_2 register (PIO0_2, address 0x4004 4008) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_2.	
		0x1	SSEL0.	
		0x2	CT16B0_CAP0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.4 PIO0_3 register

Table 64. PIO0_3 register (PIO0_3, address 0x4004 400C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_3.	
		0x1	USB_VBUS.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 64. PIO0_3 register (PIO0_3, address 0x4004 400C) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.5 PIO0_4 register

Table 65. PIO0_4 register (PIO0_4, address 0x4004 4010) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_4 (open-drain pin).	
		0x1	I2C SCL (open-drain pin).	
7:3	-	-	Reserved.	10000
9:8	I2CMODE		Selects I2C mode (see Section 7.3.8). Select Standard mode (I2CMODE = 00, default) or Standard I/O functionality (I2CMODE = 01) if the pin function is GPIO (FUNC = 000).	00
		0x0	Standard mode/ Fast-mode I2C.	
		0x1	Standard I/O functionality	
		0x2	Fast-mode Plus I2C	
		0x3	Reserved.	
31:10	-	-	Reserved.	-

7.4.1.6 PIO0_5 register

Table 66. PIO0_5 register (PIO0_5, address 0x4004 4014) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_5 (open-drain pin).	
		0x1	I2C SDA (open-drain pin).	
7:3	-	-	Reserved.	10000

Table 66. PIO0_5 register (PIO0_5, address 0x4004 4014) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
9:8	I2CMODE		Selects I2C mode (see Section 7.3.8).	00
			Select Standard mode (I2CMODE = 00, default) or Standard I/O functionality (I2CMODE = 01) if the pin function is GPIO (FUNC = 000).	
		0x0	Standard mode/ Fast-mode I2C.	
		0x1	Standard I/O functionality	
		0x2	Fast-mode Plus I2C	
		0x3	Reserved.	
31:10	-	-	Reserved.	-

7.4.1.7 PIO0_6 register

Table 67. PIO0_6 register (PIO0_6, address 0x4004 4018) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_6.	
		0x1	$\overline{\text{USB\_CONNECT}}$.	
		0x2	SCK0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.8 PIO0_7 register

Table 68. PIO0_7 register (PIO0_7, address 0x4004 401C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_7.	
		0x1	$\overline{\text{CTS}}$.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.9 PIO0_8 register

Table 69. PIO0_8 register (PIO0_8, address 0x4004 4020) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_8.	
		0x1	MISO0.	
		0x2	CT16B0_MAT0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 69. PIO0_8 register (PIO0_8, address 0x4004 4020) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.10 PIO0_9 register

Table 70. PIO0_9 register (PIO0_9, address 0x4004 4024) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_9.	
		0x1	MOSI0.	
		0x2	CT16B0_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001

Table 70. PIO0_9 register (PIO0_9, address 0x4004 4024) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.11 SWCLK_PIO0_10 register

Table 71. SWCLK_PIO0_10 register (SWCLK_PIO0_10, address 0x4004 4028) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	SWCLK.	
		0x1	PIO0_10.	
		0x2	SCK0.	
		0x3	CT16B0_MAT2.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.12 TDI_PIO0_11 register

Table 72. TDI_PIO0_11 register (TDI_PIO0_11, address 0x4004 402C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	TDI.	
		0x1	PIO0_11.	
		0x2	AD0.	
		0x3	CT32B0_MAT3.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.13 TMS_PIO0_12 register

Table 73. TMS_PIO0_12 register (TMS_PIO0_12, address 0x4004 4030) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	TMS.	
		0x1	PIO0_12.	
		0x2	AD1.	
		0x3	CT32B1_CAP0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.14 PIO0_13 register

Table 74. TDO_PIO0_13 register (TDO_PIO0_13, address 0x4004 4034) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	TDO.	
		0x1	PIO0_13.	
		0x2	AD2.	
		0x3	CT32B1_MAT0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.15 TRST_PIO0_14 register

Table 75. TRST_PIO0_14 register (TRST_PIO0_14, address 0x4004 4038) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	TRST.	
		0x1	PIO0_14.	
		0x2	AD3.	
		0x3	CT32B1_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.16 SWDIO_PIO0_15 register

Table 76. SWDIO_PIO0_15 register (SWDIO_PIO0_15, address 0x4004 403C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	SWDIO.	
		0x1	PIO0_15.	
		0x2	AD4.	
		0x3	CT32B1_MAT2.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.17 PIO0_16 register

Table 77. PIO0_16 register (PIO0_16, address 0x4004 4040) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. This pin functions as WAKEUP pin if the LPC11Uxx is in Deep power-down mode regardless of the value of FUNC. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_16.	
		0x1	AD5.	
		0x2	CT32B1_MAT3.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.18 PIO0_17 register

Table 78. PIO0_17 register (PIO0_17, address 0x4004 4044) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO0_17.	
		0x1	$\overline{\text{RTS}}$.	
		0x2	CT32B0_CAP0.	
		0x3	SCLK (UART synchronous clock).	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.19 PIO0_18 register

Table 79. PIO0_18 register (PIO0_18, address 0x4004 4048) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_18.	
		0x1	RXD.	
		0x2	CT32B0_MAT0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 79. PIO0_18 register (PIO0_18, address 0x4004 4048) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.20 PIO0_19 register

Table 80. PIO0_19 register (PIO0_19, address 0x4004 404C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_19.	
		0x1	TXD.	
		0x2	CT32B0_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001

Table 80. PIO0_19 register (PIO0_19, address 0x4004 404C) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.21 PIO0_20 register

Table 81. PIO0_20 register (PIO0_20, address 0x4004 4050) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_20.	
		0x1	CT16B1_CAP0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.22 PIO0_21 register

Table 82. PIO0_21 register (PIO0_21, address 0x4004 4054) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO0_21.	
		0x1	CT16B1_MAT0.	
		0x2	MOSI1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.23 PIO0_22 register

Table 83. PIO0_22 register (PIO0_22, address 0x4004 4058) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO0_22.	
		0x1	AD6.	
		0x2	CT16B1_MAT1.	
		0x3	MISO1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 83. PIO0_22 register (PIO0_22, address 0x4004 4058) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.24 PIO0_23 register

Table 84. PIO0_23 register (PIO0_23, address 0x4004 405C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO0_23.	
		0x1	AD7.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	

Table 84. PIO0_23 register (PIO0_23, address 0x4004 405C) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
7	ADMODE		Selects Analog/Digital mode.	1
		0	Analog input mode.	
		1	Digital functional mode.	
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter disabled.	
		1	Filter enabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled.	
			Remark: This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.25 PIO1_0 register

Table 85. PIO1_0 register (PIO1_0, address 0x4004 4060) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_0.	
		0x1	CT32B1_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	0x1
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.26 PIO1_1 register

Table 86. PIO1_1 register (PIO1_1, address 0x4004 4064) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_1.	
		0x1	CT32B1_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	0x1
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.27 PIO1_2 register

Table 87. PIO1_2 register (PIO1_2, address 0x4004 4068) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_2.	
		0x1	CT32B1_MAT2.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 87. PIO1_2 register (PIO1_2, address 0x4004 4068) bit description

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	0x1
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.28 PIO1_3 register

Table 88. PIO1_3 (PIO1_3, address 0x4004406C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_3.	
		0x1	CT32B1_MAT3.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	0x1
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.29 PIO1_4 register

Table 89. I/O configuration PIO1_4 (PIO1_4, address 0x4004 4070) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_4.	
		0x1	CT32B1_CAP0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	0x1
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.30 PIO1_5 register

Table 90. PIO1_5 register (PIO1_5, address 0x4004 4074) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO1_5.	
		0x1	CT32B1_CAP1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 90. PIO1_5 register (PIO1_5, address 0x4004 4074) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.31 PIO1_6 register

Table 91. PIO1_6 register (PIO1_6, address 0x4004 4078) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_6.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.32 PIO1_7 register

Table 92. PIO1_7 register (PIO1_7, address 0x4004 407C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	0
		0x0	PIO1_7.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.33 PIO1_8 register

Table 93. PIO1_8 register (PIO1_8, address 0x4004 4080) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	0
		0x0	PIO1_8.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 93. PIO1_8 register (PIO1_8, address 0x4004 4080) bit description

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.34 PIO1_9 register

Table 94. PIO1_9 register (PIO1_9, address 0x4004 4084) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	0
		0x0	PIO1_9.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.35 PIO1_10 register

Table 95. PIO1_10 register (PIO1_10, address 0x4004 4088) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	0
		0x0	PIO1_10.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.36 PIO1_11 register

Table 96. PIO1_11 register (PIO1_11, address 0x4004 408C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	000
		0x0	PIO1_11.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 96. PIO1_11 register (PIO1_11, address 0x4004 408C) bit description

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.37 PIO1_12 register

Table 97. PIO1_12 register (PIO1_12, address 0x4004 4090) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	000
		0x0	PIO1_12.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.38 PIO1_13 register

Table 98. PIO1_13 register (PIO1_13, address 0x4004 4094) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO1_13.	
		0x1	$\overline{\text{DTR}}$.	
		0x2	CT16B0_MAT0.	
		0x3	TXD.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.39 PIO1_14 register

Table 99. PIO1_14 register (PIO1_14, address 0x4004 4098) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO1_14.	
		0x1	$\overline{\text{DSR}}$.	
		0x2	CT16B0_MAT1.	
		0x3	RXD.	

Table 99. PIO1_14 register (PIO1_14, address 0x4004 4098) bit description

Bit	Symbol	Value	Description	Reset value
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.40 PIO1_15 register

Table 100. PIO1_15 register (PIO1_15, address 0x4004 409C) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x4 to 0x7 are reserved.	000
		0x0	PIO1_15.	
		0x1	$\overline{\text{DCD}}$.	
		0x2	CT16B0_MAT2.	
		0x3	SCK1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 100. PIO1_15 register (PIO1_15, address 0x4004 409C) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.41 PIO1_16 register

Table 101. PIO1_16 register (PIO1_16, address 0x4004 40A0) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_16.	
		0x1	$\overline{RI}$.	
		0x2	CT16B0_CAP0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.42 PIO1_17 register

Table 102. PIO1_17 register (PIO1_17, address 0x4004 40A4) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	0
		0x0	PIO1_17.	
		0x1	CT16B0_CAP1	
		0x2	RXD	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.43 PIO1_18 register

Table 103. PIO1_18 register (PIO1_18, address 0x4004 40A8) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	0
		0x0	PIO1_18	
		0x1	CT16B1_CAP1	
		0x2	TXD	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 103. PIO1_18 register (PIO1_18, address 0x4004 40A8) bit description

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	RESERVED		Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. This is not a true open-drain mode. Input cannot be pulled up above VDD.	
31:11	RESERVED		Reserved.	0

7.4.1.44 PIO1_19 register

Table 104. PIO1_19 register (PIO1_19, address 0x4004 40AC) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_19.	
		0x1	$\overline{\text{DTR}}$.	
		0x2	SSEL1.	
4:3	MODE		mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.45 PIO1_20 register

Table 105. PIO1_20 register (PIO1_20, address 0x4004 40B0) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_20.	
		0x1	$\overline{\text{DSR}}$.	
		0x2	SCK1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.46 PIO1_21 register

Table 106. PIO1_21 register (PIO1_21, address 0x4004 40B4) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_21.	
		0x1	$\overline{\text{DCD}}$.	
		0x2	MISO1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 106. PIO1_21 register (PIO1_21, address 0x4004 40B4) bit description

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.47 PIO1_22 register

Table 107. PIO1_22 register (PIO1_22, address 0x4004 40B8) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_22.	
		0x1	$\overline{RI}$.	
		0x2	MOSI1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001

Table 107. PIO1_22 register (PIO1_22, address 0x4004 40B8) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.48 PIO1_23 register

Table 108. PIO1_23 register (PIO1_23, address 0x4004 40BC) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_23.	
		0x1	CT16B1_MAT1.	
		0x2	SSEL1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.49 PIO1_24 register

Table 109. PIO1_24 register (PIO1_24, address 0x4004 40C0) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO1_24.	
		0x1	CT32B0_MAT0.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.50 PIO1_25 register

Table 110. PIO1_25 register (PIO1_25, address 0x4004 40C4) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x2 to 0x7 are reserved.	000
		0x0	PIO1_25.	
		0x1	CT32B0_MAT1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	

Table 110. PIO1_25 register (PIO1_25, address 0x4004 40C4) bit description

Bit	Symbol	Value	Description	Reset value
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.51 PIO1_26 register

Table 111. PIO1_26 register (PIO1_26, address 0x4004 40C8) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_26.	
		0x1	CT32B0_MAT2	
		0x2	RXD.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.52 PIO1_27 register

Table 112. PIO1_27 register (PIO1_27, address 0x4004 40CC) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_27.	
		0x1	CT32B0_MAT3.	
		0x2	TXD.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.53 PIO1_28 register

Table 113. PIO1_28 register (PIO1_28, address 0x4004 40D0) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_28.	
		0x1	CT32B0_CAP0.	
		0x2	SCLK.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 113. PIO1_28 register (PIO1_28, address 0x4004 40D0) bit description

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.54 PIO1_29 register

Table 114. PIO1_29 register (PIO1_29, address 0x4004 40D4) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x3 to 0x7 are reserved.	000
		0x0	PIO1_29.	
		0x1	SCK0.	
		0x2	CT32B0_CAP1.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001

Table 114. PIO1_29 register (PIO1_29, address 0x4004 40D4) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

7.4.1.55 PIO1_31 register

Table 115. PIO1_31 register (PIO1_31, address 0x4004 40DC) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. Values 0x1 to 0x7 are reserved.	000
		0x0	PIO1_31.	
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1, LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	001
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
31:11	-	-	Reserved.	0

8.1 Pin configuration

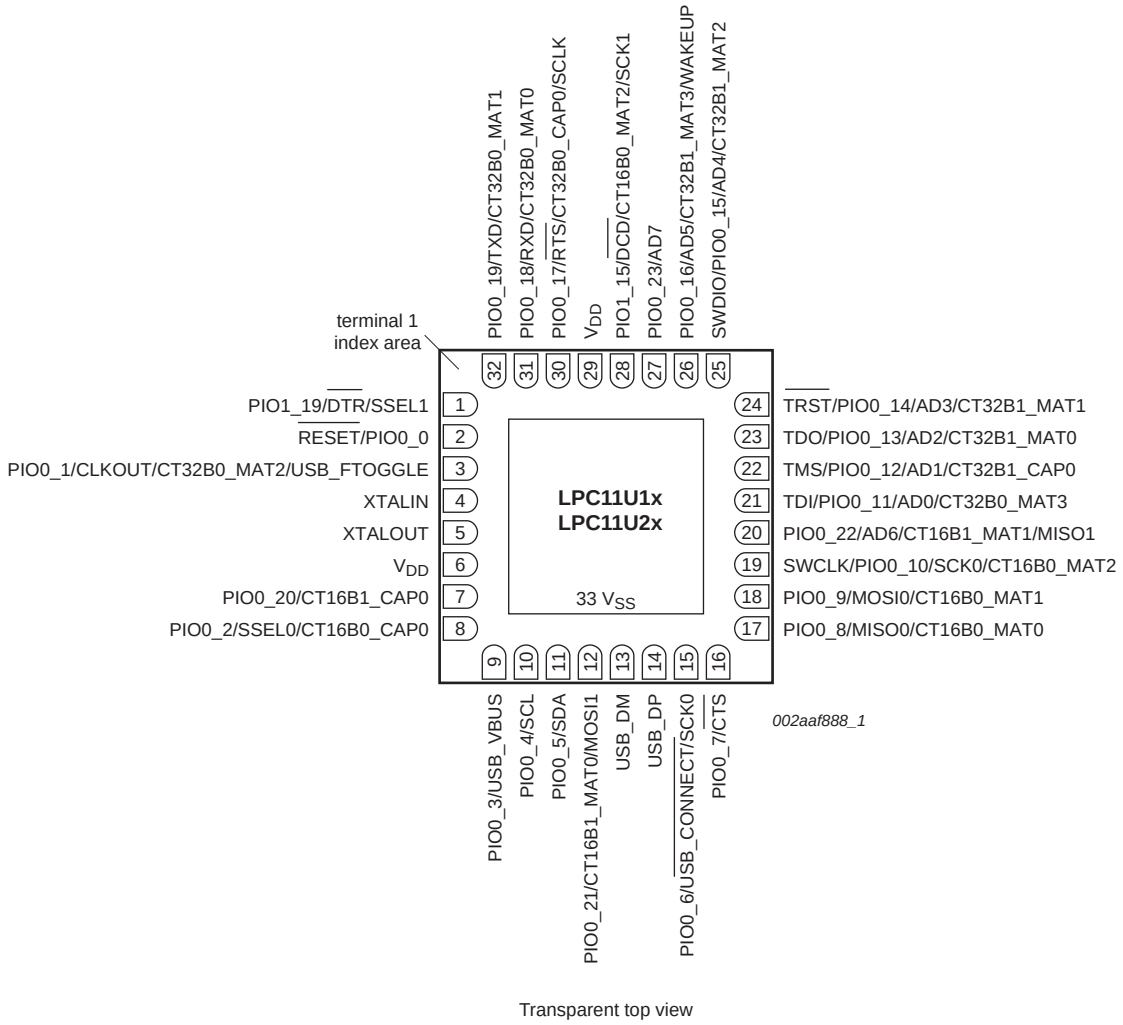


Fig 13. Pin configuration (HVQFN33)

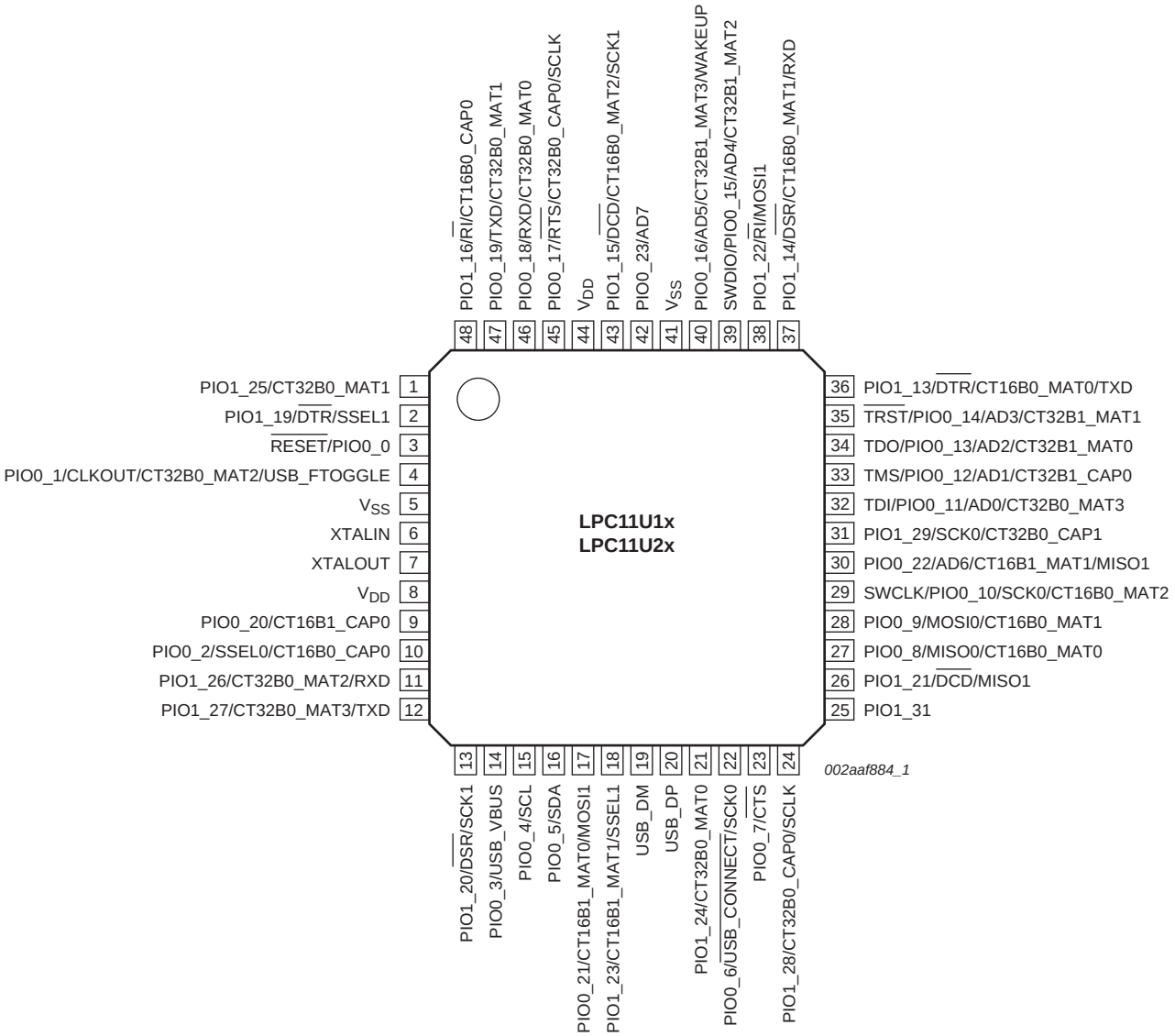


Fig 14. Pin configuration (LQFP48)

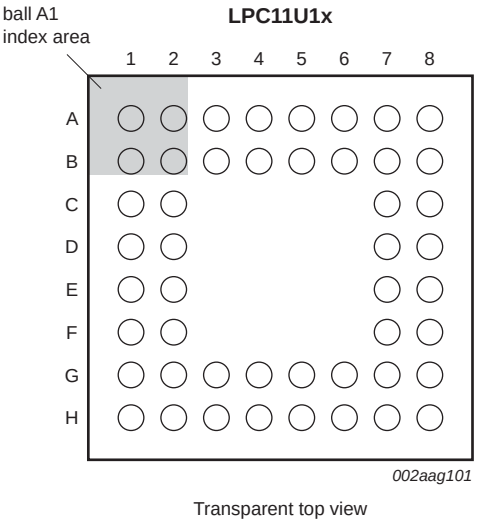
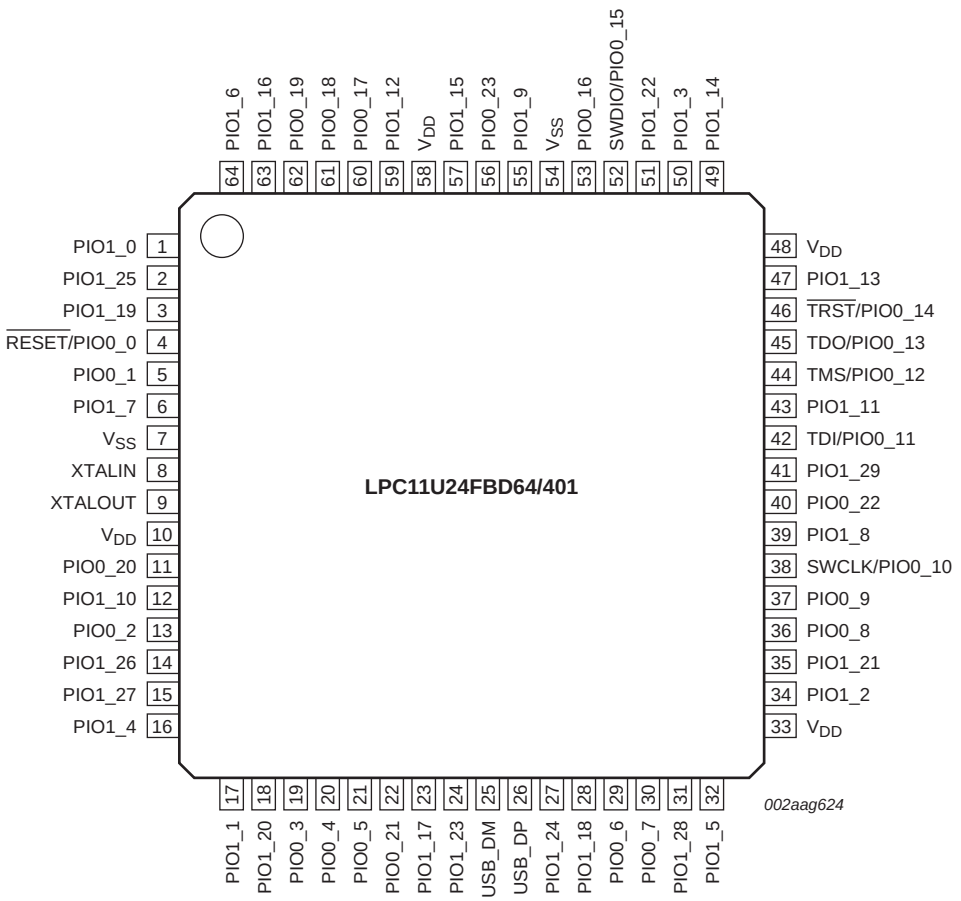


Fig 15. Pin configuration (TFBGA48)



See Table 118 for the full pin name.

Fig 16. Pin configuration (LQFP64)

8.1.1 LPC11U1x pin description

Table 116 shows all pins and their assigned digital or analog functions ordered by GPIO port number. The default function after reset is listed first. All port pins have internal pull-up resistors enabled after reset with the exception of the true open-drain pins PIO0_4 and PIO0_5.

Every port pin has a corresponding IOCON register for programming the digital or analog function, the pull-up/pull-down configuration, the repeater, and the open-drain modes.

The USART, counter/timer, and SSP functions are available on more than one port pin. Table 117 shows how peripheral functions are assigned to port pins.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
$\overline{\text{RESET}}$/PIO0_0	2	3	C1	[2]	I; PU	I	**$\overline{\text{RESET}}$ —** External reset input with 20 ns glitch filter. A LOW-going pulse as short as 50 ns 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 also serves as the debug select input. LOW level selects the JTAG boundary scan. HIGH level selects the ARM SWD debug mode.
					-	I/O	**PIO0_0 —** General purpose digital input/output pin.
PIO0_1/CLKOUT/ CT32B0_MAT2/ USB_FTOGGLE	3	4	C2	[3]	I; PU	I/O	**PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler.
					-	O	**CLKOUT —** Clockout pin.
					-	O	**CT32B0_MAT2 —** Match output 2 for 32-bit timer 0.
					-	O	**USB_FTOGGLE —** USB 1 ms Start-of-Frame signal.
PIO0_2/SSEL0/ CT16B0_CAP0	8	10	F1	[3]	I; PU	I/O	**PIO0_2 —** General purpose digital input/output pin.
					-	I/O	**SSEL0 —** Slave select for SSP0.
					-	I	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
PIO0_3/USB_VBUS	9	14	H2	[3]	I; PU	I/O	**PIO0_3 —** General purpose digital input/output pin.
					-	I	**USB_VBUS —** Monitors the presence of USB bus power.
PIO0_4/SCL	10	15	G3	[4]	I; IA	I/O	**PIO0_4 —** General purpose digital input/output pin (open-drain).
					-	I/O	**SCL —** I^2C-bus clock input/output (open-drain). High-current sink only if I^2C Fast-mode Plus is selected in the I/O configuration register.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
PIO0_5/SDA	11	16	H3	[4]	I; IA	I/O	**PIO0_5 —** General purpose digital input/output pin (open-drain).
					-	I/O	**SDA —** I^2C-bus data input/output (open-drain). High-current sink only if I^2C Fast-mode Plus is selected in the I/O configuration register.
PIO0_6/$\overline{\text{USB\_CONNECT}}$/SCK0	15	22	H6	[3]	I; PU	I/O	**PIO0_6 —** General purpose digital input/output pin.
					-	O	**$\overline{\text{USB\_CONNECT}}$ —** Signal used to switch an external 1.5 kΩ resistor under software control. Used with the SoftConnect USB feature.
					-	I/O	**SCK0 —** Serial clock for SSP0.
PIO0_7/$\overline{\text{CTS}}$	16	23	G7	[5]	I; PU	I/O	**PIO0_7 —** General purpose digital input/output pin (high-current output driver).
					-	I	**$\overline{\text{CTS}}$ —** Clear To Send input for USART.
PIO0_8/MISO0/CT16B0_MAT0	17	27	F8	[3]	I; PU	I/O	**PIO0_8 —** General purpose digital input/output pin.
					-	I/O	**MISO0 —** Master In Slave Out for SSP0.
					-	O	**CT16B0_MAT0 —** Match output 0 for 16-bit timer 0.
PIO0_9/MOSI0/CT16B0_MAT1	18	28	F7	[3]	I; PU	I/O	**PIO0_9 —** General purpose digital input/output pin.
					-	I/O	**MOSI0 —** Master Out Slave In for SSP0.
					-	O	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
SWCLK/PIO0_10/SCK0/CT16B0_MAT2	19	29	E7	[3]	I; PU	I	**SWCLK —** Serial wire clock and test clock TCK for JTAG interface.
					-	I/O	**PIO0_10 —** General purpose digital input/output pin.
					-	O	**SCK0 —** Serial clock for SSP0.
					-	O	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
TDI/PIO0_11/AD0/CT32B0_MAT3	21	32	D8	[6]	I; PU	I	**TDI —** Test Data In for JTAG interface.
					-	I/O	**PIO0_11 —** General purpose digital input/output pin.
					-	I	**AD0 —** A/D converter, input 0.
					-	O	**CT32B0_MAT3 —** Match output 3 for 32-bit timer 0.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
TMS/PIO0_12/AD1/ CT32B1_CAP0	22	33	C7	[6]	I; PU	I	**TMS —** Test Mode Select for JTAG interface.
					-	I/O	**PIO_12 —** General purpose digital input/output pin.
					-	I	**AD1 —** A/D converter, input 1.
					-	I	**CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1.
TDO/PIO0_13/AD2/ CT32B1_MAT0	23	34	C8	[6]	I; PU	O	**TDO —** Test Data Out for JTAG interface.
					-	I/O	**PIO0_13 —** General purpose digital input/output pin.
					-	I	**AD2 —** A/D converter, input 2.
					-	O	**CT32B1_MAT0 —** Match output 0 for 32-bit timer 1.
$\overline{\text{TRST}}$/PIO0_14/AD3/ CT32B1_MAT1	24	35	B7	[6]	I; PU	I	**$\overline{\text{TRST}}$ —** Test Reset for JTAG interface.
					-	I/O	**PIO0_14 —** General purpose digital input/output pin.
					-	I	**AD3 —** A/D converter, input 3.
					-	O	**CT32B1_MAT1 —** Match output 1 for 32-bit timer 1.
SWDIO/PIO0_15/AD4/ CT32B1_MAT2	25	39	B6	[6]	I; PU	I/O	**SWDIO —** Serial wire debug input/output.
					-	I/O	**PIO0_15 —** General purpose digital input/output pin.
					-	I	**AD4 —** A/D converter, input 4.
					-	O	**CT32B1_MAT2 —** Match output 2 for 32-bit timer 1.
PIO0_16/AD5/ CT32B1_MAT3/WAKEUP	26	40	A6	[6]	I; PU	I/O	**PIO0_16 —** General purpose digital input/output pin.
					-	I	**AD5 —** A/D converter, input 5.
					-	O	**CT32B1_MAT3 —** Match output 3 for 32-bit timer 1.
					-	I	**WAKEUP —** Deep power-down mode wake-up pin with 20 ns glitch filter. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. A LOW-going pulse as short as 50 ns wakes up the part.
PIO0_17/$\overline{\text{RTS}}$/ CT32B0_CAP0/SCLK	30	45	A3	[3]	I; PU	I/O	**PIO0_17 —** General purpose digital input/output pin.
					-	O	**$\overline{\text{RTS}}$ —** Request To Send output for USART.
					-	I	**CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0.
					-	I/O	**SCLK —** Serial clock input/output for USART in synchronous mode.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
PIO0_18/RXD/ CT32B0_MAT0	31	46	B3	[3]	I; PU	I/O	**PIO0_18 —** General purpose digital input/output pin.
					-	I	**RXD —** Receiver input for USART.Used in UART ISP mode.
					-	O	**CT32B0_MAT0 —** Match output 0 for 32-bit timer 0.
PIO0_19/TXD/ CT32B0_MAT1	32	47	B2	[3]	I; PU	I/O	**PIO0_19 —** General purpose digital input/output pin.
					-	O	**TXD —** Transmitter output for USART. Used in UART ISP mode.
					-	O	**CT32B0_MAT1 —** Match output 1 for 32-bit timer 0.
PIO0_20/CT16B1_CAP0	7	9	F2	[3]	I; PU	I/O	**PIO0_20 —** General purpose digital input/output pin.
					-	I	**CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1.
PIO0_21/CT16B1_MAT0/ MOSI1	12	17	G4	[3]	I; PU	I/O	**PIO0_21 —** General purpose digital input/output pin.
					-	O	**CT16B1_MAT0 —** Match output 0 for 16-bit timer 1.
					-	I/O	**MOSI1 —** Master Out Slave In for SSP1.
PIO0_22/AD6/ CT16B1_MAT1/MISO1	20	30	E8	[6]	I; PU	I/O	**PIO0_22 —** General purpose digital input/output pin.
					-	I	**AD6 —** A/D converter, input 6.
					-	O	**CT16B1_MAT1 —** Match output 1 for 16-bit timer 1.
					-	I/O	**MISO1 —** Master In Slave Out for SSP1.
PIO0_23/AD7	27	42	A5	[6]	I; PU	I/O	**PIO0_23 —** General purpose digital input/output pin.
					-	I	**AD7 —** A/D converter, input 7.
PIO1_5/CT32B1_CAP1	-	-	H8	[3]	I; PU	I/O	**PIO1_5 —** General purpose digital input/output pin.
					-	I	**CT32B1_CAP1 —** Capture input 1 for 32-bit timer 1.
PIO1_13/$\overline{\text{DTR}}$/ CT16B0_MAT0/TXD	-	36	B8	[3]	I; PU	I/O	**PIO1_13 —** General purpose digital input/output pin.
					-	O	**$\overline{\text{DTR}}$ —** Data Terminal Ready output for USART.
					-	O	**CT16B0_MAT0 —** Match output 0 for 16-bit timer 0.
					-	O	**TXD —** Transmitter output for USART.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
PIO1_14/$\overline{\text{DSR}}$/ CT16B0_MAT1/RXD	-	37	A8	[3]	I; PU	I/O	**PIO1_14 —** General purpose digital input/output pin.
					-	I	**$\overline{\text{DSR}}$ —** Data Set Ready input for USART.
					-	O	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
					-	I	**RXD —** Receiver input for USART.
PIO1_15/$\overline{\text{DCD}}$/ CT16B0_MAT2/SCK1	28	43	A4	[3]	I; PU	I/O	**PIO1_15 —** General purpose digital input/output pin.
						I	**$\overline{\text{DCD}}$ —** Data Carrier Detect input for USART.
					-	O	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
					-	I/O	**SCK1 —** Serial clock for SSP1.
PIO1_16/$\overline{\text{RI}}$/ CT16B0_CAP0	-	48	A2	[3]	I; PU	I/O	**PIO1_16 —** General purpose digital input/output pin.
					-	I	**$\overline{\text{RI}}$ —** Ring Indicator input for USART.
					-	I	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
PIO1_19/$\overline{\text{DTR}}$/SSEL1	1	2	B1	[3]	I; PU	I/O	**PIO1_19 —** General purpose digital input/output pin.
					-	O	**$\overline{\text{DTR}}$ —** Data Terminal Ready output for USART.
					-	I/O	**SSEL1 —** Slave select for SSP1.
PIO1_20/$\overline{\text{DSR}}$/SCK1	-	13	H1	[3]	I; PU	I/O	**PIO1_20 —** General purpose digital input/output pin.
					-	I	**$\overline{\text{DSR}}$ —** Data Set Ready input for USART.
					-	I/O	**SCK1 —** Serial clock for SSP1.
PIO1_21/$\overline{\text{DCD}}$/MISO1	-	26	G8	[3]	I; PU	I/O	**PIO1_21 —** General purpose digital input/output pin.
					-	I	**$\overline{\text{DCD}}$ —** Data Carrier Detect input for USART.
					-	I/O	**MISO1 —** Master In Slave Out for SSP1.
PIO1_22/$\overline{\text{RI}}$/MOSI1	-	38	A7	[3]	I; PU	I/O	**PIO1_22 —** General purpose digital input/output pin.
					-	I	**$\overline{\text{RI}}$ —** Ring Indicator input for USART.
					-	I/O	**MOSI1 —** Master Out Slave In for SSP1.
PIO1_23/CT16B1_MAT1/ SSEL1	-	18	H4	[3]	I; PU	I/O	**PIO1_23 —** General purpose digital input/output pin.
					-	O	**CT16B1_MAT1 —** Match output 1 for 16-bit timer 1.
					-	I/O	**SSEL1 —** Slave select for SSP1.

Table 116. LPC11U1x pin description

Symbol	Pin HVQFN33	Pin LQFP48	Ball TFBGA48		Reset state [1]	Type	Description
PIO1_24/CT32B0_MAT0	-	21	G6	[3]	I; PU	I/O	**PIO1_24 —** General purpose digital input/output pin.
					-	O	**CT32B0_MAT0 —** Match output 0 for 32-bit timer 0.
PIO1_25/CT32B0_MAT1	-	1	A1	[3]	I; PU	I/O	**PIO1_25 —** General purpose digital input/output pin.
					-	O	**CT32B0_MAT1 —** Match output 1 for 32-bit timer 0.
PIO1_26/CT32B0_MAT2/ RXD	-	11	G2	[3]	I; PU	I/O	**PIO1_26 —** General purpose digital input/output pin.
					-	O	**CT32B0_MAT2 —** Match output 2 for 32-bit timer 0.
					-	I	**RXD —** Receiver input for USART.
PIO1_27/CT32B0_MAT3/ TXD	-	12	G1	[3]	I; PU	I/O	**PIO1_27 —** General purpose digital input/output pin.
					-	O	**CT32B0_MAT3 —** Match output 3 for 32-bit timer 0.
					-	O	**TXD —** Transmitter output for USART.
PIO1_28/CT32B0_CAP0/ SCLK	-	24	H7	[3]	I; PU	I/O	**PIO1_28 —** General purpose digital input/output pin.
					-	I	**CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0.
					-	I/O	**SCLK —** Serial clock input/output for USART in synchronous mode.
PIO1_29/SCK0/ CT32B0_CAP1	-	31	D7	[3]	I; PU	I/O	**PIO1_29 —** General purpose digital input/output pin.
					-	I/O	**SCK0 —** Serial clock for SSP0.
					-	I	**CT32B0_CAP1 —** Capture input 1 for 32-bit timer 0.
PIO1_31	-	25	-	[3]	I; PU	I/O	**PIO1_31 —** General purpose digital input/output pin.
USB_DM	13	19	G5	[7]	F	-	**USB_DM —** USB bidirectional D− line.
USB_DP	14	20	H5	[7]	F	-	**USB_DP —** USB bidirectional D+ line.
XTALIN	4	6	D1	[8]	-	-	Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V.
XTALOUT	5	7	E1	[8]	-	-	Output from the oscillator amplifier.
V_{DD}	6; 29	8; 44	B4, E2		-	-	Supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage.
V_{SS}	33	5; 41	B5, D2		-	-	Ground.

[1] Pin state at reset for default function: I = Input; O = Output; PU = internal pull-up enabled; IA = inactive, no pull-up/down enabled; F = floating; floating pins, if not used, should be tied to ground or power to minimize power consumption.

[2] $\overline{\text{RESET}}$ functionality is not available in Deep power-down mode. Use the WAKEUP pin to reset the chip and wake up from Deep power-down mode. An external pull-up resistor is required on this pin for the Deep power-down mode.

[3] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors and configurable hysteresis.

[4] I^2C-bus pins compliant with the I^2C-bus specification for I^2C standard mode, I^2C Fast-mode, and I^2C Fast-mode Plus.

[5] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors and configurable hysteresis; includes high-current output driver.

[6] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors, configurable hysteresis, and analog input. When configured as a ADC input, digital section of the pad is disabled and the pin is not 5 V tolerant; includes digital input glitch filter.

[7] Pad provides USB functions. It is designed in accordance with the USB specification, revision 2.0 (Full-speed and Low-speed mode only). This pad is not 5 V tolerant.

[8] When the system oscillator is not used, connect XTALIN and XTALOUT as follows: XTALIN can be left floating or can be grounded (grounding is preferred to reduce susceptibility to noise). XTALOUT should be left floating.

To assign a peripheral function to a port, program the FUNC bits in the port pin's IOCON register with this function. The user must ensure that the assignment of a function to a port pin is unambiguous. Only the debug functions for JTAG and SWD are selected by default in their corresponding IOCON registers. All other functions must be programmed in the IOCON block before they can be used. For details see the *LPC11Uxx user manual*.

Table 117. Multiplexing of peripheral functions

Peripheral	Function	Type	Default	Available on ports				
				HVQFN33/LQFP48/TFBGA48		LQFP48/TFBGA48		TFBGA48
USART	RXD	I	no	PIO0_18	-	PIO1_14	PIO1_26	-
	TXD	O	no	PIO0_19	-	PIO1_13	PIO1_27	-
	$\overline{\text{CTS}}$	I	no	PIO0_7	-	-	-	-
	$\overline{\text{RTS}}$	O	no	PIO0_17	-	-	-	-
	$\overline{\text{DTR}}$	O	no	PIO1_13	PIO1_19	-	-	-
	$\overline{\text{DSR}}$	I	no	-	-	PIO1_14	PIO1_20	-
	$\overline{\text{DCD}}$	I	no	PIO1_15		PIO1_21	-	-
	$\overline{\text{RI}}$	I	no			PIO1_16	PIO1_22	-
	SCLK	I/O	no	PIO0_17		PIO1_28	-	-
SSP0	SCK0	I/O	no	PIO0_6	PIO0_10	PIO1_29		-
	SSEL0	I/O	no	PIO0_2	-	-	-	-
	MISO0	I/O	no	PIO0_8	-	-	-	-
	MOSI0	I/O	no	PIO0_9	-	-	-	-
SSP1	SCK1	I/O	no	PIO1_15		PIO1_20	-	-
	SSEL1	I/O	no	PIO1_19		PIO1_23	-	-
	MISO1	I/O	no	PIO0_22		PIO1_21	-	-
	MOSI1	I/O	no	PIO0_21		PIO1_22	-	-
CT16B0	CT16B0_CAP0	I	no	PIO0_2		PIO1_16	-	-
	CT16B0_MAT0	O	no	PIO0_8		PIO1_13	-	-
	CT16B0_MAT1	O	no	PIO0_9		PIO1_14	-	-
	CT16B0_MAT2	O	no	PIO0_10	PIO1_15	-	-	-

Table 117. Multiplexing of peripheral functions

Peripheral	Function	Type	Default	Available on ports				
				HVQFN33/LQFP48/TFBGA48		LQFP48/TFBGA48		TFBGA48
CT16B1	CT16B1_CAP0	I	no	PIO0_20	-	-	-	-
	CT16B1_MAT0	O	no	PIO0_21	-	-	-	-
	CT16B1_MAT1	O	no	PIO0_22	-	PIO1_23	-	-
CT32B0	CT32B0_CAP0	I	no	PIO0_17	-	PIO1_28	-	-
	CT32B0_CAP1	I	no	PIO1_29	-	-	-	-
	CT32B0_MAT0	O	no	PIO0_18	-	PIO1_24	-	-
	CT32B0_MAT1	O	no	PIO0_19	-	PIO1_25	-	-
	CT32B0_MAT2	O	no	PIO0_1	-	PIO1_26	-	-
	CT32B0_MAT3	O	no	PIO0_11	-	PIO1_27	-	-
CT32B1	CT32B1_CAP0	I	no	PIO0_12		-	-	-
	CT32B1_CAP1	I	no		-	-	-	PIO1_5
	CT32B1_MAT0	O	no	PIO0_13	-	-	-	-
	CT32B1_MAT1	O	no	PIO0_14	-	-	-	-
	CT32B1_MAT2	O	no	PIO0_15	-	-	-	-
	CT32B1_MAT3	O	no	PIO0_16	-	-	-	-
ADC	AD0	I	no	PIO0_11	-	-	-	-
	AD1	I	no	PIO0_12	-	-	-	-
	AD2	I	no	PIO0_13	-	-	-	-
	AD3	I	no	PIO0_14	-	-	-	-
	AD4	I	no	PIO0_15	-	-	-	-
	AD5	I	no	PIO0_16	-	-	-	-
	AD6	I	no	PIO0_22	-	-	-	-
	AD7	I	no	PIO0_23	-	-	-	-
USB	USB_VBUS	I	no	PIO0_3	-	-	-	-
	USB_FTOGGLE	O	no	PIO0_1	-	-	-	-
	$\overline{\text{USB\_CONNECT}}$	O	no	PIO0_6	-	-	-	-
CLKOUT	CLKOUT	O	no	PIO0_1	-	-	-	-
JTAG	TDI	I	yes	PIO0_11	-	-	-	-
	TMS	I	yes	PIO0_12	-	-	-	-
	TDO	O	yes	PIO0_13	-	-	-	-
	$\overline{\text{TRST}}$	I	yes	PIO0_14	-	-	-	-
	TCK	I	yes	PIO0_10	-	-	-	-
SWD	SWCLK	I	yes	PIO0_10	-	-	-	-
	SWDIO	I/O	yes	PIO0_15	-	-	-	-

8.1.2 LPC11U2x pin description

Table 118 shows all pins and their assigned digital or analog functions in order of the GPIO port number. The default function after reset is listed first. All port pins have internal pull-up resistors enabled after reset except for the true open-drain pins PIO0_4 and PIO0_5.

Every port pin has a corresponding IOCON register for programming the digital or analog function, the pull-up/pull-down configuration, the repeater, and the open-drain modes.

The USART, counter/timer, and SSP functions are available on more than one port pin.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
$\overline{\text{RESET}}$/PIO0_0	2	C1	3	4	[2]	I; PU	I	**$\overline{\text{RESET}}$ —** External reset input with 20 ns glitch filter. A LOW-going pulse as short as 50 ns 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 also serves as the debug select input. LOW level selects the JTAG boundary scan. HIGH level selects the ARM SWD debug mode.
						-	I/O	**PIO0_0 —** General purpose digital input/output pin.
PIO0_1/CLKOUT/ CT32B0_MAT2/ USB_FTOGGLE	3	C2	4	5	[3]	I; PU	I/O	**PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler or the USB device enumeration (see pin PIO0_3).
						-	O	**CLKOUT —** Clockout pin.
						-	O	**CT32B0_MAT2 —** Match output 2 for 32-bit timer 0.
						-	O	**USB_FTOGGLE —** USB 1 ms Start-of-Frame signal.
PIO0_2/SSEL0/ CT16B0_CAP0	8	F1	10	13	[3]	I; PU	I/O	**PIO0_2 —** General purpose digital input/output pin.
						-	I/O	**SSEL0 —** Slave select for SSP0.
						-	I	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
PIO0_3/USB_VBUS	9	H2	14	19	[3]	I; PU	I/O	**PIO0_3 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler. A HIGH level during reset starts the USB device enumeration.
						-	I	**USB_VBUS —** Monitors the presence of USB bus power.
PIO0_4/SCL	10	G3	15	20	[4]	I; IA	I/O	**PIO0_4 —** General purpose digital input/output pin (open-drain).
						-	I/O	**SCL —** I^2C-bus clock input/output (open-drain). High-current sink only if I^2C Fast-mode Plus is selected in the I/O configuration register.
PIO0_5/SDA	11	H3	16	21	[4]	I; IA	I/O	**PIO0_5 —** General purpose digital input/output pin (open-drain).
						-	I/O	**SDA —** I^2C-bus data input/output (open-drain). High-current sink only if I^2C Fast-mode Plus is selected in the I/O configuration register.
PIO0_6/$\overline{\text{USB_CONNECT}}$/ SCK0	15	H6	22	29	[3]	I; PU	I/O	**PIO0_6 —** General purpose digital input/output pin.
						-	O	**$\overline{\text{USB_CONNECT}}$ —** Signal used to switch an external 1.5 kΩ resistor under software control. Used with the SoftConnect USB feature.
						-	I/O	**SCK0 —** Serial clock for SSP0.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
PIO0_7/$\overline{\text{CTS}}$	16	G7	23	30	[5]	I; PU	I/O	**PIO0_7 —** General purpose digital input/output pin (high-current output driver).
						-	I	**$\overline{\text{CTS}}$ —** Clear To Send input for USART.
PIO0_8/MISO0/ CT16B0_MAT0	17	F8	27	36	[3]	I; PU	I/O	**PIO0_8 —** General purpose digital input/output pin.
						-	I/O	**MISO0 —** Master In Slave Out for SSP0.
						-	O	**CT16B0_MAT0 —** Match output 0 for 16-bit timer 0.
PIO0_9/MOSI0/ CT16B0_MAT1	18	F7	28	37	[3]	I; PU	I/O	**PIO0_9 —** General purpose digital input/output pin.
						-	I/O	**MOSI0 —** Master Out Slave In for SSP0.
						-	O	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
SWCLK/PIO0_10/SCK0/ CT16B0_MAT2	19	E7	29	38	[3]	I; PU	I	**SWCLK —** Serial wire clock and test clock TCK for JTAG interface.
						-	I/O	**PIO0_10 —** General purpose digital input/output pin.
						-	O	**SCK0 —** Serial clock for SSP0.
						-	O	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
TDI/PIO0_11/AD0/ CT32B0_MAT3	21	D8	32	42	[6]	I; PU	I	**TDI —** Test Data In for JTAG interface.
						-	I/O	**PIO0_11 —** General purpose digital input/output pin.
						-	I	**AD0 —** A/D converter, input 0.
						-	O	**CT32B0_MAT3 —** Match output 3 for 32-bit timer 0.
TMS/PIO0_12/AD1/ CT32B1_CAP0	22	C7	33	44	[6]	I; PU	I	**TMS —** Test Mode Select for JTAG interface.
						-	I/O	**PIO_12 —** General purpose digital input/output pin.
						-	I	**AD1 —** A/D converter, input 1.
						-	I	**CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1.
TDO/PIO0_13/AD2/ CT32B1_MAT0	23	C8	34	45	[6]	I; PU	O	**TDO —** Test Data Out for JTAG interface.
						-	I/O	**PIO0_13 —** General purpose digital input/output pin.
						-	I	**AD2 —** A/D converter, input 2.
						-	O	**CT32B1_MAT0 —** Match output 0 for 32-bit timer 1.
$\overline{\text{TRST}}$/PIO0_14/AD3/ CT32B1_MAT1	24	B7	35	46	[6]	I; PU	I	**$\overline{\text{TRST}}$ —** Test Reset for JTAG interface.
						-	I/O	**PIO0_14 —** General purpose digital input/output pin.
						-	I	**AD3 —** A/D converter, input 3.
						-	O	**CT32B1_MAT1 —** Match output 1 for 32-bit timer 1.
SWDIO/PIO0_15/AD4/ CT32B1_MAT2	25	B6	39	52	[6]	I; PU	I/O	**SWDIO —** Serial wire debug input/output.
						-	I/O	**PIO0_15 —** General purpose digital input/output pin.
						-	I	**AD4 —** A/D converter, input 4.
						-	O	**CT32B1_MAT2 —** Match output 2 for 32-bit timer 1.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
PIO0_16/AD5/ CT32B1_MAT3/WAKEUP	26	A6	40	53	[6]	I; PU	I/O	**PIO0_16 —** General purpose digital input/output pin.
						-	I	**AD5 —** A/D converter, input 5.
						-	O	**CT32B1_MAT3 —** Match output 3 for 32-bit timer 1.
						-	I	**WAKEUP —** Deep power-down mode wake-up pin with 20 ns glitch filter. Pull this pin HIGH externally to enter Deep power-down mode. Pull this pin LOW to exit Deep power-down mode. A LOW-going pulse as short as 50 ns wakes up the part.
PIO0_17/$\overline{\text{RTS}}$/ CT32B0_CAP0/SCLK	30	A3	45	60	[3]	I; PU	I/O	**PIO0_17 —** General purpose digital input/output pin.
						-	O	**$\overline{\text{RTS}}$ —** Request To Send output for USART.
						-	I	**CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0.
						-	I/O	**SCLK —** Serial clock input/output for USART in synchronous mode.
PIO0_18/RXD/ CT32B0_MAT0	31	B3	46	61	[3]	I; PU	I/O	**PIO0_18 —** General purpose digital input/output pin.
						-	I	**RXD —** Receiver input for USART. Used in UART ISP mode.
						-	O	**CT32B0_MAT0 —** Match output 0 for 32-bit timer 0.
PIO0_19/TXD/ CT32B0_MAT1	32	B2	47	62	[3]	I; PU	I/O	**PIO0_19 —** General purpose digital input/output pin.
						-	O	**TXD —** Transmitter output for USART. Used in UART ISP mode.
						-	O	**CT32B0_MAT1 —** Match output 1 for 32-bit timer 0.
PIO0_20/CT16B1_CAP0	7	F2	9	11	[3]	I; PU	I/O	**PIO0_20 —** General purpose digital input/output pin.
						-	I	**CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1.
PIO0_21/CT16B1_MAT0/ MOSI1	12	G4	17	22	[3]	I; PU	I/O	**PIO0_21 —** General purpose digital input/output pin.
						-	O	**CT16B1_MAT0 —** Match output 0 for 16-bit timer 1.
						-	I/O	**MOSI1 —** Master Out Slave In for SSP1.
PIO0_22/AD6/ CT16B1_MAT1/MISO1	20	E8	30	40	[6]	I; PU	I/O	**PIO0_22 —** General purpose digital input/output pin.
						-	I	**AD6 —** A/D converter, input 6.
						-	O	**CT16B1_MAT1 —** Match output 1 for 16-bit timer 1.
						-	I/O	**MISO1 —** Master In Slave Out for SSP1.
PIO0_23/AD7	27	A5	42	56	[6]	I; PU	I/O	**PIO0_23 —** General purpose digital input/output pin.
						-	I	**AD7 —** A/D converter, input 7.
PIO1_0/CT32B1_MAT0	-	-	-	1	[3]	I; PU	I/O	**PIO1_0 —** General purpose digital input/output pin.
						-	O	**CT32B1_MAT0 —** Match output 0 for 32-bit timer 1.
PIO1_1/CT32B1_MAT1	-	-	-	17	[3]	I; PU	I/O	**PIO1_1 —** General purpose digital input/output pin.
						-	O	**CT32B1_MAT1 —** Match output 1 for 32-bit timer 1.
PIO1_2/CT32B1_MAT2	-	-	-	34	[3]	I; PU	I/O	**PIO1_2 —** General purpose digital input/output pin.
						-	O	**CT32B1_MAT2 —** Match output 2 for 32-bit timer 1.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
PIO1_3/CT32B1_MAT3	-	-	-	50	[3]	I; PU	I/O	**PIO1_3 —** General purpose digital input/output pin.
						-	O	**CT32B1_MAT3 —** Match output 3 for 32-bit timer 1.
PIO1_4/CT32B1_CAP0	-	-	-	16	[3]	I; PU	I/O	**PIO1_4 —** General purpose digital input/output pin.
						-	I	**CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1.
PIO1_5/CT32B1_CAP1	-	H8	-	32	[3]	I; PU	I/O	**PIO1_5 —** General purpose digital input/output pin.
						-	I	**CT32B1_CAP1 —** Capture input 1 for 32-bit timer 1.
PIO1_6	-	-	-	64	[3]	I; PU	I/O	**PIO1_6 —** General purpose digital input/output pin.
PIO1_7	-	-	-	6	[3]	I; PU	I/O	**PIO1_7 —** General purpose digital input/output pin.
PIO1_8	-	-	-	39	[3]	I; PU	I/O	**PIO1_8 —** General purpose digital input/output pin.
PIO1_9	-	-	-	55	[3]	I; PU	I/O	**PIO1_9 —** General purpose digital input/output pin.
PIO1_10	-	-	-	12	[3]	I; PU	I/O	**PIO1_10 —** General purpose digital input/output pin.
PIO1_11	-	-	-	43	[3]	I; PU	I/O	**PIO1_11 —** General purpose digital input/output pin.
PIO1_12	-	-	-	59	[3]	I; PU	I/O	**PIO1_12 —** General purpose digital input/output pin.
PIO1_13/$\overline{\text{DTR}}$/ CT16B0_MAT0/TXD	-	B8	36	47	[3]	I; PU	I/O	**PIO1_13 —** General purpose digital input/output pin.
						-	O	**$\overline{\text{DTR}}$ —** Data Terminal Ready output for USART.
						-	O	**CT16B0_MAT0 —** Match output 0 for 16-bit timer 0.
						-	O	**TXD —** Transmitter output for USART.
PIO1_14/$\overline{\text{DSR}}$/ CT16B0_MAT1/RXD	-	A8	37	49	[3]	I; PU	I/O	**PIO1_14 —** General purpose digital input/output pin.
						-	I	**$\overline{\text{DSR}}$ —** Data Set Ready input for USART.
						-	O	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
						-	I	**RXD —** Receiver input for USART.
PIO1_15/$\overline{\text{DCD}}$/ CT16B0_MAT2/SCK1	28	A4	43	57	[3]	I; PU	I/O	**PIO1_15 —** General purpose digital input/output pin.
							I	**$\overline{\text{DCD}}$ —** Data Carrier Detect input for USART.
						-	O	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
						-	I/O	**SCK1 —** Serial clock for SSP1.
PIO1_16/$\overline{\text{RI}}$/ CT16B0_CAP0	-	A2	48	63	[3]	I; PU	I/O	**PIO1_16 —** General purpose digital input/output pin.
						-	I	**$\overline{\text{RI}}$ —** Ring Indicator input for USART.
						-	I	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
PIO1_17/CT16B0_CAP1/ RXD	-	-	-	23	[3]	I; PU	I/O	**PIO1_17 —** General purpose digital input/output pin.
						-	I	**CT16B0_CAP1 —** Capture input 1 for 16-bit timer 0.
						-	I	**RXD —** Receiver input for USART.
PIO1_18/CT16B1_CAP1/ TXD	-	-	-	28	[3]	I; PU	I/O	**PIO1_18 —** General purpose digital input/output pin.
						-	I	**CT16B1_CAP1 —** Capture input 1 for 16-bit timer 1.
						-	O	**TXD —** Transmitter output for USART.
PIO1_19/$\overline{\text{DTR}}$/SSEL1	1	B1	2	3	[3]	I; PU	I/O	**PIO1_19 —** General purpose digital input/output pin.
						-	O	**$\overline{\text{DTR}}$ —** Data Terminal Ready output for USART.
						-	I/O	**SSEL1 —** Slave select for SSP1.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
PIO1_20/$\overline{\text{DSR}}$/SCK1	-	H1	13	18	[3]	I; PU	I/O	**PIO1_20 —** General purpose digital input/output pin.
						-	I	**$\overline{\text{DSR}}$ —** Data Set Ready input for USART.
						-	I/O	**SCK1 —** Serial clock for SSP1.
PIO1_21/$\overline{\text{DCD}}$/MISO1	-	G8	26	35	[3]	I; PU	I/O	**PIO1_21 —** General purpose digital input/output pin.
						-	I	**$\overline{\text{DCD}}$ —** Data Carrier Detect input for USART.
						-	I/O	**MISO1 —** Master In Slave Out for SSP1.
PIO1_22/$\overline{\text{RI}}$/MOSI1	-	A7	38	51	[3]	I; PU	I/O	**PIO1_22 —** General purpose digital input/output pin.
						-	I	**$\overline{\text{RI}}$ —** Ring Indicator input for USART.
						-	I/O	**MOSI1 —** Master Out Slave In for SSP1.
PIO1_23/CT16B1_MAT1/ SSEL1	-	H4	18	24	[3]	I; PU	I/O	**PIO1_23 —** General purpose digital input/output pin.
						-	O	**CT16B1_MAT1 —** Match output 1 for 16-bit timer 1.
						-	I/O	**SSEL1 —** Slave select for SSP1.
PIO1_24/CT32B0_MAT0	-	G6	21	27	[3]	I; PU	I/O	**PIO1_24 —** General purpose digital input/output pin.
						-	O	**CT32B0_MAT0 —** Match output 0 for 32-bit timer 0.
PIO1_25/CT32B0_MAT1	-	A1	1	2	[3]	I; PU	I/O	**PIO1_25 —** General purpose digital input/output pin.
						-	O	**CT32B0_MAT1 —** Match output 1 for 32-bit timer 0.
PIO1_26/CT32B0_MAT2/ RXD	-	G2	11	14	[3]	I; PU	I/O	**PIO1_26 —** General purpose digital input/output pin.
						-	O	**CT32B0_MAT2 —** Match output 2 for 32-bit timer 0.
						-	I	**RXD —** Receiver input for USART.
PIO1_27/CT32B0_MAT3/ TXD	-	G1	12	15	[3]	I; PU	I/O	**PIO1_27 —** General purpose digital input/output pin.
						-	O	**CT32B0_MAT3 —** Match output 3 for 32-bit timer 0.
						-	O	**TXD —** Transmitter output for USART.
PIO1_28/CT32B0_CAP0/ SCLK	-	H7	24	31	[3]	I; PU	I/O	**PIO1_28 —** General purpose digital input/output pin.
						-	I	**CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0.
						-	I/O	**SCLK —** Serial clock input/output for USART in synchronous mode.
PIO1_29/SCK0/ CT32B0_CAP1	-	D7	31	41	[3]	I; PU	I/O	**PIO1_29 —** General purpose digital input/output pin.
						-	I/O	**SCK0 —** Serial clock for SSP0.
						-	I	**CT32B0_CAP1 —** Capture input 1 for 32-bit timer 0.
PIO1_31	-	-	25	-	[3]	I; PU	I/O	**PIO1_31 —** General purpose digital input/output pin.
USB_DM	13	G5	19	25	[7]	F	-	**USB_DM —** USB bidirectional D− line.
USB_DP	14	H5	20	26	[7]	F	-	**USB_DP —** USB bidirectional D+ line.
XTALIN	4	D1	6	8	[8]	-	-	Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V.

Table 118. LPC11U2x pin description

Symbol	Pin HVQFN33	Pin TFBGA48	Pin LQFP48	Pin LQFP64		Reset state [1]	Type	Description
XTALOUT	5	E1	7	9	[8]	-	-	Output from the oscillator amplifier.
V_{DD}	6; 29	B4; E2	8; 44	10; 33; 48; 58		-	-	Supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage.
V_{SS}	33	B5; D2	5; 41	7; 54		-	-	Ground.

[1] Pin state at reset for default function: I = Input; O = Output; PU = internal pull-up enabled; IA = inactive, no pull-up/down enabled; F = floating; If the pins are not used, tie floating pins to ground or power to minimize power consumption.

[2] $\overline{\text{RESET}}$ functionality is not available in Deep power-down mode. Use the WAKEUP pin to reset the chip and wake up from Deep power-down mode. An external pull-up resistor is required on this pin for the Deep power-down mode.

[3] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors and configurable hysteresis.

[4] I^2C-bus pins compliant with the I^2C-bus specification for I^2C standard mode, I^2C Fast-mode, and I^2C Fast-mode Plus.

[5] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors and configurable hysteresis; includes high-current output driver.

[6] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors, configurable hysteresis, and analog input. When configured as a ADC input, digital section of the pad is disabled and the pin is not 5 V tolerant; includes digital input glitch filter.

[7] Pad provides USB functions. It is designed in accordance with the USB specification, revision 2.0 (Full-speed and Low-speed mode only). This pad is not 5 V tolerant.

[8] When the system oscillator is not used, connect XTALIN and XTALOUT as follows: XTALIN can be left floating or can be grounded (grounding is preferred to reduce susceptibility to noise). Leave XTALOUT floating.

9.1 How to read this chapter

All GPIO registers refer to 32 pins on each port. Depending on the package type, not all pins are available, and the corresponding bits in the GPIO registers are reserved (see Table 119).

Table 119. GPIO pins available

Package	GPIO Port 0	GPIO Port 1
HVQFN33	PIO0_0 to PIO0_23	PIO1_15; PIO1_19
LQFPN48	PIO0_0 to PIO0_23	PIO1_13 to PIO1_16; PIO1_19 to PIO1_29; PIO1_31
TFBGA48	PIO0_0 to PIO0_23	PIO1_5; PIO1_13 to PIO1_16; PIO1_19 to PIO1_29
LQFP64	PIO0_0 to PIO0_23	PIO1_0 to PIO1_29; PIO1_31

9.2 Basic configuration

Various register blocks must be enabled to use the GPIO port and pin interrupt features:

- For the pin interrupts, select up to 8 external interrupt pins from all GPIO port pins in the SYSCON block (Table 39) and enable the clock to the pin interrupt register block in the SYSAHBCLKCTRL register (Table 23, bit 19). The pin interrupt wake-up feature is enabled in the STARTERP0 register (Table 42).
- For the group interrupt feature, enable the clock to the GROUP0 and GROUP1 register interfaces in the SYSAHBCLKCTRL register ((Table 23, bit 19). The group interrupt wake-up feature is enabled in the STARTERP1 register (Table 43).
- For the GPIO port registers, enable the clock to the GPIO port register in the SYSAHBCLKCTRL register (Table 23, bit 6).

9.3 Features

9.3.1 GPIO pin interrupt features

- Up to 8 pins can be selected from all GPIO pins as edge- or level-sensitive interrupt requests. Each request creates a separate interrupt in the NVIC.
- Edge-sensitive interrupt pins can interrupt on rising or falling edges or both.
- Level-sensitive interrupt pins can be HIGH- or LOW-active.

9.3.2 GPIO group interrupt features

- The inputs from any number of GPIO pins can be enabled to contribute to a combined group interrupt.
- The polarity of each input enabled for the group interrupt can be configured HIGH or LOW.
- Enabled interrupts can be logically combined through an OR or AND operation.

- Two group interrupts are supported to reflect two distinct interrupt patterns.
- The GPIO group interrupts can wake up the part from sleep, deep-sleep or power-down modes.

9.3.3 GPIO port features

- GPIO pins can be configured as input or output by software.
- All GPIO pins default to inputs with interrupt disabled at reset.
- Pin registers allow pins to be sensed and set individually.

9.4 Introduction

The GPIO pins can be used in several ways to set pins as inputs or outputs and use the inputs as combinations of level and edge sensitive interrupts.

9.4.1 GPIO pin interrupts

From all available GPIO pins, up to eight pins can be selected in the system control block to serve as external interrupt pins (see Table 39). The external interrupt pins are connected to eight individual interrupts in the NVIC and are created based on rising or falling edges or on the input level on the pin.

9.4.2 GPIO group interrupt

For each port/pin connected to one of the two the GPIO Grouped Interrupt blocks (GROUP0 and GROUP1), the GPIO grouped interrupt registers determine which pins are enabled to generate interrupts and what the active polarities of each of those inputs are.

The GPIO grouped interrupt registers also select whether the interrupt output will be level or edge triggered and whether it will be based on the OR or the AND of all of the enabled inputs.

When the designated pattern is detected on the selected input pins, the GPIO grouped interrupt block will generate an interrupt. If the part is in a power-savings mode it will first asynchronously wake the part up prior to asserting the interrupt request. The interrupt request line can be cleared by writing a one to the interrupt status bit in the control register.

9.4.3 GPIO port

The GPIO port registers can be used to configure each GPIO pin as input or output and read the state of each pin if the pin is configured as input or set the state of each pin if the pin is configured as output.

9.5 Register description

The GPIO consists of the following blocks:

- The GPIO pin interrupts block at address 0x4004 C000. Registers in this block enable the up to 8 pin interrupts selected in the syscon block PINTSEL registers (see Table 39) and configure the level and edge sensitivity for each selected pin interrupt. The GPIO interrupt registers are listed in Table 120 and Section 9.5.1
- The GPIO GROUP0 interrupt block at address 0x4005 C000. Registers in this block allow to configure any pin on port 0 and 1 to contribute to a combined interrupt. The GPIO GROUP0 registers are listed in Table 121 and Section 9.5.2.
- The GPIO GROUP1 interrupt block at address 0x4005 8000. Registers in this block allow to configure any pin on port 0 and 1 to contribute to a combined interrupt. The GPIO GROUP1 registers are listed in Table 122 and Section 9.5.2.
- The GPIO port block at address 0x5000 0000. Registers in this block allow to read and write to port pins and configure port pins as inputs or outputs.The GPIO port registers are listed in Table 123 and Section 9.5.3.

Note: In all GPIO registers, bits that are not shown are **reserved**.

Table 120. Register overview: GPIO pin interrupts (base address: 0x4004 C000)

Name	Access	Address offset	Description	Reset value	Reference
ISEL	R/W	0x000	Pin Interrupt Mode register	0	Table 124
IENR	R/W	0x004	Pin Interrupt Enable (Rising) register	0	Table 125
SIENR	WO	0x008	Set Pin Interrupt Enable (Rising) register	NA	Table 126
CIENR	WO	0x00C	Clear Pin Interrupt Enable (Rising) register	NA	Table 127
IENF	R/W	0x010	Pin Interrupt Enable Falling Edge / Active Level register	0	Table 128
SIENF	WO	0x014	Set Pin Interrupt Enable Falling Edge / Active Level register	NA	Table 129
CIENF	WO	0x018	Clear Pin Interrupt Enable Falling Edge / Active Level address	NA	Table 130
RISE	R/W	0x01C	Pin Interrupt Rising Edge register	0	Table 131
FALL	R/W	0x020	Pin Interrupt Falling Edge register	0	Table 132
IST	R/W	0x024	Pin Interrupt Status register	0	Table 133

Table 121. Register overview: GPIO GROUP0 interrupt (base address 0x4005 C000)

Name	Access	Address offset	Description	Reset value	Reference
CTRL	R/W	0x000	GPIO grouped interrupt control register	0	Table 134
PORT_POL0	R/W	0x020	GPIO grouped interrupt port 0 polarity register	0xFFFF FFFF	Table 135
PORT_POL1	R/W	0x024	GPIO grouped interrupt port 1 polarity register	0xFFFF FFFF	Table 136
PORT_ENA0	R/W	0x040	GPIO grouped interrupt port 0 enable register	0	Table 137
PORT_ENA1	R/W	0x044	GPIO grouped interrupt port 1 enable register	0	Table 138

Table 122. Register overview: GPIO GROUP1 interrupt (base address 0x4006 0000)

Name	Access	Address offset	Description	Reset value	Reference
CTRL	R/W	0x000	GPIO grouped interrupt control register	0	Table 134
PORT_POL0	R/W	0x020	GPIO grouped interrupt port 0 polarity register	0xFFFF FFFF	Table 135
PORT_POL1	R/W	0x024	GPIO grouped interrupt port 1 polarity register	0xFFFF FFFF	Table 136
PORT_ENA0	R/W	0x040	GPIO grouped interrupt port 0 enable register	0	Table 137
PORT_ENA1	R/W	0x044	GPIO grouped interrupt port 1 enable register	0	Table 138

GPIO port addresses can be read and written as bytes, halfwords, or words.

Table 123. Register overview: GPIO port (base address 0x5000 0000)

Name	Access	Address offset	Description	Reset value	Width	Reference
B0 to B31	R/W	0x0000 to 0x001F	Byte pin registers port 0; pins PIO0_0 to PIO0_31	ext[1]	byte (8 bit)	Table 139
B32 to B63	R/W	0x0020 to 0x002F	Byte pin registers port 1	ext[1]	byte (8 bit)	Table 140
W0 to W31	R/W	0x1000 to 0x107C	Word pin registers port 0	ext[1]	word (32 bit)	Table 141
W32 to W63	R/W	0x1080 to 0x10FC	Word pin registers port 1	ext[1]	word (32 bit)	Table 142
DIR0	R/W	0x2000	Direction registers port 0	0	word (32 bit)	Table 143
DIR1	R/W	0x2004	Direction registers port 1	0	word (32 bit)	Table 144
MASK0	R/W	0x2080	Mask register port 0	0	word (32 bit)	Table 145
MASK1	R/W	0x2084	Mask register port 1	0	word (32 bit)	Table 146
PIN0	R/W	0x2100	Port pin register port 0	ext[1]	word (32 bit)	Table 147
PIN1	R/W	0x2104	Port pin register port 1	ext[1]	word (32 bit)	Table 148
MPIN0	R/W	0x2180	Masked port register port 0	ext[1]	word (32 bit)	Table 149
MPIN1	R/W	0x2184	Masked port register port 1	ext[1]	word (32 bit)	Table 150
SET0	R/W	0x2200	Write: Set register for port 0 Read: output bits for port 0	0	word (32 bit)	Table 151
SET1	R/W	0x2204	Write: Set register for port 1 Read: output bits for port 1	0	word (32 bit)	Table 152
CLR0	WO	0x2280	Clear port 0	NA	word (32 bit)	Table 153
CLR1	WO	0x2284	Clear port 1	NA	word (32 bit)	Table 154
NOT0	WO	0x2300	Toggle port 0	NA	word (32 bit)	Table 155
NOT1	WO	0x2304	Toggle port 1	NA	word (32 bit)	Table 156

[1] “ext” in this table and subsequent tables indicates that the data read after reset depends on the state of the pin, which in turn may depend on an external source.

9.5.1 GPIO pin interrupts register description

9.5.1.1 Pin interrupt mode register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 39), one bit in the **ISEL** register determines whether the interrupt is edge or level sensitive.

Table 124. Pin interrupt mode register (ISEL, address 0x4004 C000) bit description

Bit	Symbol	Description	Reset value	Access
7:0	PMODE	Selects the interrupt mode for each pin interrupt. Bit n configures the pin interrupt selected in PINTSELn. 0 = Edge sensitive 1 = Level sensitive	0	R/W
31:8	-	Reserved.	-	-

9.5.1.2 Pin interrupt level (rising edge interrupt) enable register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the **IENR** register enables the interrupt depending on the pin interrupt mode configured in the **ISEL** register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the rising edge interrupt is enabled.
- If the pin interrupt mode is level sensitive (PMODE = 1), the level interrupt is enabled. The PINTEN_F register configures the active level (HIGH or LOW) for this interrupt.

Table 125. Pin interrupt level (rising edge interrupt enable) register (IENR, address 0x4004 C004) bit description

Bit	Symbol	Description	Reset value	Access
7:0	ENRL	Enables the rising edge or level interrupt for each pin interrupt. Bit n configures the pin interrupt selected in PINTSELn. 0 = Disable rising edge or level interrupt. 1 = Enable rising edge or level interrupt.	0	R/W
31:8	-	Reserved.	-	-

9.5.1.3 Pin interrupt level (rising edge interrupt) set register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the **SIENR** register sets the corresponding bit in the **IENR** register depending on the pin interrupt mode configured in the PINTMODE register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the rising edge interrupt is set.
- If the pin interrupt mode is level sensitive (PMODE = 1), the level interrupt is set.

Table 126. Pin interrupt level (rising edge interrupt) set register (SIENR, address 0x4004 C008) bit description

Bit	Symbol	Description	Reset value	Access
7:0	SETENRL	Ones written to this address set bits in the PINTEN_R, thus enabling interrupts. Bit n sets bit n in the PINTEN_R register. 0 = No operation. 1 = Enable rising edge or level interrupt.	NA	WO
31:8	-	Reserved.	-	-

9.5.1.4 Pin interrupt level (rising edge interrupt) clear register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the **CIENR** register clears the corresponding bit in the **IENR** register depending on the pin interrupt mode configured in the ISEL register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the rising edge interrupt is cleared.
- If the pin interrupt mode is level sensitive (PMODE = 1), the level interrupt is cleared.

Table 127. Pin interrupt level (rising edge interrupt) clear register (PCIENR, address 0x4004 C00C) bit description

Bit	Symbol	Description	Reset value	Access
7:0	CENRL	Ones written to this address clear bits in the IENR, thus disabling the interrupts. Bit n clears bit n in the **IENR** register. 0 = No operation. 1 = Disable rising edge or level interrupt.	NA	WO
31:8	-	Reserved.	-	-

9.5.1.5 Pin interrupt active level (falling edge interrupt enable) register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the PINTSEN_F register enables the falling edge interrupt or the configures the level sensitivity depending on the pin interrupt mode configured in the ISEL register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the falling edge interrupt is enabled.
- If the pin interrupt mode is level sensitive (PMODE = 1), the active level of the level interrupt (HIGH or LOW) is configured.

Table 128. Pin interrupt active level (falling edge interrupt enable) register (IENF, address 0x4004 C010) bit description

Bit	Symbol	Description	Reset value	Access
7:0	ENAF	Enables the falling edge or configures the active level interrupt for each pin interrupt. Bit n configures the pin interrupt selected in PINTSELn. 0 = Disable falling edge interrupt or set active interrupt level LOW. 1 = Enable falling edge interrupt enabled or set active interrupt level HIGH.	0	R/W
31:8	-	Reserved.	-	-

9.5.1.6 Pin interrupt active level (falling edge interrupt) set register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the S**IENF** register sets the corresponding bit in the **IENF** register depending on the pin interrupt mode configured in the ISEL register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the falling edge interrupt is set.
- If the pin interrupt mode is level sensitive (PMODE = 1), the HIGH-active interrupt is selected.

Table 129. Pin interrupt active level (falling edge interrupt) set register (SIENF, address 0x4004 C014) bit description

Bit	Symbol	Description	Reset value	Access
7:0	SETENAF	Ones written to this address set bits in the IENF, thus enabling interrupts. Bit n sets bit n in the IENF register. 0 = No operation. 1 = Select HIGH-active interrupt or enable falling edge interrupt.	NA	WO
31:8	-	Reserved.	-	-

9.5.1.7 Pin interrupt active level (falling edge interrupt) clear register

For each of the 8 pin interrupts selected in the PINTSEL registers (see Table 39), one bit in the CIENF register sets the corresponding bit in the IENF register depending on the pin interrupt mode configured in the ISEL register:

- If the pin interrupt mode is edge sensitive (PMODE = 0), the falling edge interrupt is cleared.
- If the pin interrupt mode is level sensitive (PMODE = 1), the LOW-active interrupt is selected.

Table 130. Pin interrupt active level (falling edge interrupt) clear register (CIENF, address 0x4004 C018) bit description

Bit	Symbol	Description	Reset value	Access
7:0	CENAF	Ones written to this address clears bits in the IENF, thus disabling interrupts. Bit n clears bit n in the IENF register. 0 = No operation. 1 = LOW-active interrupt selected or falling edge interrupt disabled.	NA	WO
31:8	-	Reserved.	-	-

9.5.1.8 Pin interrupt rising edge register

This register contains ones for pin interrupts selected in the PINTSEL registers (see Table 39) on which a rising edge has been detected. Writing ones to this register clears rising edge detection. Ones in this register assert an interrupt request for pins that are enabled for rising-edge interrupts. All edges are detected for all pins selected by the PINTSEL registers, regardless of whether they are interrupt-enabled.

Table 131. Pin interrupt rising edge register (RISE, address 0x4004 C01C) bit description

Bit	Symbol	Description	Reset value	Access
7:0	RDET	Rising edge detect. Bit n detects the rising edge of the pin selected in PINTSELn. Read 0: No rising edge has been detected on this pin since Reset or the last time a one was written to this bit. Write 0: no operation. Read 1: a rising edge has been detected since Reset or the last time a one was written to this bit. Write 1: clear rising edge detection for this pin.	0	R/W
31:8	-	Reserved.	-	-

9.5.1.9 Pin interrupt falling edge register

This register contains ones for pin interrupts selected in the PINTSEL registers (see Table 39) on which a falling edge has been detected. Writing ones to this register clears falling edge detection. Ones in this register assert an interrupt request for pins that are enabled for falling-edge interrupts. All edges are detected for all pins selected by the PINTSEL registers, regardless of whether they are interrupt-enabled.

Table 132. Pin interrupt falling edge register (FALL, address 0x4004 C020) bit description

Bit	Symbol	Description	Reset value	Access
7:0	FDET	Falling edge detect. Bit n detects the falling edge of the pin selected in PINTSELn. Read 0: No falling edge has been detected on this pin since Reset or the last time a one was written to this bit. Write 0: no operation. Read 1: a falling edge has been detected since Reset or the last time a one was written to this bit. Write 1: clear falling edge detection for this pin.	0	R/W
31:8	-	Reserved.	-	-

9.5.1.10 Pin interrupt status register

Reading this register returns ones for pin interrupts that are currently requesting an interrupt. For pins identified as edge-sensitive in the Interrupt Select register, writing ones to this register clears both rising- and falling-edge detection for the pin. For level-sensitive pins, writing ones inverts the corresponding bit in the Active level register, thus switching the active level on the pin.

Table 133. Pin interrupt status register (IST address 0x4004 C024) bit description

Bit	Symbol	Description	Reset value	Access
7:0	PSTAT	Pin interrupt status. Bit n returns the status, clears the edge interrupt, or inverts the active level of the pin selected in PINTSELn. Read 0: interrupt is not being requested for this interrupt pin. Write 0: no operation. Read 1: interrupt is being requested for this interrupt pin. Write 1 (edge-sensitive): clear rising- and falling-edge detection for this pin. Write 1 (level-sensitive): switch the active level for this pin (in the PINTENT_F register).	0	R/W
31:8	-	Reserved.	-	-

9.5.2 GPIO GROUP0/GROUP1 interrupt register description

9.5.2.1 Grouped interrupt control register

Table 134. GPIO grouped interrupt control register (CTRL, addresses 0x4005 C000 (GROUP0 INT) and 0x4006 0000 (GROUP1 INT)) bit description

Bit	Symbol	Value	Description	Reset value
0	INT		Group interrupt status. This bit is cleared by writing a one to it. Writing zero has no effect.	0
		0	No interrupt request is pending.	
		1	Interrupt request is active.	
1	COMB		Combine enabled inputs for group interrupt	0
		0	OR functionality: A grouped interrupt is generated when any one of the enabled inputs is active (based on its programmed polarity).	
		1	AND functionality: An interrupt is generated when all enabled bits are active (based on their programmed polarity).	
2	TRIG		Group interrupt trigger	0
		0	Edge-triggered	
		1	Level-triggered	
31:3	-	-	Reserved	0

9.5.2.2 GPIO grouped interrupt port polarity registers

The grouped interrupt port polarity registers determine how the polarity of each enabled pin contributes to the grouped interrupt. Each port is associated with its own port polarity register, and the values of both registers together determine the grouped interrupt.

Table 135. GPIO grouped interrupt port 0 polarity registers (PORT_POL0, addresses 0x4005 C020 (GROUP0 INT) and 0x4006 0020 (GROUP1 INT)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	POL0	Configure pin polarity of port 0 pins for group interrupt. Bit n corresponds to pin P0_n of port 0. 0 = the pin is active LOW. If the level on this pin is LOW, the pin contributes to the group interrupt. 1 = the pin is active HIGH. If the level on this pin is HIGH, the pin contributes to the group interrupt.	1	-

Table 136. GPIO grouped interrupt port 1 polarity registers (PORT_POL1, addresses 0x4005 C024 (GROUP0 INT) and 0x4006 0024 (GROUP1 INT)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	POL1	Configure pin polarity of port 1 pins for group interrupt. Bit n corresponds to pin P1_n of port 1. 0 = the pin is active LOW. If the level on this pin is LOW, the pin contributes to the group interrupt. 1 = the pin is active HIGH. If the level on this pin is HIGH, the pin contributes to the group interrupt.	1	-

9.5.2.3 GPIO grouped interrupt port enable registers

The grouped interrupt port enable registers enable the pins which contribute to the grouped interrupt. Each port is associated with its own port enable register, and the values of both registers together determine which pins contribute to the grouped interrupt.

Table 137. GPIO grouped interrupt port 0 enable registers (PORT_ENA0, addresses 0x4005 C040 (GROUP0 INT) and 0x4006 0040 (GROUP1 INT)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	ENA0	Enable port 0 pin for group interrupt. Bit n corresponds to pin P0_n of port 0. 0 = the port 0 pin is disabled and does not contribute to the grouped interrupt. 1 = the port 0 pin is enabled and contributes to the grouped interrupt.	0	-

Table 138. GPIO grouped interrupt port 1 enable registers (PORT_ENA1, addresses 0x4005 C044 (GROUP0 INT) and 0x4006 0044 (GROUP1 INT)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	ENA1	Enable port 1 pin for group interrupt. Bit n corresponds to pin P1_n of port 0. 0 = the port 1 pin is disabled and does not contribute to the grouped interrupt. 1 = the port 1 pin is enabled and contributes to the grouped interrupt.	0	-

9.5.3 GPIO port register description

9.5.3.1 GPIO port byte pin registers

Each GPIO pin has a byte register in this address range. Software typically reads and writes bytes to access individual pins, but can read or write halfwords to sense or set the state of two pins, and read or write words to sense or set the state of four pins.

Table 139. GPIO port 0 byte pin registers (B0 to B31, addresses 0x5000 0000 to 0x5000 001F) bit description

Bit	Symbol	Description	Reset value	Access
0	PBYTE	Read: state of the pin P0_n, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as 0. Write: loads the pin's output bit.	ext	R/W
7:1		Reserved (0 on read, ignored on write)	0	-

Table 140. GPIO port 1 byte pin registers (B32 to B63, addresses 0x5000 0020 to 0x5000 002F) bit description

Bit	Symbol	Description	Reset value	Access
0	PBYTE	Read: state of the pin P1_n, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as 0. Write: loads the pin's output bit.	ext	R/W
7:1		Reserved (0 on read, ignored on write)	0	-

9.5.3.2 GPIO port word pin registers

Each GPIO pin has a word register in this address range. Any byte, halfword, or word read in this range will be all zeros if the pin is low or all ones if the pin is high, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as zeros. Any write will clear the pin's output bit if the value written is all zeros, else it will set the pin's output bit.

Table 141. GPIO port 0 word pin registers (W0 to W31, addresses 0x5000 1000 to 0x5000 107C) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PWORD	Read 0: pin is LOW. Write 0: clear output bit. Read 0xFFFF FFFF: pin is HIGH. Write any value 0x0000 0001 to 0xFFFF FFFF: set output bit. **Remark:** Only 0 or 0xFFFF FFFF can be read. Writing any value other than 0 will set the output bit.	ext	R/W

Table 142. GPIO port 1 word pin registers (W32 to W63, addresses 0x5000 1080 to 0x5000 10FC) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PWORD	Read 0: pin is LOW. Write 0: clear output bit. Read 0xFFFF FFFF: pin is HIGH. Write any value 0x0000 0001 to 0xFFFF FFFF: set output bit. **Remark:** Only 0 or 0xFFFF FFFF can be read. Writing any value other than 0 will set the output bit.	ext	R/W

9.5.3.3 GPIO port direction registers

Each GPIO port has one direction register for configuring the port pins as inputs or outputs.

Table 143. GPIO direction port 0 register (DIR0, address 0x5000 2000) bit description

Bit	Symbol	Description	Reset value	Access
31:0	DIRP0	Selects pin direction for pin P0_n (bit 0 = P0_0, bit 1 = P0_1, ..., bit 31 = P0_31). 0 = input. 1 = output.	0	R/W

Table 144. GPIO direction port 1 register (DIR1, address 0x5000 2004) bit description

Bit	Symbol	Description	Reset value	Access
31:0	DIRP1	Selects pin direction for pin P1_n (bit 0 = P1_0, bit 1 = P1_1, ..., bit 31 = P1_31). 0 = input. 1 = output.	0	R/W

9.5.3.4 GPIO port mask registers

These registers affect writing and reading the MPORT registers. Zeroes in these registers enable reading and writing; ones disable writing and result in zeros in corresponding positions when reading.

Table 145. GPIO mask port 0 register (MASK0, address 0x5000 2080) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MASKP0	Controls which bits corresponding to P0_n are active in the P0MPORT register (bit 0 = P0_0, bit 1 = P0_1, ..., bit 31 = P0_31). 0 = Read MPORT: pin state; write MPORT: load output bit. 1 = Read MPORT: 0; write MPORT: output bit not affected.	0	R/W

Table 146. GPIO mask port 1 register (MASK1, address 0x5000 2084) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MASKP1	Controls which bits corresponding to P1_n are active in the P1MPORT register (bit 0 = P1_0, bit 1 = P1_1, ..., bit 31 = P1_31). 0 = Read MPORT: pin state; write MPORT: load output bit. 1 = Read MPORT: 0; write MPORT: output bit not affected.	0	R/W

9.5.3.5 GPIO port pin registers

Reading these registers returns the current state of the pins read, regardless of direction, masking, or alternate functions, except that pins configured as analog I/O always read as 0s. Writing these registers loads the output bits of the pins written to, regardless of the Mask register.

Table 147. GPIO port 0 pin register (PIN0, address 0x5000 2100) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PORT0	Reads pin states or loads output bits (bit 0 = P0_0, bit 1 = P0_1, ..., bit 31 = P0_31). 0 = Read: pin is low; write: clear output bit. 1 = Read: pin is high; write: set output bit.	ext	R/W

Table 148. GPIO port 1 pin register (PIN1, address 0x5000 2104) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PORT1	Reads pin states or loads output bits (bit 0 = P1_0, bit 1 = P1_1, ..., bit 31 = P1_31). 0 = Read: pin is low; write: clear output bit. 1 = Read: pin is high; write: set output bit.	ext	R/W

9.5.3.6 GPIO masked port pin registers

These registers are similar to the PORT registers, except that the value read is masked by ANDing with the inverted contents of the corresponding MASK register, and writing to one of these registers only affects output register bits that are enabled by zeros in the corresponding MASK register

Table 149. GPIO masked port 0 pin register (MPIN0, address 0x5000 2180) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MPORTP0	Masked port register (bit 0 = P0_0, bit 1 = P0_1, ..., bit 31 = P0_31). 0 = Read: pin is LOW and/or the corresponding bit in the MASK register is 1; write: clear output bit if the corresponding bit in the MASK register is 0. 1 = Read: pin is HIGH and the corresponding bit in the MASK register is 0; write: set output bit if the corresponding bit in the MASK register is 0.	ext	R/W

Table 150. GPIO masked port 1 pin register (MPIN1, address 0x5000 2184) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MPORTP1	Masked port register (bit 0 = P1_0, bit 1 = P1_1, ..., bit 31 = P1_31). 0 = Read: pin is LOW and/or the corresponding bit in the MASK register is 1; write: clear output bit if the corresponding bit in the MASK register is 0. 1 = Read: pin is HIGH and the corresponding bit in the MASK register is 0; write: set output bit if the corresponding bit in the MASK register is 0.	ext	R/W

9.5.3.7 GPIO port set registers

Output bits can be set by writing ones to these registers, regardless of MASK registers. Reading from these register returns the port's output bits, regardless of pin directions.

Table 151. GPIO set port 0 register (SET0, address 0x5000 2200) bit description

Bit	Symbol	Description	Reset value	Access
31:0	SETP0	Read or set output bits. 0 = Read: output bit: write: no operation. 1 = Read: output bit; write: set output bit.	0	R/W

Table 152. GPIO set port 1 register (SET1, address 0x5000 2204) bit description

Bit	Symbol	Description	Reset value	Access
31:0	SETP1	Read or set output bits. 0 = Read: output bit: write: no operation. 1 = Read: output bit; write: set output bit.	0	R/W

9.5.3.8 GPIO port clear registers

Output bits can be cleared by writing ones to these write-only registers, regardless of MASK registers.

Table 153. GPIO clear port 0 register (CLR0, address 0x5000 2280) bit description

Bit	Symbol	Description	Reset value	Access
31:0	CLRP0	Clear output bits: 0 = No operation. 1 = Clear output bit.	NA	WO

Table 154. GPIO clear port 1 register (CLR1, address 0x5000 2284) bit description

Bit	Symbol	Description	Reset value	Access
31:0	CLRP1	Clear output bits: 0 = No operation. 1 = Clear output bit.	NA	WO

9.5.3.9 GPIO port toggle registers

Output bits can be toggled/inverted/complemented by writing ones to these write-only registers, regardless of MASK registers.

Table 155. GPIO toggle port 0 register (NOT0, address 0x5000 2300) bit description

Bit	Symbol	Description	Reset value	Access
31:0	NOTP0	Toggle output bits: 0 = no operation. 1 = Toggle output bit.	NA	WO

Table 156. GPIO toggle port 1 register (NOT1, address 0x5000 2304) bit description

Bit	Symbol	Description	Reset value	Access
31:0	NOTP1	Toggle output bits: 0 = no operation. 1 = Toggle output bit.	NA	WO

9.6 Functional description

9.6.1 Reading pin state

Software can read the state of all GPIO pins except those selected for analog input or output in the “I/O Configuration” logic. A pin does not have to be selected for GPIO in “I/O Configuration” in order to read its state. There are four ways to read pin state:

- The state of a single pin can be read with 7 high-order zeros from a Byte Pin register.
- The state of a single pin can be read in all bits of a byte, halfword, or word from a Word Pin register.
- The state of multiple pins in a port can be read as a byte, halfword, or word from a PORT register.
- The state of a selected subset of the pins in a port can be read from a Masked Port (MPORT) register. Pins having a 1 in the port’s Mask register will read as 0 from its MPORT register.

9.6.2 GPIO output

Each GPIO pin has an output bit in the GPIO block. These output bits are the targets of write operations “to the pins”. Two conditions must be met in order for a pin’s output bit to be driven onto the pin:

1. The pin must be selected for GPIO operation in the “I/O Configuration” block, and
2. the pin must be selected for output by a 1 in its port’s DIR register.

If either or both of these conditions is (are) not met, “writing to the pin” has no effect.

There are seven ways to change GPIO output bits:

- Writing to a Byte Pin register loads the output bit from the least significant bit.
- Writing to a Word Pin register loads the output bit with the OR of all of the bits written. (This feature follows the definition of “truth” of a multi-bit value in programming languages.)
- Writing to a port’s PORT register loads the output bits of all the pins written to.

- Writing to a port's MPORT register loads the output bits of pins identified by zeros in corresponding positions of the port's MASK register.
- Writing ones to a port's SET register sets output bits.
- Writing ones to a port's CLR register clears output bits.
- Writing ones to a port's NOT register toggles/complements/inverts output bits.

The state of a port's output bits can be read from its SET register. Reading any of the registers described in 9.6.1 returns the state of pins, regardless of their direction or alternate functions.

9.6.3 Masked I/O

A port's MASK register defines which of its pins should be accessible in its MPORT register. Zeroes in MASK enable the corresponding pins to be read from and written to MPORT. Ones in MASK force a pin to read as 0 and its output bit to be unaffected by writes to MPORT. When a port's MASK register contains all zeros, its PORT and MPORT registers operate identically for reading and writing.

Users of previous NXP devices with similar GPIO blocks should be aware of an incompatibility: on the LPC11A1x, writing to the SET, CLR, and NOT registers is not affected by the MASK register. On previous devices these registers were masked.

Applications in which interrupts can result in Masked GPIO operation, or in task switching among tasks that do Masked GPIO operation, must treat code that uses the Mask register as a protected/restricted region. This can be done by interrupt disabling or by using a semaphore.

The simpler way to protect a block of code that uses a MASK register is to disable interrupts before setting the MASK register, and re-enable them after the last operation that uses the MPORT or MASK register.

More efficiently, software can dedicate a semaphore to the MASK registers, and set/capture the semaphore controlling exclusive use of the MASK registers before setting the MASK registers, and release the semaphore after the last operation that uses the MPORT or MASK registers.

9.6.4 GPIO Interrupts

Two separate GPIO interrupt facilities are provided. With pin interrupts, up to eight GPIO pins can each have separately-vectored, edge- or level-sensitive interrupts.

With group interrupts, any subset of the pins in each port can be selected to contribute to a common interrupt. Any of the pin and port interrupts can be enabled to wake the part from Deep-sleep mode or Power-down mode.

9.6.4.1 Pin interrupts

In this interrupt facility, up to 8 pins are identified as interrupt sources by the Pin Interrupt Select registers (PINTSEL0-7). All of the other Pin Interrupt registers contain 8 bits, corresponding to the pins called out by the PINTSEL0-7 registers. The PINTMODE register defines whether each interrupt pin is edge- or level-sensitive. The PINTRISE and

PINTFALL registers detect edges on each interrupt pin, and can be written to clear (and set) edge detection. The PINTST register indicates whether each interrupt pin is currently requesting an interrupt, and PINTST can be written to clear interrupts.

The other pin interrupt registers play different roles for edge-sensitive and level-sensitive pins, as described in Table 157.

Table 157. Pin interrupt registers for edge- and level-sensitive pins

Name	Edge-sensitive function	Level-sensitive function
PINTEN_R	Enables rising-edge interrupts.	Enables interrupts.
PINTSEN_R	Write to enable rising-edge interrupts.	Write to enable interrupts.
PINTCEN_R	Write to disable rising-edge interrupts.	Write to disable interrupts.
PINTEN_F	Enables falling-edge interrupts.	Selects active level.
PINTSEN_F	Write to enable falling-edge interrupts.	Write to select high-active.
PINTCEN_F	Write to disable falling-edge interrupts.	Write to select low-active.

9.6.4.2 Group interrupts

In this interrupt facility, an interrupt can be requested for each port, based on any selected subset of pins within each port. The pins that contribute to each port interrupt are selected by 1s in the port's Enable register, and an interrupt polarity can be selected for each pin in the port's Polarity register. The level on each pin is exclusive-ORed with its polarity bit and the result is ANDed with its enable bit, and these results are then inclusive-ORed among all the pins in the port, to create the port's raw interrupt request.

The raw interrupt request from each of the two group interrupts is sent to the NVIC, which can be programmed to treat it as level- or edge-sensitive (see Section 6.4), or it can be edge-detected by the wake-up interrupt logic (see Section 3.5.36).

9.6.5 Recommended practices

The following lists some recommended uses for using the GPIO port registers:

- For initial setup after Reset or re-initialization, write the PORT register(s).
- To change the state of one pin, write a Byte Pin or Word Pin register.
- To change the state of multiple pins at a time, write the SET and/or CLR registers.
- To change the state of multiple pins in a tightly controlled environment like a software state machine, consider using the NOT register. This can require less write operations than SET and CLR.
- To read the state of one pin, read a Byte Pin or Word Pin register.
- To make a decision based on multiple pins, read and mask a PORT register.

10.1 How to read this chapter

The USB on-chip drivers are available on parts LPC11U2x only.

10.2 Introduction

The boot ROM contains a USB driver to simplify the USB application development. The USB driver implements the Communication Device Class (CDC), the Human Interface Device (HID), and the Mass Storage Device (MSC) device class. Only one device function can be used by the application software.

10.3 USB driver functions

The USB device driver ROM API consists of the following modules:

- Communication Device Class (CDC) function driver
  - Communication Device Class function driver initialization parameter data structure (Table 185 “USBD_CDC_INIT_PARAM class structure”).
  - CDC class API functions structure. This module exposes functions which interact directly with USB device controller hardware (Table 184 “USBD_CDC_API class structure”).
- USB core layer
  - struct (Table 181 “_WB_T class structure”)
  - union (Table 158 “__WORD_BYTE class structure”)
  - struct (Table 159 “_BM_T class structure”)
  - struct (Table 172 “_REQUEST_TYPE class structure”)
  - struct (Table 179 “_USB_SETUP_PACKET class structure”)
  - struct (Table 175 “_USB_DEVICE_QUALIFIER_DESCRIPTOR class structure”)
  - struct USB device descriptor
  - struct (Table 175 “_USB_DEVICE_QUALIFIER_DESCRIPTOR class structure”)
  - struct USB configuration descriptor
  - struct (Table 177 “_USB_INTERFACE_DESCRIPTOR class structure”)
  - struct USB endpoint descriptor
  - struct (Table 180 “_USB_STRING_DESCRIPTOR class structure”)
  - struct (Table 173 “_USB_COMMON_DESCRIPTOR class structure”)
  - struct (Table 178 “_USB_OTHER_SPEED_CONFIGURATION class structure”)
  - USB descriptors data structure (Table 174 “_USB_CORE_DESCS_T class structure”)
  - USB device stack initialization parameter data structure (Table 183 “USBD_API_INIT_PARAM class structure”).

- USB device stack core API functions structure (Table 186 “USBD_CORE_API class structure”).

- Device Firmware Upgrade (DFU) class function driver
  - DFU descriptors data structure (Table 188 “USBD_DFU_INIT_PARAM class structure”).
  - DFU class API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 187 “USBD_DFU_API class structure”).
- HID class function driver
  - struct (Table 167 “_HID_DESCRIPTOR class structure”).
  - struct (Table 169 “_HID_REPORT_T class structure”).
  - USB descriptors data structure (Table 190 “USBD_HID_INIT_PARAM class structure”).
  - HID class API functions structure. This structure contains pointers to all the functions exposed by the HID function driver module (Table 191 “USBD_HW_API class structure”).
- USB device controller driver
  - Hardware API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 191 “USBD_HW_API class structure”).
- Mass Storage Class (MSC) function driver
  - Mass Storage Class function driver initialization parameter data structure (Table 193).
  - MSC class API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 192).

10.4 Calling the USB device driver

A fixed location in ROM contains a pointer to the ROM driver table i.e. 0x1FFF 1FF8. The ROM driver table contains a pointer to the USB driver table. Pointers to the various USB driver functions are stored in this table. USB driver functions can be called by using a C structure. Figure 17 illustrates the pointer mechanism used to access the on-chip USB driver.

```
typedef struct USBD_API

{

const USBD_HW_API_T* hw;

const USBD_CORE_API_T* core;

const USBD_MSC_API_T* msc;

const USBD_DFU_API_T* dfu;

const USBD_HID_API_T* hid;

const USBD_CDC_API_T* cdc;
```

```
const uint32_t* reserved6;
const uint32_t version;
} USBD_API_T;
```

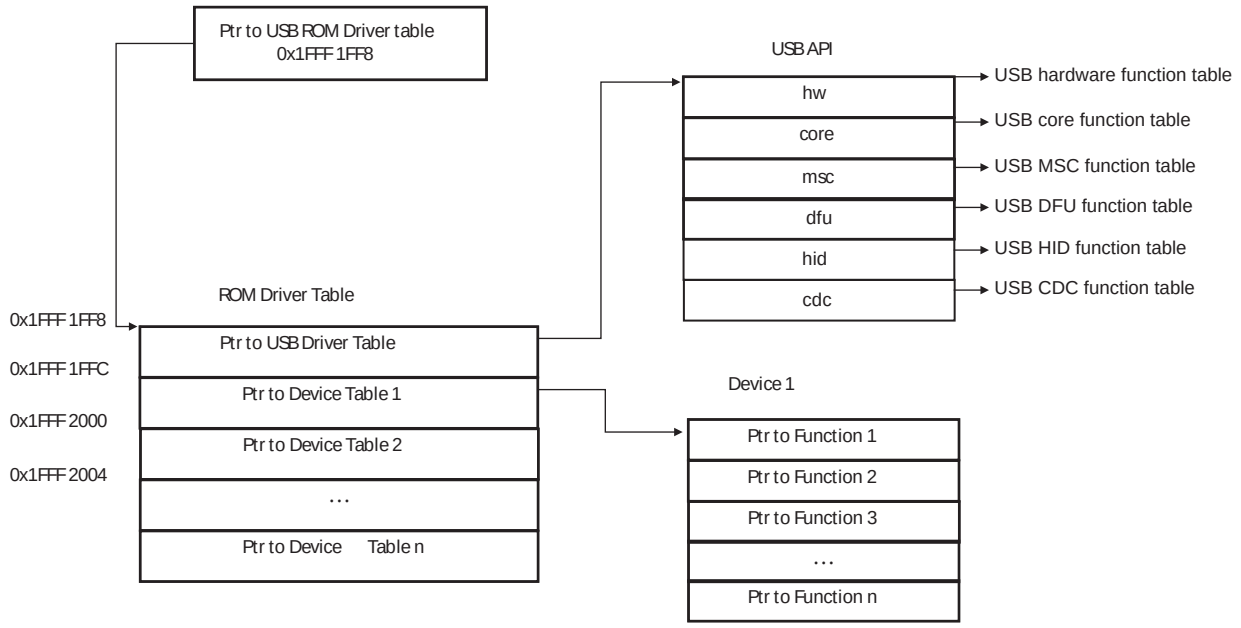


Fig 17. USB device driver pointer structure

10.5 USB API

10.5.1 __WORD_BYTE

Table 158. __WORD_BYTE class structure

Member	Description
W	`uint16_tuint16_t __WORD_BYTE::W` data member to do 16 bit access
WB	`WB_TWB_T __WORD_BYTE::WB` data member to do 8 bit access

10.5.2 _BM_T

Table 159. _BM_T class structure

Member	Description
Recipient	uint8_tuint8_t _BM_T::Recipient Recipient type.
Type	uint8_tuint8_t _BM_T::Type Request type.
Dir	uint8_tuint8_t _BM_T::Dir Direction type.

10.5.3 _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR

Table 160. _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_tuint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_tuint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_tuint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bDescriptorSubtype
bmCapabilities	uint8_tuint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bmCapabilities

10.5.4 _CDC_CALL_MANAGEMENT_DESCRIPTOR

Table 161. _CDC_CALL_MANAGEMENT_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_tuint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_tuint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_tuint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDescriptorSubtype
bmCapabilities	uint8_tuint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bmCapabilities
bDataInterface	uint8_tuint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDataInterface

10.5.5 _CDC_HEADER_DESCRIPTOR

Table 162. _CDC_HEADER_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_tuint8_t _CDC_HEADER_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_tuint8_t _CDC_HEADER_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_tuint8_t _CDC_HEADER_DESCRIPTOR::bDescriptorSubtype
bcdCDC	uint16_tuint16_t _CDC_HEADER_DESCRIPTOR::bcdCDC

10.5.6 _CDC_LINE_CODING

Table 163. _CDC_LINE_CODING class structure

Member	Description
dwDTERate	uint32_tuint32_t _CDC_LINE_CODING::dwDTERate
bCharFormat	uint8_tuint8_t _CDC_LINE_CODING::bCharFormat
bParityType	uint8_tuint8_t _CDC_LINE_CODING::bParityType
bDataBits	uint8_tuint8_t _CDC_LINE_CODING::bDataBits

10.5.7 _CDC_UNION_1SLAVE_DESCRIPTOR

Table 164. _CDC_UNION_1SLAVE_DESCRIPTOR class structure

Member	Description
sUnion	CDC_UNION_DESCRIPTORCDC_UNION_DESCRIPTOR _CDC_UNION_1SLAVE_DESCRIPTOR::sUnion
bSlaveInterfaces	uint8_tuint8_t _CDC_UNION_1SLAVE_DESCRIPTOR::bSlaveInterfaces[1][1]

10.5.8 _CDC_UNION_DESCRIPTOR

Table 165. _CDC_UNION_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_tuint8_t _CDC_UNION_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_tuint8_t _CDC_UNION_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_tuint8_t _CDC_UNION_DESCRIPTOR::bDescriptorSubtype
bMasterInterface	uint8_tuint8_t _CDC_UNION_DESCRIPTOR::bMasterInterface

10.5.9 _DFU_STATUS

Table 166. _DFU_STATUS class structure

Member	Description
bStatus	uint8_tuint8_t _DFU_STATUS::bStatus
bwPollTimeout	uint8_tuint8_t _DFU_STATUS::bwPollTimeout[3][3]
bState	uint8_tuint8_t _DFU_STATUS::bState
iString	uint8_tuint8_t _DFU_STATUS::iString

10.5.10 _HID_DESCRIPTOR

HID class-specific HID Descriptor.

Table 167. _HID_DESCRIPTOR class structure

Member	Description
bLength	`uint8_tuint8_t _HID_DESCRIPTOR::bLength` Size of the descriptor, in bytes.
bDescriptorType	`uint8_tuint8_t _HID_DESCRIPTOR::bDescriptorType` Type of HID descriptor.
bcdHID	`uint16_tuint16_t _HID_DESCRIPTOR::bcdHID` BCD encoded version that the HID descriptor and device complies to.
bCountryCode	`uint8_tuint8_t _HID_DESCRIPTOR::bCountryCode` Country code of the localized device, or zero if universal.
bNumDescriptors	`uint8_tuint8_t _HID_DESCRIPTOR::bNumDescriptors` Total number of HID report descriptors for the interface.
DescriptorList	`PRE_PACK struct POST_PACK _HID_DESCRIPTOR::_HID_DESCRIPTOR_LISTPRE_PACK struct POST_PACK _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST _HID_DESCRIPTOR::DescriptorList[1][1]` Array of one or more descriptors

10.5.11 _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST

Table 168. _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST class structure

Member	Description
bDescriptorType	`uint8_tuint8_t _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST::bDescriptorType` Type of HID report.
wDescriptorLength	`uint16_tuint16_t _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST::wDescriptorLength` Length of the associated HID report descriptor, in bytes.

10.5.12 _HID_REPORT_T

HID report descriptor data structure.

Table 169. _HID_REPORT_T class structure

Member	Description
len	`uint16_tuint16_t _HID_REPORT_T::len` Size of the report descriptor in bytes.
idle_time	`uint8_tuint8_t _HID_REPORT_T::idle_time` This value is used by stack to respond to Set_Idle & GET_Idle requests for the specified report ID. The value of this field specified the rate at which duplicate reports are generated for the specified Report ID. For example, a device with two input reports could specify an idle rate of 20 milliseconds for report ID 1 and 500 milliseconds for report ID 2.
__pad	`uint8_tuint8_t _HID_REPORT_T::__pad` Padding space.
desc	`uint8_t *uint8_t* _HID_REPORT_T::desc` Report descriptor.

10.5.13 _MSC_CBW

Table 170. _MSC_CBW class structure

Member	Description
dSignature	uint32_tuint32_t _MSC_CBW::dSignature
dTag	uint32_tuint32_t _MSC_CBW::dTag
dDataLength	uint32_tuint32_t _MSC_CBW::dDataLength
bmFlags	uint8_tuint8_t _MSC_CBW::bmFlags
bLUN	uint8_tuint8_t _MSC_CBW::bLUN
bCBLength	uint8_tuint8_t _MSC_CBW::bCBLength
CB	uint8_tuint8_t _MSC_CBW::CB[16][16]

10.5.14 _MSC_CSW

Table 171. _MSC_CSW class structure

Member	Description
dSignature	uint32_tuint32_t _MSC_CSW::dSignature
dTag	uint32_tuint32_t _MSC_CSW::dTag
dDataResidue	uint32_tuint32_t _MSC_CSW::dDataResidue
bStatus	uint8_tuint8_t _MSC_CSW::bStatus

10.5.15 _REQUEST_TYPE

Table 172. _REQUEST_TYPE class structure

Member	Description
B	uint8_tuint8_t _REQUEST_TYPE::B byte wide access member
BM	BM_TBM_T _REQUEST_TYPE::BM bitfield structure access member

10.5.16 _USB_COMMON_DESCRIPTOR

Table 173. _USB_COMMON_DESCRIPTOR class structure

Member	Description
bLength	uint8_tuint8_t _USB_COMMON_DESCRIPTOR::bLength Size of this descriptor in bytes
bDescriptorType	uint8_tuint8_t _USB_COMMON_DESCRIPTOR::bDescriptorType Descriptor Type

10.5.17 _USB_CORE_DESCS_T

USB descriptors data structure.

Table 174. _USB_CORE_DESCS_T class structure

Member	Description
device_desc	`uint8_t *uint8_t* _USB_CORE_DESCS_T::device_desc` Pointer to USB device descriptor
string_desc	`uint8_t *uint8_t* _USB_CORE_DESCS_T::string_desc` Pointer to array of USB string descriptors
full_speed_desc	`uint8_t *uint8_t* _USB_CORE_DESCS_T::full_speed_desc` Pointer to USB device configuration descriptor when device is operating in full speed mode.
high_speed_desc	`uint8_t *uint8_t* _USB_CORE_DESCS_T::high_speed_desc` Pointer to USB device configuration descriptor when device is operating in high speed mode. For full-speed only implementation this pointer should be same as full_speed_desc.
device_qualifier	`uint8_t *uint8_t* _USB_CORE_DESCS_T::device_qualifier` Pointer to USB device qualifier descriptor. For full-speed only implementation this pointer should be set to null (0).

10.5.18 _USB_DEVICE_QUALIFIER_DESCRIPTOR

Table 175. _USB_DEVICE_QUALIFIER_DESCRIPTOR class structure

Member	Description
bLength	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bLength` Size of descriptor
bDescriptorType	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDescriptorType` Device Qualifier Type
bcdUSB	`uint16_tuint16_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bcdUSB` USB specification version number (e.g., 0200H for V2.00)
bDeviceClass	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceClass` Class Code
bDeviceSubClass	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceSubClass` SubClass Code
bDeviceProtocol	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceProtocol` Protocol Code
bMaxPacketSize0	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bMaxPacketSize0` Maximum packet size for other speed
bNumConfigurations	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bNumConfigurations` Number of Other-speed Configurations
bReserved	`uint8_tuint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bReserved` Reserved for future use, must be zero

10.5.19 _USB_DFU_FUNC_DESCRIPTOR

Table 176. _USB_DFU_FUNC_DESCRIPTOR class structure

Member	Description
bLength	`uint8_tuint8_t _USB_DFU_FUNC_DESCRIPTOR::bLength`
bDescriptorType	`uint8_tuint8_t _USB_DFU_FUNC_DESCRIPTOR::bDescriptorType`
bmAttributes	`uint8_tuint8_t _USB_DFU_FUNC_DESCRIPTOR::bmAttributes`
wDetachTimeOut	`uint16_tuint16_t _USB_DFU_FUNC_DESCRIPTOR::wDetachTimeOut`
wTransferSize	`uint16_tuint16_t _USB_DFU_FUNC_DESCRIPTOR::wTransferSize`
bcdDFUVersion	`uint16_tuint16_t _USB_DFU_FUNC_DESCRIPTOR::bcdDFUVersion`

10.5.20 _USB_INTERFACE_DESCRIPTOR

Table 177. _USB_INTERFACE_DESCRIPTOR class structure

Member	Description
bLength	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bLength` Size of this descriptor in bytes
bDescriptorType	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bDescriptorType` INTERFACE Descriptor Type
bInterfaceNumber	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceNumber` Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
bAlternateSetting	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bAlternateSetting` Value used to select this alternate setting for the interface identified in the prior field
bNumEndpoints	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bNumEndpoints` Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the Default Control Pipe.
bInterfaceClass	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceClass` Class code (assigned by the USB-IF).
bInterfaceSubClass	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceSubClass` Subclass code (assigned by the USB-IF).
bInterfaceProtocol	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceProtocol` Protocol code (assigned by the USB).
iInterface	`uint8_tuint8_t _USB_INTERFACE_DESCRIPTOR::iInterface` Index of string descriptor describing this interface

10.5.21 _USB_OTHER_SPEED_CONFIGURATION

Table 178. _USB_OTHER_SPEED_CONFIGURATION class structure

Member	Description
bLength	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bLength` Size of descriptor
bDescriptorType	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bDescriptorType` Other_speed_Configuration Type
wTotalLength	`uint16_tuint16_t _USB_OTHER_SPEED_CONFIGURATION::wTotalLength` Total length of data returned
bNumInterfaces	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bNumInterfaces` Number of interfaces supported by this speed configuration
bConfigurationValue	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bConfigurationValue` Value to use to select configuration
IConfiguration	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::IConfiguration` Index of string descriptor
bmAttributes	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bmAttributes` Same as Configuration descriptor
bMaxPower	`uint8_tuint8_t _USB_OTHER_SPEED_CONFIGURATION::bMaxPower` Same as Configuration descriptor

10.5.22 _USB_SETUP_PACKET

Table 179. _USB_SETUP_PACKET class structure

Member	Description
bmRequestType	`REQUEST_TYPEREQUEST_TYPE _USB_SETUP_PACKET::bmRequestType` This bit-mapped field identifies the characteristics of the specific request. _BM_T.
bRequest	`uint8_tuint8_t _USB_SETUP_PACKET::bRequest` This field specifies the particular request. The Type bits in the bmRequestType field modify the meaning of this field. USBD_REQUEST.
wValue	`WORD_BYTEWORD_BYTE _USB_SETUP_PACKET::wValue` Used to pass a parameter to the device, specific to the request.
wIndex	`WORD_BYTEWORD_BYTE _USB_SETUP_PACKET::wIndex` Used to pass a parameter to the device, specific to the request. The wIndex field is often used in requests to specify an endpoint or an interface.
wLength	`uint16_tuint16_t _USB_SETUP_PACKET::wLength` This field specifies the length of the data transferred during the second phase of the control transfer.

10.5.23 _USB_STRING_DESCRIPTOR

Table 180. _USB_STRING_DESCRIPTOR class structure

Member	Description
bLength	`uint8_tuint8_t _USB_STRING_DESCRIPTOR::bLength` Size of this descriptor in bytes
bDescriptorType	`uint8_tuint8_t _USB_STRING_DESCRIPTOR::bDescriptorType` STRING Descriptor Type
bString	`uint16_tuint16_t _USB_STRING_DESCRIPTOR::bString` UNICODE encoded string

10.5.24 _WB_T

Table 181. _WB_T class structure

Member	Description
L	`uint8_tuint8_t _WB_T::L` lower byte
H	`uint8_tuint8_t _WB_T::H` upper byte

10.5.25 USBD_API

Main USBD API functions structure.This structure contains pointer to various USB Device stack's sub-module function tables. This structure is used as main entry point to access various methods (grouped in sub-modules) exposed by ROM based USB device stack.

Table 182. USBD_API class structure

Member	Description
hw	`const USBD_HW_API_T *const USBD_HW_API_T* USBD_API::hw` Pointer to function table which exposes functions which interact directly with USB device stack's core layer.
core	`const USBD_CORE_API_T *const USBD_CORE_API_T* USBD_API::core` Pointer to function table which exposes functions which interact directly with USB device controller hardware.
msc	`const USBD_MSC_API_T *const USBD_MSC_API_T* USBD_API::msc` Pointer to function table which exposes functions provided by MSC function driver module.
dfu	`const USBD_DFU_API_T *const USBD_DFU_API_T* USBD_API::dfu` Pointer to function table which exposes functions provided by DFU function driver module.
hid	`const USBD_HID_API_T *const USBD_HID_API_T* USBD_API::hid` Pointer to function table which exposes functions provided by HID function driver module.

Table 182. USBD_API class structure

Member	Description
cdc	`const USBD_CDC_API_T *const USBD_CDC_API_T* USBD_API::cdc` Pointer to function table which exposes functions provided by CDC-ACM function driver module.
reserved6	`const uint32_t *const uint32_t* USBD_API::reserved6` Reserved for future function driver module.
version	`const uint32_tconst uint32_t USBD_API::version` Version identifier of USB ROM stack. The version is defined as 0x0CHDMhCC where each nibble represents version number of the corresponding component. CC - 7:0 - 8bit core version number h - 11:8 - 4bit hardware interface version number M - 15:12 - 4bit MSC class module version number D - 19:16 - 4bit DFU class module version number H - 23:20 - 4bit HID class module version number C - 27:24 - 4bit CDC class module version number H - 31:28 - 4bit reserved

10.5.26 USBD_API_INIT_PARAM

USB device stack initialization parameter data structure.

Table 183. USBD_API_INIT_PARAM class structure

Member	Description
usb_reg_base	`uint32_tuint32_t USBD_API_INIT_PARAM::usb_reg_base` USB device controller's base register address.
mem_base	`uint32_tuint32_t USBD_API_INIT_PARAM::mem_base` Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 2048 byte boundary.
mem_size	`uint32_tuint32_t USBD_API_INIT_PARAM::mem_size` The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_HW_API::GetMemSize() routine.
max_num_ep	`uint8_tuint8_t USBD_API_INIT_PARAM::max_num_ep` max number of endpoints supported by the USB device controller instance (specified by
pad0	`uint8_tuint8_t USBD_API_INIT_PARAM::pad0[3][3]`
USB_Reset_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Reset_Event` Event for USB interface reset. This event fires when the USB host requests that the device reset its interface. This event fires after the control endpoint has been automatically configured by the library. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will prevent the device from enumerating correctly or operate properly.
USB_Suspend_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Suspend_Event` Event for USB suspend. This event fires when the USB host suspends the device by halting its transmission of Start Of Frame pulses to the device. This is generally hooked in order to move the device over to a low power state until the host wakes up the device. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will cause other system issues.

Table 183. USBD_API_INIT_PARAM class structure

Member	Description
USB_Resume_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Resume_Event` Event for USB wake up or resume. This event fires when a the USB device interface is suspended and the host wakes up the device by supplying Start Of Frame pulses. This is generally hooked to pull the user application out of a low power state and back into normal operating mode. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will cause other system issues.
reserved_sbz	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::reserved_sbz` Reserved parameter should be set to zero.
USB_SOF_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_SOF_Event` Event for USB Start Of Frame detection, when enabled. This event fires at the start of each USB frame, once per millisecond in full-speed mode or once per 125 microseconds in high-speed mode, and is synchronized to the USB bus. This event is time-critical; it is run once per millisecond (full-speed mode) and thus long handlers will significantly degrade device performance. This event should only be enabled when needed to reduce device wake-ups. This event is not normally active - it must be manually enabled and disabled via the USB interrupt register. **Remark:** This event is not normally active - it must be manually enabled and disabled via the USB interrupt register.
USB_WakeUpCfg	`USB_PARAM_CB_TUSB_PARAM_CB_T USBD_API_INIT_PARAM::USB_WakeUpCfg` Event for remote wake-up configuration, when enabled. This event fires when the USB host request the device to configure itself for remote wake-up capability. The USB host sends this request to device which report remote wake-up capable in their device descriptors, before going to low-power state. The application layer should implement this callback if they have any special on board circuit to triger remote wake up event. Also application can use this callback to differentiate the following SUSPEND event is caused by cable plug-out or host SUSPEND request. The device can wake-up host only after receiving this callback and remote wake-up feature is enabled by host. To signal remote wake-up the device has to generate resume signaling on bus by calling usapi.hw->WakeUp() routine. Parameters: 1. hUsb = Handle to the USB device stack. 2. param1 = When 0 - Clear the wake-up configuration, 1 - Enable the wake-up configuration. Returns: The call back should return ErrorCode_t type to indicate success or error condition.
USB_Power_Event	`USB_PARAM_CB_TUSB_PARAM_CB_T USBD_API_INIT_PARAM::USB_Power_Event` Reserved parameter should be set to zero.
USB_Error_Event	`USB_PARAM_CB_TUSB_PARAM_CB_T USBD_API_INIT_PARAM:USB_Error_Event` Event for error condition. This event fires when USB device controller detect an error condition in the system. Parameters: 1. hUsb = Handle to the USB device stack. 2. param1 = USB device interrupt status register. Returns: The call back should return ErrorCode_t type to indicate success or error condition.

Table 183. USBD_API_INIT_PARAM class structure

Member	Description
USB_Configure_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Configure_Event` Event for USB configuration number changed. This event fires when a the USB host changes the selected configuration number. On receiving configuration change request from host, the stack enables/configures the endpoints needed by the new configuration before calling this callback function. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will prevent the device from enumerating correctly or operate properly.
USB_Interface_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Interface_Event` Event for USB interface setting changed. This event fires when a the USB host changes the interface setting to one of alternate interface settings. On receiving interface change request from host, the stack enables/configures the endpoints needed by the new alternate interface setting before calling this callback function. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will prevent the device from enumerating correctly or operate properly.
USB_Feature_Event	`USB_CB_TUSB_CB_T USBD_API_INIT_PARAM::USB_Feature_Event` Event for USB feature changed. This event fires when a the USB host send set/clear feature request. The stack handles this request for USB_FEATURE_REMOTE_WAKEUP, USB_FEATURE_TEST_MODE and USB_FEATURE_ENDPOINT_STALL features only. On receiving feature request from host, the stack handle the request appropriately and then calls this callback function. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will prevent the device from enumerating correctly or operate properly.
virt_to_phys	`uint32_t(*uint32_t(* USBD_API_INIT_PARAM::virt_to_phys)(void *vaddr))(void *vaddr)` Reserved parameter for future use. should be set to zero.
cache_flush	`void(*void(* USBD_API_INIT_PARAM::cache_flush)(uint32_t *start_adr, uint32_t *end_adr))(uint32_t *start_adr, uint32_t *end_adr)` Reserved parameter for future use. should be set to zero.

10.5.27 USBD_CDC_API

CDC class API functions structure.This module exposes functions which interact directly with USB device controller hardware.

Table 184. USBD_CDC_API class structure

Member	Description
GetMemSize	`uint32_t(*uint32_t USBD_CDC_API::GetMemSize)(USBD_CDC_INIT_PARAM_T *param)` Function to determine the memory required by the CDC function driver module. This function is called by application layer before calling pUsbApi->CDC->Init(), to allocate memory used by CDC function driver module. The application should allocate the memory which is accessible by USB controller/DMA controller. **Remark:** Some memory areas are not accessible by all bus masters. Parameters: 1. param = Structure containing CDC function driver module initialization parameters. Returns: Returns the required memory size in bytes.

Table 184. USBD_CDC_API class structure

Member	Description
init	`ErrorCode_t(*ErrorCode_t USBD_CDC_API::init)(USBD_HANDLE_T hUsb, USBD_CDC_INIT_PARAM_T *param, USBD_HANDLE_T *phCDC)` Function to initialize CDC function driver module. This function is called by application layer to initialize CDC function driver module. hUsbHandle to the USB device stack. paramStructure containing CDC function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing CDC function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either CDC_Write() or CDC_Read() or CDC_Verify() callbacks are not defined. 4. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed. 5. ERR_USBD_BAD_EP_DESC = Wrong endpoint descriptor is passed.
SendNotification	`ErrorCode_t(*ErrorCode_t USBD_CDC_API::SendNotification)(USBD_HANDLE_T hCdc, uint8_t bNotification, uint16_t data)` Function to initialize CDC function driver module. This function is called by application layer to initialize CDC function driver module. hUsbHandle to the USB device stack. paramStructure containing CDC function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing CDC function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either CDC_Write() or CDC_Read() or CDC_Verify() callbacks are not defined. 4. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed. 5. ERR_USBD_BAD_EP_DESC = Wrong endpoint descriptor is passed.

10.5.28 USBD_CDC_INIT_PARAM

Communication Device Class function driver initialization parameter data structure.

Table 185. USBD_CDC_INIT_PARAM class structure

Member	Description
mem_base	`uint32_tuint32_t USBD_CDC_INIT_PARAM::mem_base` Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	`uint32_tuint32_t USBD_CDC_INIT_PARAM::mem_size` The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_CDC_API::GetMemSize() routine.
cif_intf_desc	`uint8_t *uint8_t* USBD_CDC_INIT_PARAM::cif_intf_desc` Pointer to the control interface descriptor within the descriptor array (
dif_intf_desc	`uint8_t *uint8_t* USBD_CDC_INIT_PARAM::dif_intf_desc` Pointer to the data interface descriptor within the descriptor array (
CIC_GetRequest	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CIC_GetRequest)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length)` Communication Interface Class specific get request callback function. This function is provided by the application software. This function gets called when host sends CIC management element get requests. The setup packet data (hCdcHandle to CDC function driver. pSetupPointer to setup packet received from host. pBufferPointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthAmount of data to be sent back to host. Parameters: 1. hCdc = Handle to CDC function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data to be sent back to host. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 185. USBD_CDC_INIT_PARAM class structure

Member	Description
CIC_SetRequest	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CIC_SetRequest)(USBD_HANDLE_T hCdc, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length))(USBD_HANDLE_T hCdc, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length)` Communication Interface Class specific set request callback function. This function is provided by the application software. This function gets called when host sends a CIC management element requests. The setup packet data (hCdcHandle to CDC function driver. pSetupPointer to setup packet received from host. pBufferPointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthAmount of data copied to destination buffer. Parameters: 1. hCdc = Handle to CDC function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
CDC_BulkIN_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_BulkIN_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Communication Device Class specific BULK IN endpoint handler. The application software should provide the BULK IN endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 185. USBD_CDC_INIT_PARAM class structure

Member	Description
CDC_BulkOUT_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_BulkOUT_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Communication Device Class specific BULK OUT endpoint handler. The application software should provide the BULK OUT endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
SendEncpsCmd	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::SendEncpsCmd)(USBD_HANDLE_T hCDC, uint8_t *buffer, uint16_t len))(USBD_HANDLE_T hCDC, uint8_t *buffer, uint16_t len)`
GetEncpsResp	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::GetEncpsResp)(USBD_HANDLE_T hCDC, uint8_t **buffer, uint16_t *len))(USBD_HANDLE_T hCDC, uint8_t **buffer, uint16_t *len)`
SetCommFeature	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::SetCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t *buffer, uint16_t len))(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t *buffer, uint16_t len)`
GetCommFeature	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::GetCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t **pBuffer, uint16_t *len))(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t **pBuffer, uint16_t *len)`
ClrCommFeature	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::ClrCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature))(USBD_HANDLE_T hCDC, uint16_t feature)`
SetCtrlLineState	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::SetCtrlLineState)(USBD_HANDLE_T hCDC, uint16_t state))(USBD_HANDLE_T hCDC, uint16_t state)`
SendBreak	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::SendBreak)(USBD_HANDLE_T hCDC, uint16_t mstime))(USBD_HANDLE_T hCDC, uint16_t mstime)`

Table 185. USBD_CDC_INIT_PARAM class structure

Member	Description
SetLineCode	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::SetLineCode)(USBD_HANDLE_T hCDC, CDC_LINE_CODING *line_coding))(USBD_HANDLE_T hCDC, CDC_LINE_CODING *line_coding)`
CDC_InterruptEP_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_InterruptEP_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional Communication Device Class specific INTERRUPT IN endpoint handler. The application software should provide the INT IN endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
CDC_Ep0_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_Ep0_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional user overridable function to replace the default CDC class handler. The application software could override the default EP0 class handler with their own by providing the handler function address as this data member of the parameter structure. Application which like the default handler should set this data member to zero before calling the USBD_CDC_API::Init(). **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

10.5.29 USBD_CORE_API

USBD stack Core API functions structure.

Table 186. USBD_CORE_API class structure

Member	Description
RegisterClassHandler	`ErrorCode_t(*ErrorCode_t USBD_CORE_API::RegisterClassHandler)(USBD_HANDLE_T hUsb, USB_EP_HANDLER_T pfn, void *data)` Function to register class specific EP0 event handler with USB device stack. The application layer uses this function when it has to register the custom class's EP0 handler. The stack calls all the registered class handlers on any EP0 event before going through default handling of the event. This gives the class handlers to implement class specific request handlers and also to override the default stack handling for a particular event targeted to the interface. Check USB_EP_HANDLER_T for more details on how the callback function should be implemented. Also application layer could use this function to register EP0 handler which responds to vendor specific requests. hUsbHandle to the USB device stack. pfnClass specific EP0 handler function. dataPointer to the data which will be passed when callback function is called by the stack. Parameters: 1. hUsb = Handle to the USB device stack. 2. pfn = Class specific EP0 handler function. 3. data = Pointer to the data which will be passed when callback function is called by the stack. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_TOO_MANY_CLASS_HDLR(0x0004000c) = The number of class handlers registered is greater than the number of handlers allowed by the stack.
RegisterEpHandler	`ErrorCode_t(*ErrorCode_t USBD_CORE_API::RegisterEpHandler)(USBD_HANDLE_T hUsb, uint32_t ep_index, USB_EP_HANDLER_T pfn, void *data)` Function to register interrupt/event handler for the requested endpoint with USB device stack. The application layer uses this function to register the custom class's EP0 handler. The stack calls all the registered class handlers on any EP0 event before going through default handling of the event. This gives the class handlers to implement class specific request handlers and also to override the default stack handling for a particular event targeted to the interface. Check USB_EP_HANDLER_T for more details on how the callback function should be implemented. hUsbHandle to the USB device stack. ep_indexClass specific EP0 handler function. pfnClass specific EP0 handler function. dataPointer to the data which will be passed when callback function is called by the stack. Parameters: 1. hUsb = Handle to the USB device stack. 2. ep_index = Class specific EP0 handler function. 3. pfn = Class specific EP0 handler function. 4. data = Pointer to the data which will be passed when callback function is called by the stack. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_TOO_MANY_CLASS_HDLR(0x0004000c) = Too many endpoint handlers.

Table 186. USBD_CORE_API class structure

Member	Description
SetupStage	`void(*void USBD_CORE_API::SetupStage)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in setup state. This function is called by USB stack and the application layer to set the EP0 state machine in setup state. This function will read the setup packet received from USB host into stack's buffer. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
DataInStage	`void(*void USBD_CORE_API::DataInStage)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in data_in state. This function is called by USB stack and the application layer to set the EP0 state machine in data_in state. This function will write the data present in EP0Data buffer to EP0 FIFO for transmission to host. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
DataOutStage	`void(*void USBD_CORE_API::DataOutStage)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in data_out state. This function is called by USB stack and the application layer to set the EP0 state machine in data_out state. This function will read the control data (EP0 out packets) received from USB host into EP0Data buffer. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.

Table 186. USBD_CORE_API class structure

Member	Description
StatusInStage	`void(*void USBD_CORE_API::StatusInStage)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in status_in state. This function is called by USB stack and the application layer to set the EP0 state machine in status_in state. This function will send zero length IN packet on EP0 to host, indicating positive status. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
StatusOutStage	`void(*void USBD_CORE_API::StatusOutStage)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in status_out state. This function is called by USB stack and the application layer to set the EP0 state machine in status_out state. This function will read the zero length OUT packet received from USB host on EP0. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
StallEp0	`void(*void USBD_CORE_API::StallEp0)(USBD_HANDLE_T hUsb)` Function to set EP0 state machine in stall state. This function is called by USB stack and the application layer to generate STALL signalling on EP0 endpoint. This function will also reset the EP0Data buffer. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.

10.5.30 USBD_DFU_API

DFU class API functions structure.This module exposes functions which interact directly with USB device controller hardware.

Table 187. USBD_DFU_API class structure

Member	Description
GetMemSize	`uint32_t(*uint32_t USBD_DFU_API::GetMemSize)(USBD_DFU_INIT_PARAM_T *param)` Function to determine the memory required by the DFU function driver module. This function is called by application layer before calling pUsbApi->dfu->Init(), to allocate memory used by DFU function driver module. The application should allocate the memory which is accessible by USB controller/DMA controller. **Remark:** Some memory areas are not accessible by all bus masters. Parameters: 1. param = Structure containing DFU function driver module initialization parameters. Returns: Returns the required memory size in bytes.
init	`ErrorCode_t(*ErrorCode_t USBD_DFU_API::init)(USBD_HANDLE_T hUsb, USBD_DFU_INIT_PARAM_T *param, uint32_t init_state)` Function to initialize DFU function driver module. This function is called by application layer to initialize DFU function driver module. hUsbHandle to the USB device stack. paramStructure containing DFU function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing DFU function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either DFU_Write() or DFU_Done() or DFU_Read() callbacks are not defined. 4. ERR_USBD_BAD_DESC = USB_DFU_DESCRIPTOR_TYPE is not defined immediately after interface descriptor.wTransferSize in descriptor doesn't match the value passed in param->wTransferSize.DFU_Detach() is not defined while USB_DFU_WILL_DETACH is set in DFU descriptor. 5. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed.

10.5.31 USBD_DFU_INIT_PARAM

USB descriptors data structure.

Table 188. USBD_DFU_INIT_PARAM class structure

Member	Description
mem_base	`uint32_tuint32_t USBD_DFU_INIT_PARAM::mem_base` Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	`uint32_tuint32_t USBD_DFU_INIT_PARAM::mem_size` The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_DFU_API::GetMemSize() routine.
wTransferSize	`uint16_tuint16_t USBD_DFU_INIT_PARAM::wTransferSize` DFU transfer block size in number of bytes. This value should match the value set in DFU descriptor provided as part of the descriptor array (
pad	`uint16_tuint16_t USBD_DFU_INIT_PARAM::pad`
intf_desc	`uint8_t *uint8_t* USBD_DFU_INIT_PARAM::intf_desc` Pointer to the DFU interface descriptor within the descriptor array (
DFU_Write	`uint8_t(*uint8_t(* USBD_DFU_INIT_PARAM::DFU_Write)(uint32_t block_num, uint8_t **src, uint32_t length, uint8_t *bwPollTimeout))(uint32_t block_num, uint8_t **src, uint32_t length, uint8_t *bwPollTimeout)` DFU Write callback function. This function is provided by the application software. This function gets called when host sends a write command. For application using zero-copy buffer scheme this function is called for the first time with block_numDestination start address. srcPointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. bwPollTimeoutPointer to a 3 byte buffer which the callback implementer should fill with the amount of minimum time, in milliseconds, that the host should wait before sending a subsequent DFU_GETSTATUS request. lengthNumber of bytes to be written. Parameters: 1. block_num = Destination start address. 2. src = Pointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. bwPollTimeout = Pointer to a 3 byte buffer which the callback implementer should fill with the amount of minimum time, in milliseconds, that the host should wait before sending a subsequent DFU_GETSTATUS request. 4. length = Number of bytes to be written. Returns: Returns DFU_STATUS_ values defined in mw_usbd_dfu.h.

Table 188. USBD_DFU_INIT_PARAM class structure

Member	Description
DFU_Read	`uint32_t(*uint32_t(* USBD_DFU_INIT_PARAM::DFU_Read)(uint32_t block_num, uint8_t **dst, uint32_t length))(uint32_t block_num, uint8_t **dst, uint32_t length)` DFU Read callback function. This function is provided by the application software. This function gets called when host sends a read command. block_numDestination start address. dstPointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthAmount of data copied to destination buffer. Parameters: 1. block_num = Destination start address. 2. dst = Pointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. length = Amount of data copied to destination buffer. Returns: Returns DFU_STATUS_ values defined in mw_usbd_dfu.h.

Table 188. USBD_DFU_INIT_PARAM class structure

Member	Description
DFU_Done	`void(*void(* USBD_DFU_INIT_PARAM::DFU_Done)(void))(void)` DFU done callback function. This function is provided by the application software. This function gets called after download is finished. Nothing. Returns: Nothing.
DFU_Detach	`void(*void(* USBD_DFU_INIT_PARAM::DFU_Detach)(USBD_HANDLE_T hUsb))(USBD_HANDLE_T hUsb)` DFU detach callback function. This function is provided by the application software. This function gets called after USB_REQ_DFU_DETACH is received. Applications which set USB_DFU_WILL_DETACH bit in DFU descriptor should define this function. As part of this function application can call Connect() routine to disconnect and then connect back with host. For application which rely on WinUSB based host application should use this feature since USB reset can be invoked only by kernel drivers on Windows host. By implementing this feature host doesn't have to issue reset instead the device has to do it automatically by disconnect and connect procedure. hUsbHandle DFU control structure. Parameters: 1. hUsb = Handle DFU control structure. Returns: Nothing.
DFU_Ep0_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_DFU_INIT_PARAM::DFU_Ep0_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional user overridable function to replace the default DFU class handler. The application software could override the default EP0 class handler with their own by providing the handler function address as this data member of the parameter structure. Application which like the default handler should set this data member to zero before calling the USBD_DFU_API::Init(). **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

10.5.32 USBD_HID_API

HID class API functions structure.This structure contains pointers to all the function exposed by HID function driver module.

Table 189. USBD_HID_API class structure

Member	Description
GetMemSize	`uint32_t(*uint32_t USBD_HID_API::GetMemSize)(USBD_HID_INIT_PARAM_T *param)` Function to determine the memory required by the HID function driver module. This function is called by application layer before calling pUsbApi->hid->Init(), to allocate memory used by HID function driver module. The application should allocate the memory which is accessible by USB controller/DMA controller. **Remark:** Some memory areas are not accessible by all bus masters. Parameters: 1. param = Structure containing HID function driver module initialization parameters. Returns: Returns the required memory size in bytes.
init	`ErrorCode_t(*ErrorCode_t USBD_HID_API::init)(USBD_HANDLE_T hUsb, USBD_HID_INIT_PARAM_T *param)` Function to initialize HID function driver module. This function is called by application layer to initialize HID function driver module. On successful initialization the function returns a handle to HID function driver module in passed param structure. hUsbHandle to the USB device stack. paramStructure containing HID function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing HID function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either HID_GetReport() or HID_SetReport() callback are not defined. 4. ERR_USBD_BAD_DESC = HID_HID_DESCRIPTOR_TYPE is not defined immediately after interface descriptor. 5. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed. 6. ERR_USBD_BAD_EP_DESC = Wrong endpoint descriptor is passed.

10.5.33 USBD_HID_INIT_PARAM

USB descriptors data structure.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
mem_base	`uint32_tuint32_t USBD_HID_INIT_PARAM::mem_base` Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	`uint32_tuint32_t USBD_HID_INIT_PARAM::mem_size` The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_HID_API::GetMemSize() routine.
max_reports	`uint8_tuint8_t USBD_HID_INIT_PARAM::max_reports` Number of HID reports supported by this instance of HID class driver.
pad	`uint8_tuint8_t USBD_HID_INIT_PARAM::pad[3][3]`
intf_desc	`uint8_t *uint8_t* USBD_HID_INIT_PARAM::intf_desc` Pointer to the HID interface descriptor within the descriptor array (
report_data	`USB_HID_REPORT_T *USB_HID_REPORT_T* USBD_HID_INIT_PARAM::report_data` Pointer to an array of HID report descriptor data structure (**Remark:** This array should be of global scope.
HID_GetReport	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_GetReport)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length)` HID get report callback function. This function is provided by the application software. This function gets called when host sends a HID_REQUEST_GET_REPORT request. The setup packet data (**Remark:** HID reports are sent via interrupt IN endpoint also. This function is called only when report request is received on control endpoint. Application should implement HID_EpIn_Hdlr to send reports to host via interrupt IN endpoint. Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing report data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
HID_SetReport	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetReport)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length)` HID set report callback function. This function is provided by the application software. This function gets called when host sends a HID_REQUEST_SET_REPORT request. The setup packet data (hHidHandle to HID function driver. pSetupPointer to setup packet received from host. pBufferPointer to a pointer of data buffer containing report data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthAmount of data copied to destination buffer. Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing report data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
HID_GetPhysDesc	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_GetPhysDesc)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuf, uint16_t *length))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuf, uint16_t *length)` Optional callback function to handle HID_GetPhysDesc request. The application software could provide this callback HID_GetPhysDesc handler to handle get physical descriptor requests sent by the host. When host requests Physical Descriptor set 0, application should return a special descriptor identifying the number of descriptor sets and their sizes. A Get_Descriptor request with the Physical Index equal to 1 should return the first Physical Descriptor set. A device could possibly have alternate uses for its items. These can be enumerated by issuing subsequent Get_Descriptor requests while incrementing the Descriptor Index. A device should return the last descriptor set to requests with an index greater than the last number defined in the HID descriptor. **Remark:** Applications which don't have physical descriptor should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuf = Pointer to a pointer of data buffer containing physical descriptor data. If the physical descriptor is in USB accessible memory area application could just update the pointer or else it should copy the descriptor to the address pointed by this pointer. 4. length = Amount of data copied to destination buffer or descriptor length. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
HID_SetIdle	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetIdle)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t idleTime))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t idleTime)` Optional callback function to handle HID_REQUEST_SET_IDLE request. The application software could provide this callback to handle HID_REQUEST_SET_IDLE requests sent by the host. This callback is provided to applications to adjust timers associated with various reports, which are sent to host over interrupt endpoint. The setup packet data (**Remark:** Applications which don't send reports on Interrupt endpoint or don't have idle time between reports should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet recived from host. 3. idleTime = Idle time to be set for the specified report. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
HID_SetProtocol	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetProtocol)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t protocol))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t protocol)` Optional callback function to handle HID_REQUEST_SET_PROTOCOL request. The application software could provide this callback to handle HID_REQUEST_SET_PROTOCOL requests sent by the host. This callback is provided to applications to adjust modes of their code between boot mode and report mode. **Remark:** Applications which don't support protocol modes should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet recived from host. 3. protocol = Protocol mode. 0 = Boot Protocol 1 = Report Protocol Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
HID_EpIn_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_EpIn_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional Interrupt IN endpoint event handler. The application software could provide Interrupt IN endpoint event handler. Application which send reports to host on interrupt endpoint should provide an endpoint event handler through this data member. This data member is ignored if the interface descriptor hUsbHandle to the USB device stack. dataHandle to HID function driver. eventType of endpoint event. See USBD_EVENT_T for more details. Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Handle to HID function driver. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should return ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
HID_EpOut_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_EpOut_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional Interrupt OUT endpoint event handler. The application software could provide Interrupt OUT endpoint event handler. Application which receives reports from host on interrupt endpoint should provide an endpoint event handler through this data member. This data member is ignored if the interface descriptor hUsbHandle to the USB device stack. dataHandle to HID function driver. eventType of endpoint event. See USBD_EVENT_T for more details. Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Handle to HID function driver. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should return ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
HID_GetReportDesc	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_GetReportDesc)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuf, uint16_t *length))(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuf, uint16_t *length)` Optional user overridable function to replace the default HID_GetReportDesc handler. The application software could override the default HID_GetReportDesc handler with their own by providing the handler function address as this data member of the parameter structure. Application which like the default handler should set this data member to zero before calling the USBD_HID_API::Init() and also provide report data array **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 190. USBD_HID_INIT_PARAM class structure

Member	Description
HID_Ep0_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_HID_INIT_PARAM::HID_Ep0_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional user overridable function to replace the default HID class handler. The application software could override the default EP0 class handler with their own by providing the handler function address as this data member of the parameter structure. Application which like the default handler should set this data member to zero before calling the USBD_HID_API::Init(). **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

10.5.34 USBD_HW_API

Hardware API functions structure.This module exposes functions which interact directly with USB device controller hardware.

Table 191. USBD_HW_API class structure

Member	Description
GetMemSize	`uint32_t(*uint32_t USBD_HW_API::GetMemSize)(USBD_API_INIT_PARAM_T *param)` Function to determine the memory required by the USB device stack's DCD and core layers. This function is called by application layer before calling pUsbApi->hw-> **Remark:** Some memory areas are not accessible by all bus masters. Parameters: 1. param = Structure containing USB device stack initialization parameters. Returns: Returns the required memory size in bytes.
Init	`ErrorCode_t(*ErrorCode_t USBD_HW_API::Init)(USBD_HANDLE_T *phUsb, USB_CORE_DESCS_T *pDesc, USBD_API_INIT_PARAM_T *param)` Function to initialize USB device stack's DCD and core layers. This function is called by application layer to initialize USB hardware and core layers. On successful initialization the function returns a handle to USB device stack which should be passed to the rest of the functions. phUsbPointer to the USB device stack handle of type USBD_HANDLE_T. paramStructure containing USB device stack initialization parameters. Parameters: 1. phUsb = Pointer to the USB device stack handle of type USBD_HANDLE_T. 2. param = Structure containing USB device stack initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK(0) = On success 2. ERR_USBD_BAD_MEM_BUF(0x0004000b) = When insufficient memory buffer is passed or memory is not aligned on 2048 boundary.
Connect	`void(*void USBD_HW_API::Connect)(USBD_HANDLE_T hUsb, uint32_t con)` Function to make USB device visible/invisible on the USB bus. This function is called after the USB initialization. This function uses the soft connect feature to make the device visible on the USB bus. This function is called only after the application is ready to handle the USB data. The enumeration process is started by the host after the device detection. The driver handles the enumeration process according to the USB descriptors passed in the USB initialization function. hUsbHandle to the USB device stack. conStates whether to connect (1) or to disconnect (0). Parameters: 1. hUsb = Handle to the USB device stack. 2. con = States whether to connect (1) or to disconnect (0). Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
ISR	`void(*void USBD_HW_API::ISR)(USBD_HANDLE_T hUsb)` Function to USB device controller interrupt events. When the user application is active the interrupt handlers are mapped in the user flash space. The user application must provide an interrupt handler for the USB interrupt and call this function in the interrupt handler routine. The driver interrupt handler takes appropriate action according to the data received on the USB bus. hUsbHandle to the USB device stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
Reset	`void(*void USBD_HW_API::Reset)(USBD_HANDLE_T hUsb)` Function to Reset USB device stack and hardware controller. Reset USB device stack and hardware controller. Disables all endpoints except EP0. Clears all pending interrupts and resets endpoint transfer queues. This function is called internally by pUsbApi->hw->init() and from reset event. hUsbHandle to the USB device stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
ForceFullSpeed	`void(*void USBD_HW_API::ForceFullSpeed)(USBD_HANDLE_T hUsb, uint32_t cfg)` Function to force high speed USB device to operate in full speed mode. This function is useful for testing the behavior of current device when connected to a full speed only hosts. hUsbHandle to the USB device stack. cfgWhen 1 - set force full-speed or 0 - clear force full-speed. Parameters: 1. hUsb = Handle to the USB device stack. 2. cfg = When 1 - set force full-speed or 0 - clear force full-speed. Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
WakeUpCfg	`void(*void USBD_HW_API::WakeUpCfg)(USBD_HANDLE_T hUsb, uint32_t cfg)` Function to configure USB device controller to walk-up host on remote events. This function is called by application layer to configure the USB device controller to wake up on remote events. It is recommended to call this function from users's USB_WakeUpCfg() callback routine registered with stack. **Remark:** User's USB_WakeUpCfg() is registered with stack by setting the USB_WakeUpCfg member of USBD_API_INIT_PARAM_T structure before calling pUsbApi->hw->Init() routine. Certain USB device controllers needed to keep some clocks always on to generate resume signaling through pUsbApi->hw->WakeUp(). This hook is provided to support such controllers. In most controllers cases this is an empty routine. Parameters: 1. hUsb = Handle to the USB device stack. 2. cfg = When 1 - Configure controller to wake on remote events or 0 - Configure controller not to wake on remote events. Returns: Nothing.
SetAddress	`void(*void USBD_HW_API::SetAddress)(USBD_HANDLE_T hUsb, uint32_t adr)` Function to set USB address assigned by host in device controller hardware. This function is called automatically when USB_REQUEST_SET_ADDRESS request is received by the stack from USB host. This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly. Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. hUsbHandle to the USB device stack. adrUSB bus Address to which the device controller should respond. Usually assigned by the USB host. Parameters: 1. hUsb = Handle to the USB device stack. 2. adr = USB bus Address to which the device controller should respond. Usually assigned by the USB host. Returns: Nothing.
Configure	`void(*void USBD_HW_API::Configure)(USBD_HANDLE_T hUsb, uint32_t cfg)` Function to configure device controller hardware with selected configuration. This function is called automatically when USB_REQUEST_SET_CONFIGURATION request is received by the stack from USB host. This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly. Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. hUsbHandle to the USB device stack. cfgConfiguration index. Parameters: 1. hUsb = Handle to the USB device stack. 2. cfg = Configuration index. Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
ConfigEP	`void(*void USBD_HW_API::ConfigEP)(USBD_HANDLE_T hUsb, USB_ENDPOINT_DESCRIPTOR *pEPD)` Function to configure USB Endpoint according to descriptor. This function is called automatically when USB_REQUEST_SET_CONFIGURATION request is received by the stack from USB host. All the endpoints associated with the selected configuration are configured. This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly. Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. hUsbHandle to the USB device stack. pEPDEndpoint descriptor structure defined in USB 2.0 specification. Parameters: 1. hUsb = Handle to the USB device stack. 2. pEPD = Endpoint descriptor structure defined in USB 2.0 specification. Returns: Nothing.
DirCtrlEP	`void(*void USBD_HW_API::DirCtrlEP)(USBD_HANDLE_T hUsb, uint32_t dir)` Function to set direction for USB control endpoint EP0. This function is called automatically by the stack on need basis. This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly. Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. hUsbHandle to the USB device stack. cfgWhen 1 - Set EP0 in IN transfer mode 0 - Set EP0 in OUT transfer mode Parameters: 1. hUsb = Handle to the USB device stack. 2. cfg = When 1 - Set EP0 in IN transfer mode 0 - Set EP0 in OUT transfer mode Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
EnableEP	`void(*void USBD_HW_API::EnableEP)(USBD_HANDLE_T hUsb, uint32_t EPNum)` Function to enable selected USB endpoint. This function enables interrupts on selected endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing. This function enables interrupts on selected endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number corresponding to the event as per USB specification. ie. An EP1_IN is represented by 0x81 number. For device events set this param to 0x0. eventType of endpoint event. See USBD_EVENT_T for more details. enable1 - enable event, 0 - disable event. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number corresponding to the event as per USB specification. ie. An EP1_IN is represented by 0x81 number. For device events set this param to 0x0. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. 4. enable = 1 - enable event, 0 - disable event. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK(0) = - On success 2. ERR_USBD_INVALID_REQ(0x00040001) = - Invalid event type.
DisableEP	`void(*void USBD_HW_API::DisableEP)(USBD_HANDLE_T hUsb, uint32_t EPNum)` Function to disable selected USB endpoint. This function disables interrupts on selected endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
ResetEP	`void(*void USBD_HW_API::ResetEP)(USBD_HANDLE_T hUsb, uint32_t EPNum)` Function to reset selected USB endpoint. This function flushes the endpoint buffers and resets data toggle logic. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.
SetStallEP	`void(*void USBD_HW_API::SetStallEP)(USBD_HANDLE_T hUsb, uint32_t EPNum)` Function to STALL selected USB endpoint. Generates STALL signalling for requested endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.
ClrStallEP	`void(*void USBD_HW_API::ClrStallEP)(USBD_HANDLE_T hUsb, uint32_t EPNum)` Function to clear STALL state for the requested endpoint. This function clears STALL state for the requested endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.

Table 191. USBD_HW_API class structure

Member	Description
SetTestMode	`ErrorCode_t(*ErrorCode_t USBD_HW_API::SetTestMode)(USBD_HANDLE_T hUsb, uint8_t mode)` Function to set high speed USB device controller in requested test mode. USB-IF requires the high speed device to be put in various test modes for electrical testing. This USB device stack calls this function whenever it receives USB_REQUEST_CLEAR_FEATURE request for USB_FEATURE_TEST_MODE. Users can put the device in test mode by directly calling this function. Returns ERR_USBD_INVALID_REQ when device controller is full-speed only. hUsbHandle to the USB device stack. modeTest mode defined in USB 2.0 electrical testing specification. Parameters: 1. hUsb = Handle to the USB device stack. 2. mode = Test mode defined in USB 2.0 electrical testing specification. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK(0) = - On success 2. ERR_USBD_INVALID_REQ(0x00040001) = - Invalid test mode or Device controller is full-speed only.
ReadEP	`uint32_t(*uint32_t USBD_HW_API::ReadEP)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint8_t *pData)` Function to read data received on the requested endpoint. This function is called by USB stack and the application layer to read the data received on the requested endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. pDataPointer to the data buffer where data is to be copied. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. 3. pData = Pointer to the data buffer where data is to be copied. Returns: Returns the number of bytes copied to the buffer.

Table 191. USBD_HW_API class structure

Member	Description
ReadReqEP	`uint32_t(*uint32_t USBD_HW_API::ReadReqEP)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint8_t *pData, uint32_t len)` Function to queue read request on the specified endpoint. This function is called by USB stack and the application layer to queue a read request on the specified endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. pDataPointer to the data buffer where data is to be copied. This buffer address should be accessible by USB DMA master. lenLength of the buffer passed. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. 3. pData = Pointer to the data buffer where data is to be copied. This buffer address should be accessible by USB DMA master. 4. len = Length of the buffer passed. Returns: Returns the length of the requested buffer.
ReadSetupPkt	`uint32_t(*uint32_t USBD_HW_API::ReadSetupPkt)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint32_t *pData)` Function to read setup packet data received on the requested endpoint. This function is called by USB stack and the application layer to read setup packet data received on the requested endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP0_IN is represented by 0x80 number. pDataPointer to the data buffer where data is to be copied. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP0_IN is represented by 0x80 number. 3. pData = Pointer to the data buffer where data is to be copied. Returns: Returns the number of bytes copied to the buffer.

Table 191. USBD_HW_API class structure

Member	Description
WriteEP	`uint32_t(*uint32_t USBD_HW_API::WriteEP)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint8_t *pData, uint32_t cnt)` Function to write data to be sent on the requested endpoint. This function is called by USB stack and the application layer to send data on the requested endpoint. hUsbHandle to the USB device stack. EPNumEndpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. pDataPointer to the data buffer from where data is to be copied. cntNumber of bytes to write. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. 3. pData = Pointer to the data buffer from where data is to be copied. 4. cnt = Number of bytes to write. Returns: Returns the number of bytes written.
WakeUp	`void(*void USBD_HW_API::WakeUp)(USBD_HANDLE_T hUsb)` Function to generate resume signaling on bus for remote host wake-up. This function is called by application layer to remotely wake up host controller when system is in suspend state. Application should indicate this remote wake up capability by setting USB_CONFIG_REMOTE_WAKEUP in bmAttributes of Configuration Descriptor. Also this routine will generate resume signalling only if host enables USB_FEATURE_REMOTE_WAKEUP by sending SET_FEATURE request before suspending the bus. hUsbHandle to the USB device stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
EnableEvent	`ErrorCode_t(*ErrorCode_t(* USBD_HW_API::EnableEvent)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint32_t event_type, uint32_t enable))(USBD_HANDLE_T hUsb, uint32_t EPNum, uint32_t event_type, uint32_t enable)`

10.5.35 USBD_MSC_API

MSC class API functions structure.This module exposes functions which interact directly with USB device controller hardware.

Table 192. USBD_MSC_API class structure

Member	Description
GetMemSize	`uint32_t(*uint32_t USBD_MSC_API::GetMemSize)(USBD_MSC_INIT_PARAM_T *param)` Function to determine the memory required by the MSC function driver module. This function is called by application layer before calling pUsbApi->msc->Init(), to allocate memory used by MSC function driver module. The application should allocate the memory which is accessible by USB controller/DMA controller. **Remark:** Some memory areas are not accessible by all bus masters. Parameters: 1. param = Structure containing MSC function driver module initialization parameters. Returns: Returns the required memory size in bytes.
init	`ErrorCode_t(*ErrorCode_t USBD_MSC_API::init)(USBD_HANDLE_T hUsb, USBD_MSC_INIT_PARAM_T *param)` Function to initialize MSC function driver module. This function is called by application layer to initialize MSC function driver module. hUsbHandle to the USB device stack. paramStructure containing MSC function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing MSC function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either MSC_Write() or MSC_Read() or MSC_Verify() callbacks are not defined. 4. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed. 5. ERR_USBD_BAD_EP_DESC = Wrong endpoint descriptor is passed.

10.5.36 USBD_MSC_INIT_PARAM

Mass Storage class function driver initialization parameter data structure.

Table 193. USBD_MSC_INIT_PARAM class structure

Member	Description
mem_base	`uint32_tuint32_t USBD_MSC_INIT_PARAM::mem_base` Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	`uint32_tuint32_t USBD_MSC_INIT_PARAM::mem_size` The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_MSC_API::GetMemSize() routine.
InquiryStr	`uint8_t *uint8_t* USBD_MSC_INIT_PARAM::InquiryStr` Pointer to the 28 character string. This string is sent in response to the SCSI Inquiry command. **Remark:** The data pointed by the pointer should be of global scope.
BlockCount	`uint32_tuint32_t USBD_MSC_INIT_PARAM::BlockCount` Number of blocks present in the mass storage device
BlockSize	`uint32_tuint32_t USBD_MSC_INIT_PARAM::BlockSize` Block size in number of bytes
MemorySize	`uint32_tuint32_t USBD_MSC_INIT_PARAM::MemorySize` Memory size in number of bytes
intf_desc	`uint8_t *uint8_t* USBD_MSC_INIT_PARAM::intf_desc` Pointer to the interface descriptor within the descriptor array (
MSC_Write	`void(*void(* USBD_MSC_INIT_PARAM::MSC_Write)(uint32_t offset, uint8_t **src, uint32_t length))(uint32_t offset, uint8_t **src, uint32_t length)` MSC Write callback function. This function is provided by the application software. This function gets called when host sends a write command. offsetDestination start address. srcPointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthNumber of bytes to be written. Parameters: 1. offset = Destination start address. 2. src = Pointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. length = Number of bytes to be written. Returns: Nothing.

Table 193. USBD_MSC_INIT_PARAM class structure

Member	Description
MSC_Read	`void(*void(* USBD_MSC_INIT_PARAM::MSC_Read)(uint32_t offset, uint8_t **dst, uint32_t length))(uint32_t offset, uint8_t **dst, uint32_t length)` MSC Read callback function. This function is provided by the application software. This function gets called when host sends a read command. offsetSource start address. dstPointer to a pointer to the source of data. The MSC function drivers implemented in stack are written with zero-copy model. Meaning the stack doesn't make an extra copy of buffer before writing/reading data from USB hardware FIFO. Hence the parameter is pointer to a pointer containing address buffer (uint8_t** dst). So that the user application can update the buffer pointer instead of copying data to address pointed by the parameter. /note The updated buffer address should be accessible by USB DMA master. If user doesn't want to use zero-copy model, then the user should copy data to the address pointed by the passed buffer pointer parameter and shouldn't change the address value. See Zero-Copy Data Transfer model for more details on zero-copy concept. lengthNumber of bytes to be read. Parameters: 1. offset = Source start address. 2. dst = Pointer to a pointer to the source of data. The MSC function drivers implemented in stack are written with zero-copy model. Meaning the stack doesn't make an extra copy of buffer before writing/reading data from USB hardware FIFO. Hence the parameter is pointer to a pointer containing address buffer (uint8_t** dst). So that the user application can update the buffer pointer instead of copying data to address pointed by the parameter. /note The updated buffer address should be access able by USB DMA master. If user doesn't want to use zero-copy model, then the user should copy data to the address pointed by the passed buffer pointer parameter and shouldn't change the address value. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. length = Number of bytes to be read. Returns: Nothing.

Table 193. USBD_MSC_INIT_PARAM class structure

Member	Description
MSC_Verify	`ErrorCode_t(*ErrorCode_t(* USBD_MSC_INIT_PARAM::MSC_Verify)(uint32_t offset, uint8_t buf[], uint32_t length))(uint32_t offset, uint8_t buf[], uint32_t length)` MSC Verify callback function. This function is provided by the application software. This function gets called when host sends a verify command. The callback function should compare the buffer with the destination memory at the requested offset and offsetDestination start address. bufBuffer containing the data sent by the host. lengthNumber of bytes to verify. Parameters: 1. offset = Destination start address. 2. buf = Buffer containing the data sent by the host. 3. length = Number of bytes to verify. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = If data in the buffer matches the data at destination 2. ERR_FAILED = At least one byte is different.
MSC_GetWriteBuf	`void(*void(* USBD_MSC_INIT_PARAM::MSC_GetWriteBuf)(uint32_t offset, uint8_t **buff_adr, uint32_t length))(uint32_t offset, uint8_t **buff_adr, uint32_t length)` Optional callback function to optimize MSC_Write buffer transfer. This function is provided by the application software. This function gets called when host sends SCSI_WRITE10/SCSI_WRITE12 command. The callback function should update the offsetDestination start address. bufBuffer containing the data sent by the host. lengthNumber of bytes to write. Parameters: 1. offset = Destination start address. 2. buf = Buffer containing the data sent by the host. 3. length = Number of bytes to write. Returns: Nothing.

Table 193. USBD_MSC_INIT_PARAM class structure

Member	Description
MSC_Ep0_Hdlr	`ErrorCode_t(*ErrorCode_t(* USBD_MSC_INIT_PARAM::MSC_Ep0_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event)` Optional user overridable function to replace the default MSC class handler. The application software could override the default EP0 class handler with their own by providing the handler function address as this data member of the parameter structure. Application which like the default handler should set this data member to zero before calling the USBD_MSC_API::Init(). **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

11.1 How to read this chapter

The USB block is available on all LPC11Uxx parts.

11.2 Basic configuration

- Pins: Configure the USB pins in the IOCON register block.
- In the SYSAHBCLKCTRL register, enable the clock to the USB controller register interface by setting bit 14 and to the USB RAM by setting bit 27 (see Table 23).
- Power: Enable the power to the USB PHY and to the USB PLL, if used, in the PDRUNCFG register (Table 46).
- Configure the USB main clock (see Table 28).
- Configure the USB wake-up signal (see Section 11.7.6) if needed.

11.3 Features

- USB2.0 full-speed device controller.
- Supports 10 physical (5 logical) endpoints including one control endpoint.
- Single and double-buffering supported.
- Each non-control endpoint supports bulk, interrupt, or isochronous endpoint types.
- Supports wake-up from Deep-sleep mode on USB activity and remote wake-up.
- Supports SoftConnect.
- Link Power Management (LPM) supported.

11.4 General description

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. The LPC11Uxx device controller supports up to 10 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 latency 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 LPC11Uxx enables full-speed (12 Mb/s) data exchange with a USB host controller.

Figure 18 shows the block diagram of the USB device controller.

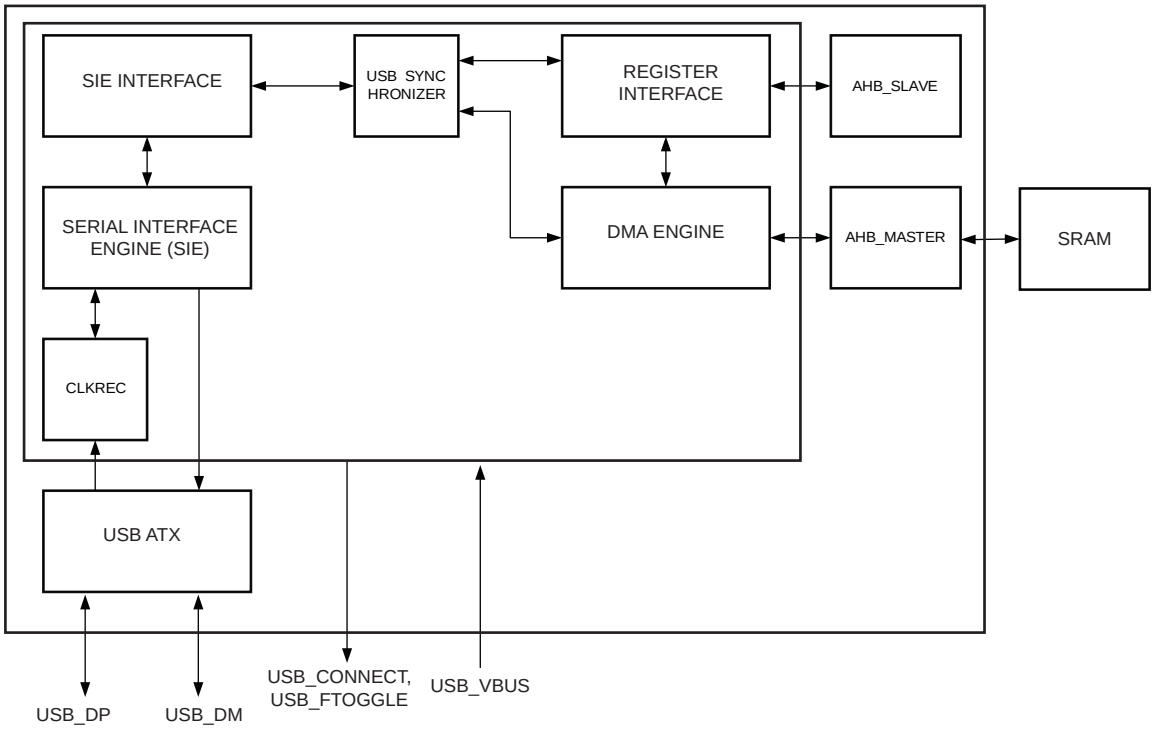


Fig 18. USB block diagram

The USB Device Controller has a built-in analog transceiver (ATX). The USB ATX sends/receives the bi-directional USB_DP and USB_DM signals of the USB bus.

The SIE implements the full USB protocol layer. It is completely hardwired for speed and needs no software intervention. It handles transfer of data between the endpoint buffers in USB 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.

11.4.1 USB software interface

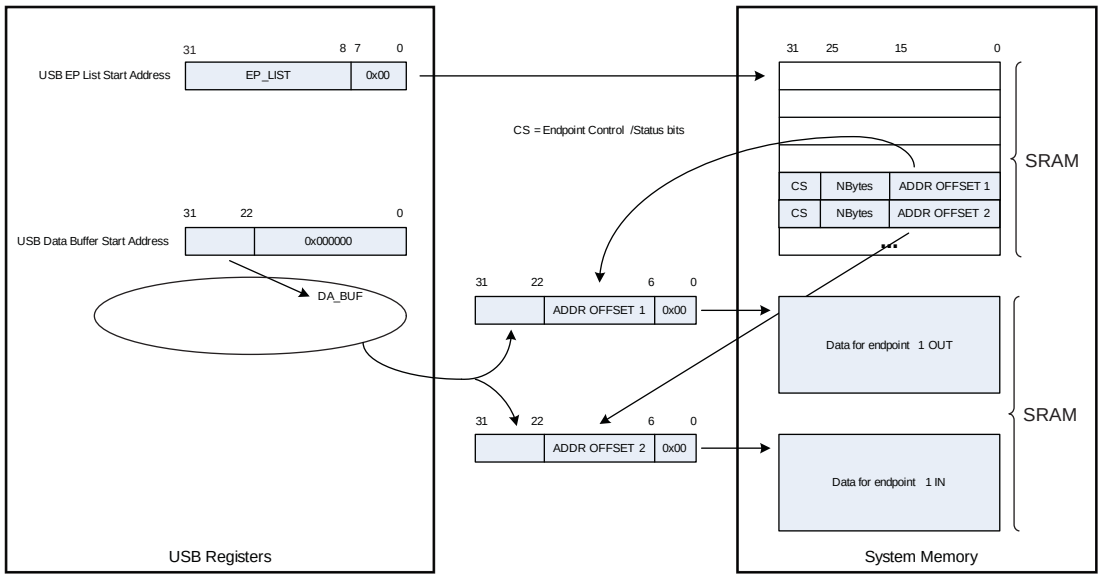


Fig 19. USB software interface

11.4.2 Fixed endpoint configuration

Table 194 shows the supported endpoint configurations. The packet size is configurable up to the maximum value shown in Table 194 for each type of end point.

Table 194. Fixed endpoint configuration

Logical endpoint	Physical endpoint	Endpoint type	Direction	Max packet size (byte)	Double buffer
0	0	Control	Out	64	No
0	1	Control	In	64	No
1	2	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
1	3	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
2	4	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
2	5	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
3	6	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
3	7	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
4	8	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
4	9	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes

11.4.3 SoftConnect

The connection to the USB is accomplished by bringing USB_DP (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 USB_DP and V_{DD} (+3.3 V). Software can then control the CONNECT signal by writing to the DCON bit in the DEVCMDSTAT register.

11.4.4 Interrupts

The USB controller has two interrupt lines USB_Int_Req_IRQ and USB_Int_Req_FIQ. Software can program the corresponding bit in the USB interrupt routing register to route the interrupt condition to one of these entries in the NVIC table Table 58. An interrupt is generated by the hardware if both the interrupt status bit and the corresponding interrupt enable bit are set. The interrupt status bit is set by hardware if the interrupt condition occurs (irrespective of the interrupt enable bit setting).

11.4.5 Suspend and resume

The USB protocol insists on power management by the USB device. This becomes even more important if the device draws power from the bus (bus-powered device). The following constraints should be met by the bus-powered device.

- A device in the non-configured state should draw a maximum of 100mA from the USB bus.
- 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.
- A suspended device should draw a maximum of 500 μA.

A device will go into the L2 suspend state if there is no activity on the USB bus for more than 3 ms. A suspended device wakes up, if there is transmission from the host (host-initiated wake up). The USB controller on the LPC11Uxx also supports software initiated remote wake-up. To initiate remote wake-up, software on the device must enable all clocks and clear the suspend bit. This will cause the hardware to generate a remote wake-up signal upstream.

The USB controller on the LPC11Uxx supports Link Power Management (LPM). Link Power Management defines an additional link power management state L1 that supplements the existing L2 state by utilizing most of the existing suspend/resume infrastructure but provides much faster transitional latencies between L1 and L0 (On).

The assertion of USB suspend signal indicates that there was no activity on the USB bus for the last 3 ms. At this time an interrupt is sent to the processor on which the software can start preparing the device for suspend.

If there is no activity for the next 2 ms, the USB need_clock signal will go low. This indicates that the USB main clock can be switched off.

When activity is detected on the USB bus, the USB suspend signal is deactivated and USB need_clock signal is activated. This process is fully combinatorial and hence no USB main clock is required to activate the US B need_clock signal.

11.4.6 Frame toggle output

The USB_FTOGGLE output pin reflects the 1 kHz clock derived from the incoming Start of Frame tokens sent by the USB host.

11.4.7 Clocking

The LPC11Uxx USB device controller has the following clock connections:

- USB main clock: The USB main clock is the 48 MHz +/- 500 ppm clock from the dedicated USB PLL or the main clock (see Table 27). If the main clock is used, the system PLL output must be 48 MHz and derived from the system oscillator. The USB main clock is used to recover the 12 MHz clock from the USB bus.
- AHB clock: This is the AHB system bus clock. The minimum frequency of the AHB clock is 16 MHz when the USB device controller is receiving or transmitting USB packets.

11.5 Pin description

The device controller can access one USB port.

Table 195. USB device pin description

Name	Direction	Description
V_{BUS}	I	V_{BUS} status input. When this function is not enabled via its corresponding IOCON register, it is driven HIGH internally.
$\overline{\text{USB\_CONNECT}}$	O	SoftConnect control signal.
USB_FTOGGLE	O	USB 1 ms SoF signal.
USB_DP	I/O	Positive differential data.
USB_DM	I/O	Negative differential data.

11.6 Register description

Table 196. Register overview: USB (base address: 0x4008 0000)

Name	Access	Address offset	Description	Reset value	Reference
DEVCMDSTAT	R/W	0x000	USB Device Command/Status register	0x00000800	Table 197
INFO	R/W	0x004	USB Info register	0	Table 198
EPLISTSTART	R/W	0x008	USB EP Command/Status List start address	0	Table 199
DATABUFSTART	R/W	0x00C	USB Data buffer start address	0	Table 200
LPM	R/W	0x010	USB Link Power Management register	0	Table 201
EPSKIP	R/W	0x014	USB Endpoint skip	0	Table 202
EPINUSE	R/W	0x018	USB Endpoint Buffer in use	0	Table 203
EPBUFCFG	R/W	0x01C	USB Endpoint Buffer Configuration register	0	Table 204
INTSTAT	R/W	0x020	USB interrupt status register	0	Table 205
INTEN	R/W	0x024	USB interrupt enable register	0	Table 206

Table 196. Register overview: USB (base address: 0x4008 0000)

Name	Access	Address offset	Description	Reset value	Reference
INTSETSTAT	R/W	0x028	USB set interrupt status register	0	Table 207
INTROUTING	R/W	0x02C	USB interrupt routing register	0	Table 208
EPTOGGLE	R	0x034	USB Endpoint toggle register	0	Table 209

11.6.1 USB Device Command/Status register (DEVCMDSTAT)

Table 197. USB Device Command/Status register (DEVCMDSTAT, address 0x4008 0000) bit description

Bit	Symbol	Value	Description	Reset value	Access
6:0	DEV_ADDR		USB device address. After bus reset, the address is reset to 0x00. If the enable bit is set, the device will respond on packets for function address DEV_ADDR. When receiving a SetAddress Control Request from the USB host, software must program the new address before completing the status phase of the SetAddress Control Request.	0	RW
7	DEV_EN		USB device enable. If this bit is set, the HW will start responding on packets for function address DEV_ADDR.	0	RW
8	SETUP		SETUP token received. If a SETUP token is received and acknowledged by the device, this bit is set. As long as this bit is set all received IN and OUT tokens will be NAKed by HW. SW must clear this bit by writing a one. If this bit is zero, HW will handle the tokens to the CTRL EP0 as indicated by the CTRL EP0 IN and OUT data information programmed by SW.	0	RWC
9	PLL_ON		Always PLL Clock on:	0	RW
		0	USB_NeedClk functional		
		1	USB_NeedClk always 1. Clock will not be stopped in case of suspend.		
10	-		Reserved.	0	RO
11	LPM_SUP		LPM Supported:	1	RW
		0	LPM not supported.		
		1	LPM supported.		
12	INTONNAK_AO		Interrupt on NAK for interrupt and bulk OUT EP	0	RW
		0	Only acknowledged packets generate an interrupt		
		1	Both acknowledged and NAKed packets generate interrupts.		
13	INTONNAK_AI		Interrupt on NAK for interrupt and bulk IN EP	0	RW
		0	Only acknowledged packets generate an interrupt		
		1	Both acknowledged and NAKed packets generate interrupts.		
14	INTONNAK_CO		Interrupt on NAK for control OUT EP	0	RW
		0	Only acknowledged packets generate an interrupt		
		1	Both acknowledged and NAKed packets generate interrupts.		
15	INTONNAK_CI		Interrupt on NAK for control IN EP	0	RW
		0	Only acknowledged packets generate an interrupt		
		1	Both acknowledged and NAKed packets generate interrupts.		

Table 197. USB Device Command/Status register (DEVCMDSTAT, address 0x4008 0000) bit description

Bit	Symbol	Value	Description	Reset value	Access
16	DCON		Device status - connect. The connect bit must be set by SW to indicate that the device must signal a connect. The pull-up resistor on USB_DP will be enabled when this bit is set and the VbusDebounced bit is one.	0	RW
17	DSUS		Device status - suspend. The suspend bit indicates the current suspend state. It is set to 1 when the device hasn't seen any activity on its upstream port for more than 3 milliseconds. It is reset to 0 on any activity. When the device is suspended (Suspend bit DSUS = 1) and the software writes a 0 to it, the device will generate a remote wake-up. This will only happen when the device is connected (Connect bit = 1). When the device is not connected or not suspended, a writing a 0 has no effect. Writing a 1 never has an effect.	0	RW
18	-		Reserved.	0	RO
19	LPM_SUS		Device status - LPM Suspend. This bit represents the current LPM suspend state. It is set to 1 by HW when the device has acknowledged the LPM request from the USB host and the Token Retry Time of 10 μs has elapsed. When the device is in the LPM suspended state (LPM suspend bit = 1) and the software writes a zero to this bit, the device will generate a remote walk-up. Software can only write a zero to this bit when the LPM_REWP bit is set to 1. HW resets this bit when it receives a host initiated resume. HW only updates the LPM_SUS bit when the LPM_SUPP bit is equal to one.	0	RW
20	LPM_REWP		LPM Remote Wake-up Enabled by USB host. HW sets this bit to one when the bRemoteWake bit in the LPM extended token is set to 1. HW will reset this bit to 0 when it receives the host initiated LPM resume, when a remote wake-up is sent by the device or when a USB bus reset is received. Software can use this bit to check if the remote wake-up feature is enabled by the host for the LPM transaction.	0	RO
23:20	-		Reserved.	0	RO
24	DCON_C		Device status - connect change. The Connect Change bit is set when the device's pull-up resistor is disconnected because VBus disappeared. The bit is reset by writing a one to it.	0	RWC
25	DSUS_C		Device status - suspend change. The suspend change bit is set to 1 when the suspend bit toggles. The suspend bit can toggle because: - The device goes in the suspended state - The device is disconnected - The device receives resume signaling on its upstream port. The bit is reset by writing a one to it.	0	RWC
26	DRES_C		Device status - reset change. This bit is set when the device received a bus reset. On a bus reset the device will automatically go to the default state (unconfigured and responding to address 0). The bit is reset by writing a one to it.	0	RWC

Table 197. USB Device Command/Status register (DEVCMDSTAT, address 0x4008 0000) bit description

Bit	Symbol	Value	Description	Reset value	Access
27	-		Reserved.	0	RO
28	VBUSDEBOUNCED		This bit indicates if Vbus is detected or not. The bit raises immediately when Vbus becomes high. It drops to zero if Vbus is low for at least 3 ms. If this bit is high and the DCon bit is set, the HW will enable the pull-up resistor to signal a connect.	0	RO
31:29	-		Reserved.	0	RO

11.6.2 USB Info register (INFO)

Table 198. USB Info register (INFO, address 0x4008 0004) bit description

Bit	Symbol	Value	Description	Reset value	Access
10:0	FRAME_NR		Frame number. This contains the frame number of the last successfully received SOF. 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.	0	RO
14:11	ERR_CODE		The error code which last occurred:	0	RW
		0x0	No error		
		0x1	PID encoding error		
		0x2	PID unknown		
		0x3	Packet unexpected		
		0x4	Token CRC error		
		0x5	Data CRC error		
		0x6	Time out		
		0x7	Babble		
		0x8	Truncated EOP		
		0x9	Sent/Received NAK		
		0xA	Sent Stall		
		0xB	Overrun		
		0xC	Sent empty packet		
		0xD	Bitstuff error		
		0xE	Sync error		
		0xF	Wrong data toggle		
15	-		Reserved.	0	RO
31:16	-	-	Reserved	-	RO

11.6.3 USB EP Command/Status List start address (EPLISTSTART)

This 32-bit register indicates the start address of the USB EP Command/Status List. Only a subset of these bits is programmable by software. The 8 least-significant bits are hardcoded to zero because the list must start on a 256 byte boundary. Bits 31 to 8 can be programmed by software.

Table 199. USB EP Command/Status List start address (EPLISTSTART, address 0x4008 0008) bit description

Bit	Symbol	Description	Reset value	Access
7:0	-	Reserved	0	RO
31:8	EP_LIST	Start address of the USB EP Command/Status List.	0	R/W

11.6.4 USB Data buffer start address (DATABUFSTART)

This register indicates the page of the AHB address where the endpoint data can be located.

Table 200. USB Data buffer start address (DATABUFSTART, address 0x4008 000C) bit description

Bit	Symbol	Description	Reset value	Access
21:0	-	Reserved	0	R
31:22	DA_BUF	Start address of the buffer pointer page where all endpoint data buffers are located.	0	R/W

11.6.5 USB Link Power Management register (LPM)

Table 201. Link Power Management register (LPM, address 0x4008 0010) bit description

Bit	Symbol	Description	Reset value	Access
3:0	HIRD_HW	Host Initiated Resume Duration - HW. This is the HIRD value from the last received LPM token	0	RO
7:4	HIRD_SW	Host Initiated Resume Duration - SW. This is the time duration required by the USB device system to come out of LPM initiated suspend after receiving the host initiated LPM resume.	0	R/W
8	DATA_PENDING	As long as this bit is set to one and LPM supported bit is set to one, HW will return a NYET handshake on every LPM token it receives. If LPM supported bit is set to one and this bit is zero, HW will return an ACK handshake on every LPM token it receives. If SW has still data pending and LPM is supported, it must set this bit to 1.	0	R/W
31:9	RESERVED	Reserved	0	RO

11.6.6 USB Endpoint skip (EPSKIP)

Table 202. USB Endpoint skip (EPSKIP, address 0x4008 0014) bit description

Bit	Symbol	Description	Reset value	Access
29:0	SKIP	Endpoint skip: Writing 1 to one of these bits, will indicate to HW that it must deactivate the buffer assigned to this endpoint and return control back to software. When HW has deactivated the endpoint, it will clear this bit, but it will not modify the EPINUSE bit. An interrupt will be generated when the Active bit goes from 1 to 0. Note: In case of double-buffering, HW will only clear the Active bit of the buffer indicated by the EPINUSE bit.	0	R/W
31:30	-	Reserved	0	R

11.6.7 USB Endpoint Buffer in use (EPINUSE)

Table 203. USB Endpoint Buffer in use (EPINUSE, address 0x4008 0018) bit description

Bit	Symbol	Description	Reset value	Access
1:0	-	Reserved. Fixed to zero because the control endpoint zero is fixed to single-buffering for each physical endpoint.	0	R
9:2	BUF	Buffer in use: This register has one bit per physical endpoint. 0: HW is accessing buffer 0. 1: HW is accessing buffer 1.	0	R/W
31:10	-	Reserved	0	R

11.6.8 USB Endpoint Buffer Configuration (EPBUFCFG)

Table 204. USB Endpoint Buffer Configuration (EPBUFCFG, address 0x4008 001C) bit description

Bit	Symbol	Description	Reset value	Access
1:0	-	Reserved. Fixed to zero because the control endpoint zero is fixed to single-buffering for each physical endpoint.	0	R
9:2	BUF_SB	Buffer usage: This register has one bit per physical endpoint. 0: Single-buffer. 1: Double-buffer. If the bit is set to single-buffer (0), it will not toggle the corresponding EPINUSE bit when it clears the active bit. If the bit is set to double-buffer (1), HW will toggle the EPINUSE bit when it clears the Active bit for the buffer.	0	R/W
31:10	-	Reserved	0	R

11.6.9 USB interrupt status register (INTSTAT)

Table 205. USB interrupt status register (INTSTAT, address 0x4008 0020) bit description

Bit	Symbol	Description	Reset value	Access
0	EP0OUT	Interrupt status register bit for the Control EP0 OUT direction. This bit will be set if NBytes transitions to zero or the skip bit is set by software or a SETUP packet is successfully received for the control EP0. If the IntOnNAK_CO is set, this bit will also be set when a NAK is transmitted for the Control EP0 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
1	EP0IN	Interrupt status register bit for the Control EP0 IN direction. This bit will be set if NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_CI is set, this bit will also be set when a NAK is transmitted for the Control EP0 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
2	EP1OUT	Interrupt status register bit for the EP1 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP1 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
3	EP1IN	Interrupt status register bit for the EP1 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP1 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
4	EP2OUT	Interrupt status register bit for the EP2 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP2 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
5	EP2IN	Interrupt status register bit for the EP2 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP2 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
6	EP3OUT	Interrupt status register bit for the EP3 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP3 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC

Table 205. USB interrupt status register (INTSTAT, address 0x4008 0020) bit description

Bit	Symbol	Description	Reset value	Access
7	EP3IN	Interrupt status register bit for the EP3 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP3 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
8	EP4OUT	Interrupt status register bit for the EP4 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP4 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
9	EP4IN	Interrupt status register bit for the EP4 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP4 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
29:10	-	Reserved	0	RO
30	FRAME_INT	Frame interrupt. This bit is set to one every millisecond when the VbusDebounced bit and the DCON bit are set. This bit can be used by software when handling isochronous endpoints. Software can clear this bit by writing a one to it.	0	R/WC
31	DEV_INT	Device status interrupt. This bit is set by HW when one of the bits in the Device Status Change register are set. Software can clear this bit by writing a one to it.	0	R/WC

11.6.10 USB interrupt enable register (INTEN)

Table 206. USB interrupt enable register (INTEN, address 0x4008 0024) bit description

Bit	Symbol	Description	Reset value	Access
9:0	EP_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W
29:10	-	Reserved	0	RO
30	FRAME_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W
31	DEV_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W

11.6.11 USB set interrupt status register (INTSETSTAT)

Table 207. USB set interrupt status register (INTSETSTAT, address 0x4008 0028) bit description

Bit	Symbol	Description	Reset value	Access
9:0	EP_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W
29:10	-	Reserved	0	RO
30	FRAME_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W
31	DEV_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W

11.6.12 USB interrupt routing register (INTROUTING)

Table 208. USB interrupt routing register (INTROUTING, address 0x4008 002C) bit description

Bit	Symbol	Description	Reset value	Access
9:0	ROUTE_INT9_0	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W
29:10	-	Reserved	0	RO
30	ROUTE_INT30	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W
31	ROUTE_INT31	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W

11.6.13 USB Endpoint toggle (EPTOGGLE)

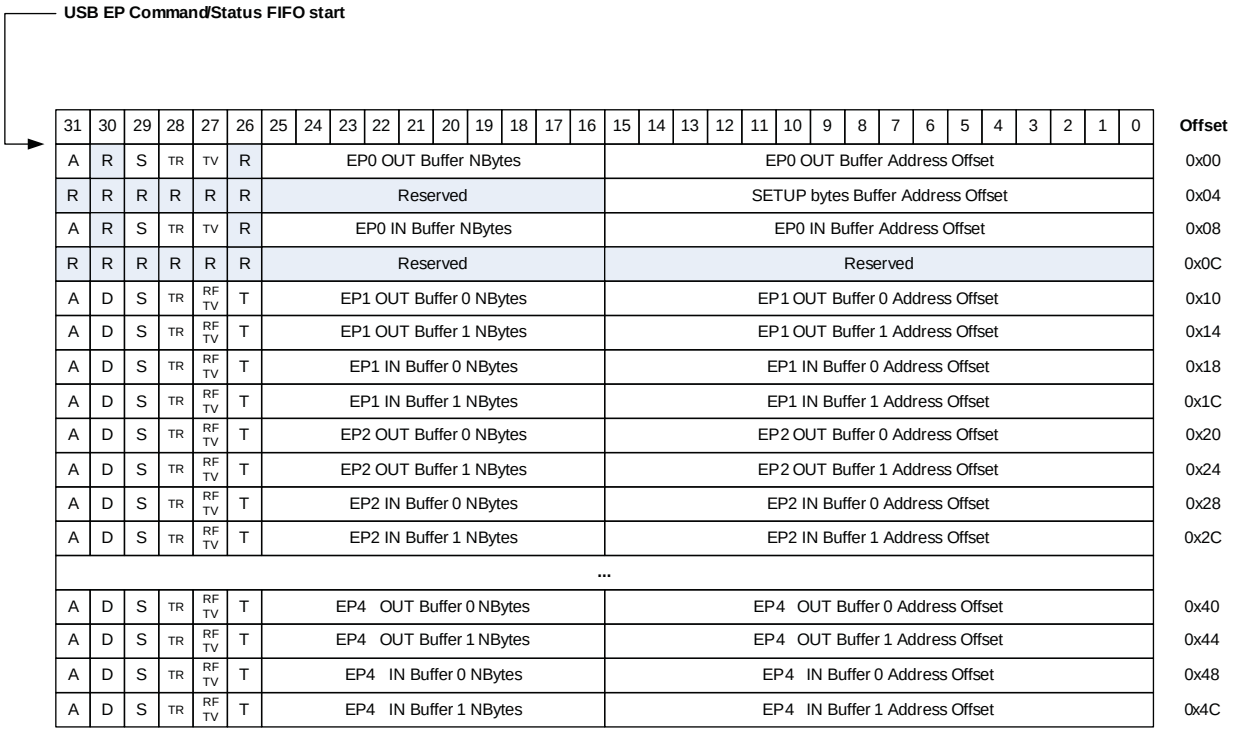
Table 209. USB Endpoint toggle (EPTOGGLE, address 0x4008 0034) bit description

Bit	Symbol	Description	Reset value	Access
9:0	TOGGLE	Endpoint data toggle: This field indicates the current value of the data toggle for the corresponding endpoint.	0	R
31:10	-	Reserved	0	R

11.7 Functional description

11.7.1 Endpoint command/status list

Figure 20 gives an overview on how the Endpoint List is organized in memory. The USB EP Command/Status List start register points to the start of the list that contains all the endpoint information in memory. The order of the endpoints is fixed as shown in the picture.



31	30	29	28	27	26	25 … 16	15 … 0	Offset
A	R	S	TR	TV	R	EP0 OUT Buffer NBytes	EP0 OUT Buffer Address Offset	0x00
R	R	R	R	R	R	Reserved	SETUP bytes Buffer Address Offset	0x04
A	R	S	TR	TV	R	EP0 IN Buffer NBytes	EP0 IN Buffer Address Offset	0x08
R	R	R	R	R	R	Reserved	Reserved	0x0C
A	D	S	TR	RF TV	T	EP1 OUT Buffer 0 NBytes	EP1 OUT Buffer 0 Address Offset	0x10
A	D	S	TR	RF TV	T	EP1 OUT Buffer 1 NBytes	EP1 OUT Buffer 1 Address Offset	0x14
A	D	S	TR	RF TV	T	EP1 IN Buffer 0 NBytes	EP1 IN Buffer 0 Address Offset	0x18
A	D	S	TR	RF TV	T	EP1 IN Buffer 1 NBytes	EP1 IN Buffer 1 Address Offset	0x1C
A	D	S	TR	RF TV	T	EP2 OUT Buffer 0 NBytes	EP2 OUT Buffer 0 Address Offset	0x20
A	D	S	TR	RF TV	T	EP2 OUT Buffer 1 NBytes	EP2 OUT Buffer 1 Address Offset	0x24
A	D	S	TR	RF TV	T	EP2 IN Buffer 0 NBytes	EP2 IN Buffer 0 Address Offset	0x28
A	D	S	TR	RF TV	T	EP2 IN Buffer 1 NBytes	EP2 IN Buffer 1 Address Offset	0x2C
...								
A	D	S	TR	RF TV	T	EP4 OUT Buffer 0 NBytes	EP4 OUT Buffer 0 Address Offset	0x40
A	D	S	TR	RF TV	T	EP4 OUT Buffer 1 NBytes	EP4 OUT Buffer 1 Address Offset	0x44
A	D	S	TR	RF TV	T	EP4 IN Buffer 0 NBytes	EP4 IN Buffer 0 Address Offset	0x48
A	D	S	TR	RF TV	T	EP4 IN Buffer 1 NBytes	EP4 IN Buffer 1 Address Offset	0x4C

Fig 20. Endpoint command/status list (see also Table 210)

Table 210. Endpoint commands

Symbol	Access	Description
A	RW	Active The buffer is enabled. HW can use the buffer to store received OUT data or to transmit data on the IN endpoint. Software can only set this bit to '1'. As long as this bit is set to one, software is not allowed to update any of the values in this 32-bit word. In case software wants to deactivate the buffer, it must write a one to the corresponding "skip" bit in the USB Endpoint skip register. Hardware can only write this bit to zero. It will do this when it receives a short packet or when the NBytes field transitions to zero or when software has written a one to the "skip" bit.
D	RW	Disabled 0: The selected endpoint is enabled. 1: The selected endpoint is disabled. If a USB token is received for an endpoint that has the disabled bit set, hardware will ignore the token and not return any data or handshake. When a bus reset is received, software must set the disable bit of all endpoints to 1. Software can only modify this bit when the active bit is zero.
S	RW	Stall 0: The selected endpoint is not stalled 1: The selected endpoint is stalled The Active bit has always higher priority than the Stall bit. This means that a Stall handshake is only sent when the active bit is zero and the stall bit is one. Software can only modify this bit when the active bit is zero.
TR	RW	Toggle Reset When software sets this bit to one, the HW will set the toggle value equal to the value indicated in the "toggle value" (TV) bit. For the control endpoint zero, this is not needed to be used because the hardware resets the endpoint toggle to one for both directions when a setup token is received. For the other endpoints, the toggle can only be reset to zero when the endpoint is reset.
RF / TV	RW	Rate Feedback mode / Toggle value For bulk endpoints and isochronous endpoints this bit is reserved and must be set to zero. For the control endpoint zero this bit is used as the toggle value. When the toggle reset bit is set, the data toggle is updated with the value programmed in this bit. When the endpoint is used as an interrupt endpoint, it can be set to the following values. 0: Interrupt endpoint in 'toggle mode' 1: Interrupt endpoint in 'rate feedback mode'. This means that the data toggle is fixed to zero for all data packets. When the interrupt endpoint is in 'rate feedback mode', the TR bit must always be set to zero.

Table 210. Endpoint commands

Symbol	Access	Description
T	RW	Endpoint Type 0: Generic endpoint. The endpoint is configured as a bulk or interrupt endpoint 1: Isochronous endpoint
NBytes	RW	For OUT endpoints this is the number of bytes that can be received in this buffer. For IN endpoints this is the number of bytes that must be transmitted. HW decrements this value with the packet size every time when a packet is successfully transferred. Note: If a short packet is received on an OUT endpoint, the active bit will be cleared and the NBytes value indicates the remaining buffer space that is not used. Software calculates the received number of bytes by subtracting the remaining NBytes from the programmed value.
Address Offset	RW	Bits 21 to 6 of the buffer start address. The address offset is updated by hardware after each successful reception/transmission of a packet. Hardware increments the original value with the integer value when the packet size is divided by 64. Examples: • If an isochronous packet of 200 bytes is successfully received, the address offset is incremented by 3. • If a packet of 64 bytes is successfully received, the address offset is incremented by 1. • If a packet of less than 64 bytes is received, the address offset is not incremented.

Remark: When receiving a SETUP token for endpoint zero, the HW will only read the SETUP bytes Buffer Address offset to know where it has to store the received SETUP bytes. The hardware will ignore all other fields. In case the SETUP stage contains more than 8 bytes, it will only write the first 8 bytes to memory. A USB compliant host must never send more than 8 bytes during the SETUP stage.

11.7.2 Control endpoint 0

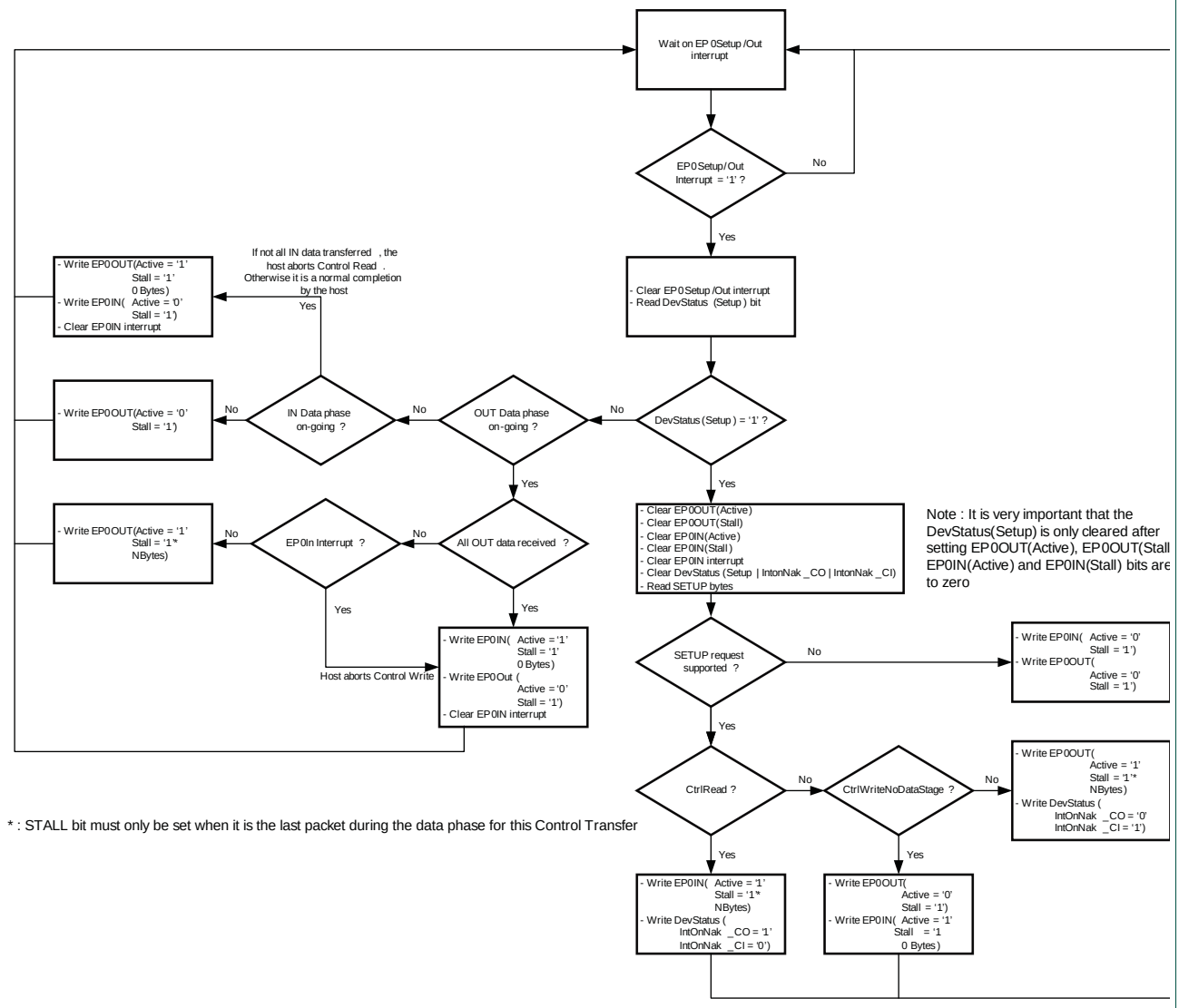
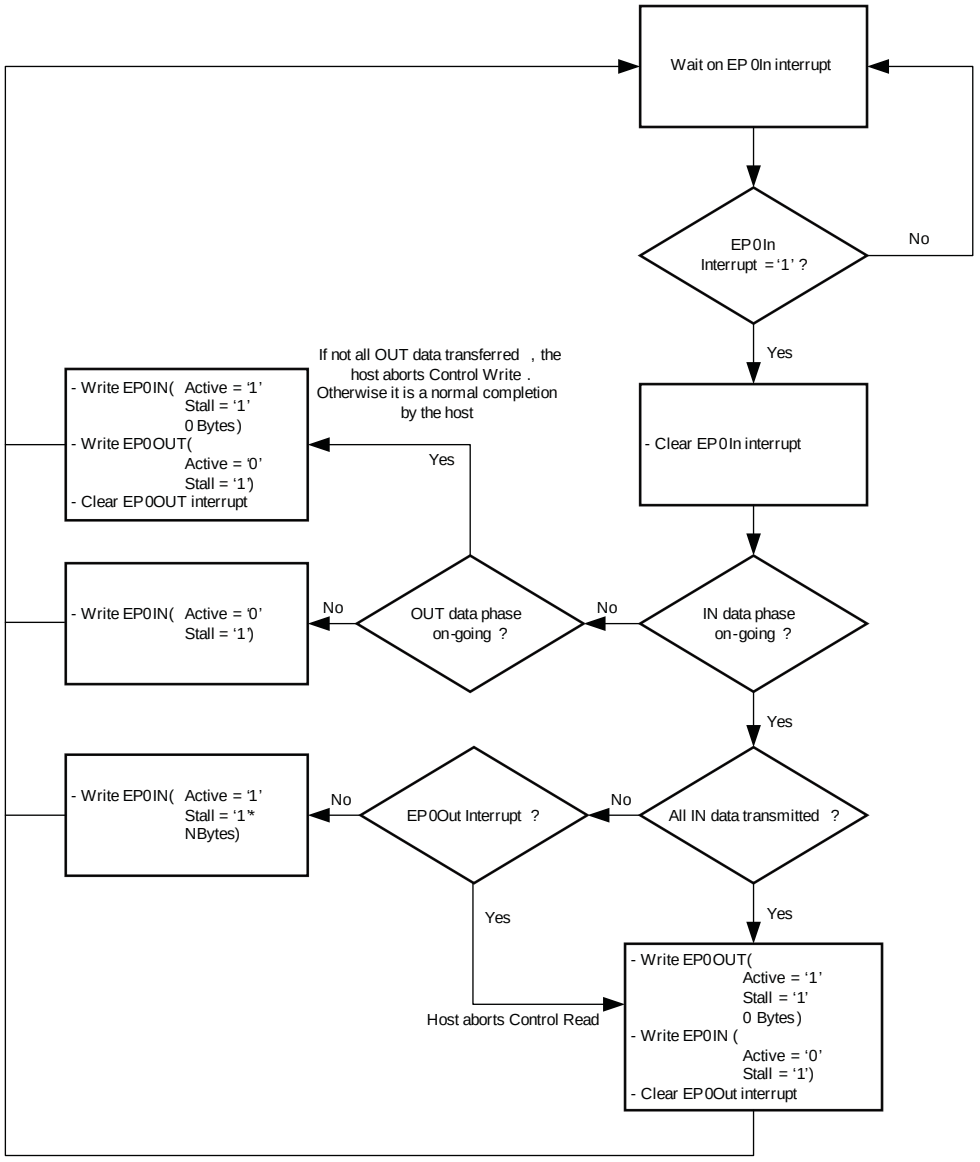


Fig 21. Flowchart of control endpoint 0 - OUT direction



* : STALL bit must only be set when it is the last packet during the data phase for this Control Transfer

Fig 22. Flowchart of control endpoint 0 - IN direction

11.7.3 Generic endpoint: single-buffering

To enable single-buffering, software must set the corresponding "USB EP Buffer Config" bit to zero. In the "USB EP Buffer in use" register, software can indicate which buffer is used in this case.

When software wants to transfer data, it programs the different bits in the Endpoint command/status entry and sets the active bits. The hardware will transmit/receive multiple packets for this endpoint until the NBytes value is equal to zero. When NBytes goes to zero, hardware clears the active bit and sets the corresponding interrupt status bit.

Software must wait until hardware has cleared the Active bit to change some of the command/status bits. This prevents hardware from overwriting a new value programmed by software with some old values that were still cached.

If software wants to disable the active bit before the hardware has finished handling the complete buffer, it can do this by setting the corresponding endpoint skip bit in USB endpoint skip register.

11.7.4 Generic endpoint: double-buffering

To enable double-buffering, software must set the corresponding "USB EP Buffer Config" bit to one. The "USB EP Buffer in use" register indicates which buffer will be used by HW when the next token is received.

When HW clears the active bit of the current buffer in use, it will switch the buffer in use. Software can also force HW to use a certain buffer by writing to the "USB EP Buffer in use" bit.

11.7.5 Special cases

11.7.5.1 Use of the Active bit

The use of the Active bit is a bit different between OUT and IN endpoints.

When data must be received for the OUT endpoint, the software will set the Active bit to one and program the NBytes field to the maximum number of bytes it can receive.

When data must be transmitted for an IN endpoint, the software sets the Active bit to one and programs the NBytes field to the number of bytes that must be transmitted.

11.7.5.2 Generation of a STALL handshake

Special care must be taken when programming the endpoint to send a STALL handshake. A STALL handshake is only sent in the following situations:

- The endpoint is enabled (Disabled bit = 0)
- The active bit of the endpoint is set to 0. (No packet needs to be received/transmitted for that endpoint).
- The stall bit of the endpoint is set to one.

11.7.5.3 Clear Feature (endpoint halt)

When a non-control endpoint has returned a STALL handshake, the host will send a Clear Feature (Endpoint Halt) for that endpoint. When the device receives this request, the endpoint must be unstalled and the toggle bit for that endpoint must be reset back to zero. In order to do that the software must program the following items for the endpoint that is indicated.

If the endpoint is used in single-buffer mode, program the following:

- Set STALL bit (S) to 0.
- Set toggle reset bit (TR) to 1 and set toggle value bit (TV) to 0.

If the endpoint is used in double-buffer mode, program the following:

- Set the STALL bit of both buffer 0 and buffer 1 to 0.
- Read the buffer in use bit for this endpoint.
- Set the toggle reset bit (TR) to 1 and set the toggle value bit (TV) to 0 for the buffer indicated by the buffer in use bit.

11.7.5.4 Set configuration

When a SetConfiguration request is received with a configuration value different from zero, the device software must enable all endpoints that will be used in this configuration and reset all the toggle values. To do so, it must generate the procedure explained in Section 11.7.5.3 for every endpoint that will be used in this configuration.
For all endpoints that are not used in this configuration, it must set the Disabled bit (D) to one.

11.7.6 USB wake-up

11.7.6.1 Waking up from Deep-sleep and Power-down modes on USB activity

To allow the LPC11Uxx to wake up from Deep-sleep or Power-down mode on USB activity, complete the following steps:

1. Set bit AP_CLK in the USBCLKCTRL register (Table 40) to 0 (default) to enable automatic control of the USB need_clock signal.
2. Wait until USB activity is suspended by polling the DSUS bit in the DSVCMD_STAT register (DSUS = 1).
3. The USB need_clock signal will be deasserted after another 2 ms. Poll the USBCLKST register until the USB need_clock status bit is 0 (Table 41).
4. Once the USBCLKST register returns 0, enable the USB activity wake-up interrupt in the NVIC (# 30) and clear it.
5. Set bit 1 in the USBCLKCTRL register to 1 to trigger the USB activity wake-up interrupt on the rising edge of the USB need_clock signal.
6. Enable the wake-up from Deep-sleep or Power-down modes on this interrupt by enabling the USB need_clock signal in the STARTERP1 register (Table 43, bit 19).
7. Enter Deep-sleep or Power-down modes by writing to the PCON register.
8. Execute a WFI instruction.

The LPC11Uxx will automatically wake up and resume execution on USB activity.

11.7.6.2 Remote wake-up

To issue a remote wake-up when the USB activity is suspended, complete the following steps:

1. Set bit AP_CLK in the USBCLKCTRL register to 0 (Table 40, default) to enable automatic control of the USB need_clock signal.
2. When it is time to issue a remote wake-up, turn on the USB clock and enable the USB clock source.
3. Force the USB clock on by writing a 1 to bit AP_CLK (Table 40, bit 0) in the USBCLKCTRL register.
4. Write a 0 to the DSUS bit in the DSVCMD_STAT register.

5. Wait until the USB leaves the suspend state by polling the DSUS bit in the DSVCMD_STAT register (DSUS =0).
6. Clear the AP_CLK bit (Table 40, bit 0) in the USBCLKCTRL to enable automatic USB clock control.

12.1 How to read this chapter

The USART controller is available on all LPC11Uxx parts.

12.2 Basic configuration

The USART is configured as follows:

- Pins: The USART pins must be configured in the corresponding IOCON registers (see Section 7.4).
- The USART block is enabled through the SYSAHBCLKCTRL register (see Table 23).
- The peripheral USART clock (PCLK), which is used by the USART baud rate generator, is controlled by the UARTCLKDIV register (see Table 25).

12.3 Features

- 16-byte receive and transmit FIFOs.
- Register locations conform to ‘550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Built-in baud rate generator.
- Software or hardware flow control.
- RS-485/EIA-485 9-bit mode support with output enable.
- $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ flow control and other modem control signals.
- 1X-clock send or receive.
- ISO 7816-3 compliant smart card interface.
- IrDA support.

12.4 Pin description

Table 211. USART pin description

Pin	Type	Description
RXD	Input	Serial Input. Serial receive data.
TXD	Output	Serial Output. Serial transmit data (input/output in smart card mode).
$\overline{\text{RTS}}$	Output	Request To Send. RS-485 direction control pin.
$\overline{\text{CTS}}$	Input	Clear To Send.
$\overline{\text{DTR}}$	Output	Data Terminal Ready.
$\overline{\text{DSR}}$	Input	Data Set Ready. (Not available on HVQFN33-pin packages).
$\overline{\text{DCD}}$	Input	Data Carrier Detect.
$\overline{\text{RI}}$	Input	Ring Indicator. (Not available on HVQFN33-pin packages).
SCLK	I/O	Serial Clock.

12.5 Register description

The USART contains registers organized as shown in Table 212. The Divisor Latch Access Bit (DLAB) is contained in the LCR register bit 7 and enables access to the Divisor Latches.

Offsets/addresses not shown in Table 212 are reserved.

Table 212. Register overview: USART (base address: 0x4000 8000)

Name	Access	Address offset	Description	Reset value[1]	Reference
RBR	RO	0x000	Receiver Buffer Register. Contains the next received character to be read. (DLAB=0)	NA	Table 213
THR	WO	0x000	Transmit Holding Register. The next character to be transmitted is written here. (DLAB=0)	NA	Table 214
DLL	R/W	0x000	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. (DLAB=1)	0x01	Table 215
DLM	R/W	0x004	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. (DLAB=1)	0	Table 216
IER	R/W	0x004	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential USART interrupts. (DLAB=0)	0	Table 217
IIR	RO	0x008	Interrupt ID Register. Identifies which interrupt(s) are pending.	0x01	Table 218
FCR	WO	0x008	FIFO Control Register. Controls USART FIFO usage and modes.	0	Table 219
LCR	R/W	0x00C	Line Control Register. Contains controls for frame formatting and break generation.	0	Table 221
MCR	R/W	0x010	Modem Control Register.	0	Table 222
LSR	RO	0x014	Line Status Register. Contains flags for transmit and receive status, including line errors.	0x60	Table 224
MSR	RO	0x018	Modem Status Register.	0	Table 225
SCR	R/W	0x01C	Scratch Pad Register. Eight-bit temporary storage for software.	0	Table 226
ACR	R/W	0x020	Auto-baud Control Register. Contains controls for the auto-baud feature.	0	Table 227
ICR	R/W	0x024	IrDA Control Register. Enables and configures the IrDA (remote control) mode.	0	Table 228
FDR	R/W	0x028	Fractional Divider Register. Generates a clock input for the baud rate divider.	0x10	Table 230
OSR	R/W	0x02C	Oversampling Register. Controls the degree of oversampling during each bit time.	0xF0	Table 232
TER	R/W	0x030	Transmit Enable Register. Turns off USART transmitter for use with software flow control.	0x80	Table 233
HDEN	R/W	0x040	Half duplex enable register.	0	Table 234
SCICTRL	R/W	0x048	Smart Card Interface Control register. Enables and configures the Smart Card Interface feature.	0	Table 235

Table 212. Register overview: USART (base address: 0x4000 8000)

Name	Access	Address offset	Description	Reset value[1]	Reference
RS485CTRL	R/W	0x04C	RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes.	0	Table 236
RS485ADRMATCH	R/W	0x050	RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode.	0	Table 237
RS485DLY	R/W	0x054	RS-485/EIA-485 direction control delay.	0	Table 238
SYNCCTRL	R/W	0x058	Synchronous mode control register.	0	Table 239

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

12.5.1 USART Receiver Buffer Register (when DLAB = 0, Read Only)

The RBR is the top byte of the USART 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) contains the first-received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeros.

The Divisor Latch Access Bit (DLAB) in the LCR must be zero in order to access the RBR. The RBR is always Read Only.

Since PE, FE and BI bits (see Table 224) correspond to the byte 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 LSR register, and then to read a byte from the RBR.

Table 213. USART Receiver Buffer Register when DLAB = 0, Read Only (RBR - address 0x4000 8000) bit description

Bit	Symbol	Description	Reset Value
7:0	RBR	The USART Receiver Buffer Register contains the oldest received byte in the USART RX FIFO.	undefined
31:8	-	Reserved	-

12.5.2 USART Transmitter Holding Register (when DLAB = 0, Write Only)

The THR is the top byte of the USART 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 the LCR must be zero in order to access the THR. The THR is always Write Only.

Table 214. USART Transmitter Holding Register when DLAB = 0, Write Only (THR - address 0x4000 8000) bit description

Bit	Symbol	Description	Reset Value
7:0	THR	Writing to the USART Transmit Holding Register causes the data to be stored in the USART transmit FIFO. The byte will be sent when it is the oldest byte in the FIFO and the transmitter is available.	NA
31:8	-	Reserved	-

12.5.3 USART Divisor Latch LSB and MSB Registers (when DLAB = 1)

The USART Divisor Latch is part of the USART Baud Rate Generator and holds the value used (optionally with the Fractional Divider) to divide the UART_PCLK clock in order to produce the baud rate clock, which must be the multiple of the desired baud rate that is specified by the Oversampling Register (typically 16X). The DLL and DLM registers together form a 16-bit divisor. DLL contains the lower 8 bits of the divisor and DLM contains the higher 8 bits. A zero value is treated like 0x0001. The Divisor Latch Access Bit (DLAB) in the LCR must be one in order to access the USART Divisor Latches. Details on how to select the right value for DLL and DLM can be found in Section 12.5.14.

Table 215. USART Divisor Latch LSB Register when DLAB = 1 (DLL - address 0x4000 8000) bit description

Bit	Symbol	Description	Reset value
7:0	DLLSB	The USART Divisor Latch LSB Register, along with the DLM register, determines the baud rate of the USART.	0x01
31:8	-	Reserved	-

Table 216. USART Divisor Latch MSB Register when DLAB = 1 (DLM - address 0x4000 8004) bit description

Bit	Symbol	Description	Reset value
7:0	DLMSB	The USART Divisor Latch MSB Register, along with the DLL register, determines the baud rate of the USART.	0x00
31:8	-	Reserved	-

12.5.4 USART Interrupt Enable Register (when DLAB = 0)

The IER is used to enable the various USART interrupt sources.

Table 217. USART Interrupt Enable Register when DLAB = 0 (IER - address 0x4000 8004) bit description

Bit	Symbol	Value	Description	Reset value
0	RBRINTEN		RBR Interrupt Enable. Enables the Receive Data Available interrupt. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupt.	
		1	Enable the RDA interrupt.	
1	THREINTEN		THRE Interrupt Enable. Enables the THRE interrupt. The status of this interrupt can be read from LSR[5].	0
		0	Disable the THRE interrupt.	
		1	Enable the THRE interrupt.	
2	RLSINTEN		Enables the Receive Line Status interrupt. The status of this interrupt can be read from LSR[4:1].	-
		0	Disable the RLS interrupt.	
		1	Enable the RLS interrupt.	
3	MSINTEN		Enables the Modem Status interrupt. The components of this interrupt can be read from the MSR.	
		0	Disable the MS interrupt.	
		1	Enable the MS interrupt.	

Table 217. USART Interrupt Enable Register when DLAB = 0 (IER - address 0x4000 8004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
7:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.5.5 USART Interrupt Identification Register (Read Only)

IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during a IIR access. If an interrupt occurs during a IIR access, the interrupt is recorded for the next IIR access.

Table 218. USART Interrupt Identification Register Read only (IIR - address 0x4004 8008) bit description

Bit	Symbol	Value	Description	Reset value
0	INTSTATUS		Interrupt status. Note that IIR[0] is active low. The pending interrupt can be determined by evaluating IIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	INTID		Interrupt identification. IER[3:1] identifies an interrupt corresponding to the USART Rx FIFO. All other values of IER[3:1] not listed below are reserved.	0
		0x3	1 - Receive Line Status (RLS).	
		0x2	2a - Receive Data Available (RDA).	
		0x6	2b - Character Time-out Indicator (CTI).	
		0x1	3 - THRE Interrupt.	
		0x0	4 - Modem status	
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFOEN		These bits are equivalent to FCR[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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Bits IIR[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 one and 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 219. Given the status of IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The USART RLS interrupt (IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the USART RX input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The USART Rx error condition that set the interrupt can be observed via LSR[4:1]. The interrupt is cleared upon a LSR read.

The USART RDA interrupt (IIR[3:1] = 010) shares the second level priority with the CTI interrupt (IIR[3:1] = 110). The RDA is activated when the USART Rx FIFO reaches the trigger level defined in FCR7:6 and is reset when the USART 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 (IIR[3:1] = 110) is a second level interrupt and is set when the USART Rx FIFO contains at least one character and no USART Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any USART Rx FIFO activity (read or write of USART RSR) will clear the interrupt. This interrupt is intended to flush the USART RBR after a message has been received that is not a multiple of the trigger level size. For example, if a 105 character message was to be sent 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 219. USART Interrupt Handling

IIR[3:0] value[1]	Priority	Interrupt type	Interrupt source	Interrupt reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE[2] or PE[2] or FE[2] or BI[2]	LSR Read[2]
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (FCR0=1)	RBR Read[3] or USART FIFO drops below trigger level

Table 219. USART Interrupt Handling

IIR[3:0] value[1]	Priority	Interrupt type	Interrupt source	Interrupt reset
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	RBR Read[3]
0010	Third	THRE	THRE[2]	IIR Read[4] (if source of interrupt) or THR write
0000	Fourth	Modem Status	CTS, DSR, 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 12.5.9 “USART Line Status Register (Read-Only)”

[3] For details see Section 12.5.1 “USART Receiver Buffer Register (when DLAB = 0, Read Only)”

[4] For details see Section 12.5.5 “USART Interrupt Identification Register (Read Only)” and Section 12.5.2 “USART Transmitter Holding Register (when DLAB = 0, Write Only)”

The USART THRE interrupt (IIR[3:1] = 001) is a third level interrupt and is activated when the USART THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the USART 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 THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the USART THR FIFO has held two or more characters at one time and currently, the THR is empty. The THRE interrupt is reset when a THR write occurs or a read of the IIR occurs and the THRE is the highest interrupt (IIR[3:1] = 001).

The modem status interrupt (IIR3:1 = 000) is the lowest priority USART interrupt and is activated whenever there is a state change on the CTS, DCD, or DSR or a trailing edge on the RI pin. The source of the modem interrupt can be read in MSR3:0. Reading the MSR clears the modem interrupt.

12.5.6 USART FIFO Control Register (Write Only)

The FCR controls the operation of the USART RX and TX FIFOs.

Table 220. USART FIFO Control Register Write only (FCR - address 0x4000 8008) bit description

Bit	Symbol	Value	Description	Reset value
0	FIFOEN		FIFO enable	0
		0	USART FIFOs are disabled. Must not be used in the application.	
		1	Active high enable for both USART Rx and TX FIFOs and FCR[7:1] access. This bit must be set for proper USART operation. Any transition on this bit will automatically clear the USART FIFOs.	
1	RXFIFO RES		RX FIFO Reset	0
		0	No impact on either of USART FIFOs.	
		1	Writing a logic 1 to FCR[1] will clear all bytes in USART Rx FIFO, reset the pointer logic. This bit is self-clearing.	
2	TXFIFO RES		TX FIFO Reset	0
		0	No impact on either of USART FIFOs.	
		1	Writing a logic 1 to FCR[2] will clear all bytes in USART TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	-	-	Reserved	0
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RXTL		RX Trigger Level. These two bits determine how many USART FIFO characters must be received by the FIFO before an interrupt is activated.	0
		0x0	Trigger level 0 (1 character or 0x01).	
		0x1	Trigger level 1 (4 characters or 0x04).	
		0x2	Trigger level 2 (8 characters or 0x08).	
		0x3	Trigger level 3 (14 characters or 0x0E).	
31:8	-	-	Reserved	-

12.5.7 USART Line Control Register

The LCR determines the format of the data character that is to be transmitted or received.

Table 221. USART Line Control Register (LCR - address 0x4000 800C) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	WLS		Word Length Select	0
		0x0	5-bit character length.	
		0x1	6-bit character length.	
		0x2	7-bit character length.	
		0x3	8-bit character length.	
2	SBS		Stop Bit Select	0
		0	1 stop bit.	
		1	2 stop bits (1.5 if LCR[1:0]=00).	
3	PE		Parity Enable	0

Table 221. USART Line Control Register (LCR - address 0x4000 800C) bit description

Bit	Symbol	Value	Description	Reset Value
		0	Disable parity generation and checking.	
		1	Enable parity generation and checking.	
5:4	PS		Parity Select	0
		0x0	Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.	
		0x1	Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.	
		0x2	Forced 1 stick parity.	
		0x3	Forced 0 stick parity.	
6	BC		Break Control	0
		0	Disable break transmission.	
		1	Enable break transmission. Output pin USART TXD is forced to logic 0 when LCR[6] is active high.	
7	DLAB		Divisor Latch Access Bit	0
		0	Disable access to Divisor Latches.	
		1	Enable access to Divisor Latches.	
31:8	-	-	Reserved	-

12.5.8 USART Modem Control Register

The MCR enables the modem loopback mode and controls the modem output signals.

Table 222. USART Modem Control Register (MCR - address 0x4000 8010) bit description

Bit	Symbol	Value	Description	Reset value
0	DTRCTRL		Source for modem output pin $\overline{\text{DTR}}$. This bit reads as 0 when modem loopback mode is active.	0
1	RTSCTRL		Source for modem output pin $\overline{\text{RTS}}$. This bit reads as 0 when modem loopback mode is active.	0
3:2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
4	LMS		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, RXD, has no effect on loopback and output pin, TXD is held in marking state. The $\overline{\text{DSR}}$, $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, and $\overline{\text{RI}}$ pins are ignored. Externally, $\overline{\text{DTR}}$ and $\overline{\text{RTS}}$ are set inactive. Internally, the upper four bits of the MSR are driven by the lower four bits of the MCR. This permits modem status interrupts to be generated in loopback mode by writing the lower four bits of MCR.	0
		0	Disable modem loopback mode.	
		1	Enable modem loopback mode.	
5	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
6	RTSEN		RTS enable	0

Table 222. USART Modem Control Register (MCR - address 0x4000 8010) bit description

Bit	Symbol	Value	Description	Reset value
		0	Disable auto-rts flow control.	
		1	Enable auto-rts flow control.	
7	CTSEN		CTS enable	0
		0	Disable auto-cts flow control.	
		1	Enable auto-cts flow control.	
31:8	-	-	Reserved	-

12.5.8.1 Auto-flow control

If auto-RTS mode is enabled, the USART's receiver FIFO hardware controls the $\overline{RTS}$ output of the USART. If the auto-CTS mode is enabled, the USART's transmitter will only start sending if the $\overline{CTS}$ pin is low.

12.5.8.1.1 Auto-RTS

The auto-RTS function is enabled by setting the RTSen bit. Auto-RTS data flow control originates in the RBR 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, $\overline{RTS}$ is deasserted (to a high value). It is possible that the sending USART sends an additional byte after the trigger level is reached (assuming the sending USART has another byte to send) because it might not recognize the deassertion of $\overline{RTS}$ until after it has begun sending the additional byte. $\overline{RTS}$ is automatically reasserted (to a low value) once the receiver FIFO has reached the previous trigger level. The reassertion of $\overline{RTS}$ signals the sending USART to continue transmitting data.

If Auto-RTS mode is disabled, the RTSen bit controls the $\overline{RTS}$ output of the USART. If Auto-RTS mode is enabled, hardware controls the $\overline{RTS}$ output, and the actual value of $\overline{RTS}$ will be copied in the RTS Control bit of the USART. As long as Auto-RTS is enabled, the value of the RTS Control bit is read-only for software.

Example: Suppose the USART operating in type '550 mode has the trigger level in FCR set to 0x2, then, if Auto-RTS is enabled, the USART will deassert the $\overline{RTS}$ output as soon as the receive FIFO contains 8 bytes (Table 220 on page 218). The $\overline{RTS}$ output will be reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

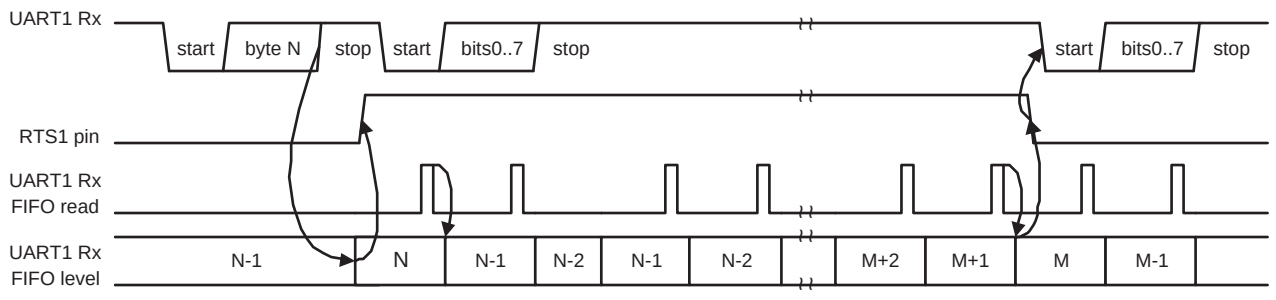


Fig 23. Auto-RTS Functional Timing

12.5.8.1.2 Auto-CTS

The Auto-CTS function is enabled by setting the CTSen bit. If Auto-CTS is enabled, the transmitter circuitry checks the $\overline{CTS}$ input before sending the next data byte. When $\overline{CTS}$ is active (low), the transmitter sends the next byte. To stop the transmitter from sending the following byte, $\overline{CTS}$ must be released before the middle of the last stop bit that is currently being sent. In Auto-CTS mode, a change of the $\overline{CTS}$ signal does not trigger a modem status interrupt unless the CTS Interrupt Enable bit is set, but the Delta CTS bit in the MSR will be set. Table 223 lists the conditions for generating a Modem Status interrupt.

Table 223. Modem status interrupt generation

Enable modem status interrupt (IER[3])	CTSen (MCR[7])	CTS interrupt enable (IER[7])	Delta CTS (MSR[0])	Delta DCD or trailing edge RI or Delta DSR (MSR[3:1])	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 typically eliminates the need for CTS interrupts. When flow control is enabled, a $\overline{CTS}$ 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 24 illustrates the Auto-CTS functional timing.

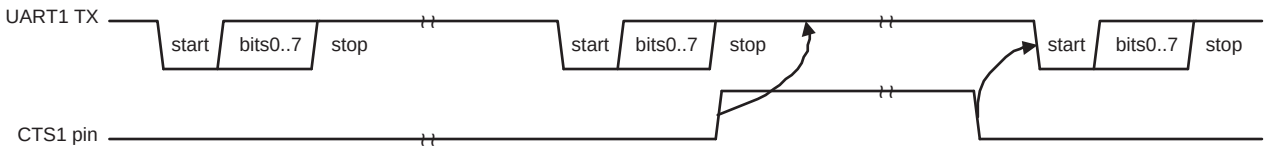


Fig 24. Auto-CTS Functional Timing

During transmission of the second character the $\overline{CTS}$ signal is negated. The third character is not sent thereafter. The USART maintains 1 on TXD as long as $\overline{CTS}$ is negated (high). As soon as $\overline{CTS}$ is asserted, transmission resumes and a start bit is sent followed by the data bits of the next character.

12.5.9 USART Line Status Register (Read-Only)

The LSR is a read-only register that provides status information on the USART TX and RX blocks.

Table 224. USART Line Status Register Read only (LSR - address 0x4000 8014) bit description

Bit	Symbol	Value	Description	Reset Value
0	RDR		Receiver Data Ready:LSR[0] is set when the RBR holds an unread character and is cleared when the USART RBR FIFO is empty.	0
		0	RBR is empty.	
		1	RBR contains valid data.	
1	OE		Overrun Error. The overrun error condition is set as soon as it occurs. A LSR read clears LSR[1]. LSR[1] is set when USART RSR has a new character assembled and the USART RBR FIFO is full. In this case, the USART RBR FIFO will not be overwritten and the character in the USART RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun error status is active.	
2	PE		Parity Error. When the parity bit of a received character is in the wrong state, a parity error occurs. A LSR read clears LSR[2]. Time of parity error detection is dependent on FCR[0]. **Note:** A parity error is associated with the character at the top of the USART RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	
3	FE		Framing Error. When the stop bit of a received character is a logic 0, a framing error occurs. A LSR read clears LSR[3]. The time of the framing error detection is dependent on FCR0. Upon detection of a framing error, the RX will attempt to re-synchronize 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. **Note:** A framing error is associated with the character at the top of the USART RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	BI		Break Interrupt. When RXD1 is held in the spacing state (all zeros) 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). A LSR read clears this status bit. The time of break detection is dependent on FCR[0]. **Note:** The break interrupt is associated with the character at the top of the USART RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	

Table 224. USART Line Status Register Read only (LSR - address 0x4000 8014) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
5	THRE		Transmitter Holding Register Empty. THRE is set immediately upon detection of an empty USART THR and is cleared on a THR write.	1
		0	THR contains valid data.	
		1	THR is empty.	
6	TEMT		Transmitter Empty. TEMT is set when both THR and TSR are empty; TEMT is cleared when either the TSR or the THR contain valid data.	1
		0	THR and/or the TSR contains valid data.	
		1	THR and the TSR are empty.	
7	RXFE		Error in RX FIFO. LSR[7] is set when a character with a RX error such as framing error, parity error or break interrupt, is loaded into the RBR. This bit is cleared when the LSR register is read and there are no subsequent errors in the USART FIFO.	0
		0	RBR contains no USART RX errors or FCR[0]=0.	
		1	USART RBR contains at least one USART RX error.	
8	TXERR		Tx Error. In smart card T=0 operation, this bit is set when the smart card has NACKed a transmitted character, one more than the number of times indicated by the TXRETRY field.	0
31:9	-	-	Reserved	-

12.5.10 USART Modem Status Register

The MSR is a read-only register that provides status information on USART input signals. Bit 0 is cleared when (after) this register is read.

Table 225: USART Modem Status Register (MSR - address 0x4000 8018) bit description

Bit	Symbol	Value	Description	Reset value
0	DCTS		Delta CTS. Set upon state change of input CTS. Cleared on an MSR read.	0
		0	No change detected on modem input, CTS.	
		1	State change detected on modem input, CTS.	
1	DDSR		Delta DSR. Set upon state change of input DSR. Cleared on an MSR read.	0
		0	No change detected on modem input, DSR.	
		1	State change detected on modem input, DSR.	
2	TERI		Trailing Edge RI. Set upon low to high transition of input RI. Cleared on an MSR read.	0
		0	No change detected on modem input, RI.	
		1	Low-to-high transition detected on RI.	

Table 225: USART Modem Status Register (MSR - address 0x4000 8018) bit description

Bit	Symbol	Value	Description	Reset value
3	DDCD		Delta DCD. Set upon state change of input DCD. Cleared on an MSR read.	0
		0	No change detected on modem input, DCD.	
		1	State change detected on modem input, DCD.	
4	CTS	-	Clear To Send State. Complement of input signal CTS. This bit is connected to MCR[1] in modem loopback mode.	0
5	DSR	-	Data Set Ready State. Complement of input signal DSR. This bit is connected to MCR[0] in modem loopback mode.	0
6	RI	-	Ring Indicator State. Complement of input RI. This bit is connected to MCR[2] in modem loopback mode.	0
7	DCD	-	Data Carrier Detect State. Complement of input DCD. This bit is connected to MCR[3] in modem loopback mode.	0
31:8	-	-	Reserved, the value read from a reserved bit is not defined.	NA

12.5.11 USART Scratch Pad Register

The SCR has no effect on the USART 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 SCR has occurred.

Table 226. USART Scratch Pad Register (SCR - address 0x4000 801C) bit description

Bit	Symbol	Description	Reset Value
7:0	PAD	A readable, writable byte.	0x00
31:8	-	Reserved	-

12.5.12 USART Auto-baud Control Register

The USART Auto-baud Control Register (ACR) controls the process of measuring the incoming clock/data rate for baud rate generation, and can be read and written at the user's discretion.

Table 227. Auto-baud Control Register (ACR - address 0x4000 8020) 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		Start mode	0
		0	No restart	

Table 227. Auto-baud Control Register (ACR - address 0x4000 8020) bit description

Bit	Symbol	Value	Description	Reset value
		1	Restart in case of time-out (counter restarts at next USART Rx falling edge)	
7:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	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 IIR.	
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 IIR.	
31:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

12.5.12.1 Auto-baud

The USART 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 DLM and DLL accordingly.

Auto-baud is started by setting the ACR Start bit. Auto-baud can be stopped by clearing the ACR 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 ACR Mode bit. In Mode 0 the baud rate is measured on two subsequent falling edges of the USART 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 USART Rx pin (the length of the start bit).

The ACR 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 USART Rx pin.

The auto-baud function can generate two interrupts.

- The IIR ABTOInt interrupt will get set if the interrupt is enabled (IER ABToIntEn is set and the auto-baud rate measurement counter overflows).
- The IIR ABEOInt interrupt will get set if the interrupt is enabled (IER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding ACR ABTOIntClr and ABEOIntEn bits.

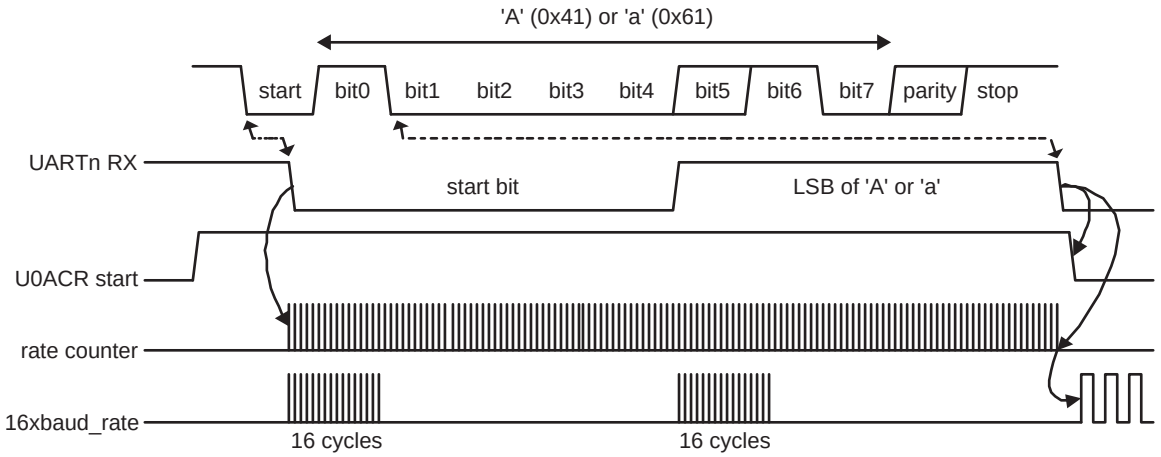
The fractional baud rate generator must be disabled (DIVADDVAL = 0) during auto-baud. Also, when auto-baud is used, any write to DLM and DLL registers should be done before ACR register write. The minimum and the maximum baud rates supported by USART are a function of USART_PCLK and the number of data bits, stop bits and parity bits.

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax \quad (2)$$

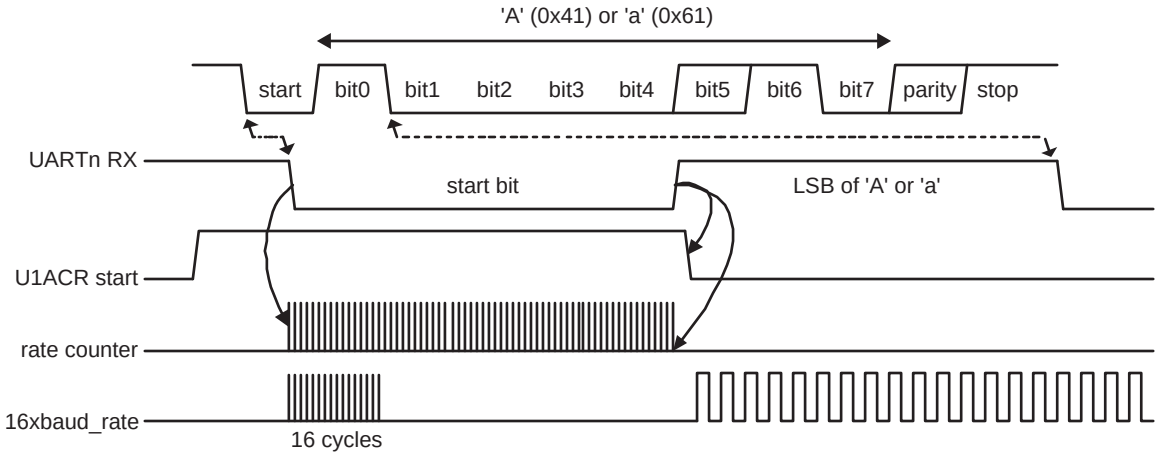
12.5.12.2 Auto-baud modes

When the software is expecting an “AT” command, it configures the USART with the expected character format and sets the ACR Start bit. The initial values in the divisor latches DLM and DLM don‘t care. Because of the “A” or “a” ASCII coding (“A” = 0x41, “a” = 0x61), the USART Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On ACR Start bit setting, the baud rate measurement counter is reset and the USART RSR is reset. The RSR baud rate is switched to the highest rate.
2. A falling edge on USART Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting UART_PCLK cycles.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the USART input clock, guaranteeing the start bit is stored in the RSR.
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 USART input clock (UART_PCLK).
5. If Mode = 0, the rate counter will stop on next falling edge of the USART Rx pin. If Mode = 1, the rate counter will stop on the next rising edge of the USART Rx pin.
6. The rate counter is loaded into DLM/DLL and the baud rate will be switched to normal operation. After setting the DLM/DLL, the end of auto-baud interrupt IIR ABEOInt will be set, if enabled. The RSR will now continue receiving the remaining bits of the character.



a. Mode 0 (start bit and LSB are used for auto-baud)



b. Mode 1 (only start bit is used for auto-baud)

Fig 25. Auto-baud a) mode 0 and b) mode 1 waveform

12.5.13 IrDA Control Register

The IrDA Control Register enables and configures the IrDA mode. The value of the ICR should not be changed while transmitting or receiving data, or data loss or corruption may occur.

Table 228: IrDA Control Register (ICR - 0x4000 8024) bit description

Bit	Symbol	Value	Description	Reset value
0	IRDAEN		IrDA mode enable	0
		0	IrDA mode is disabled.	
		1	IrDA mode is enabled.	
1	IRDAINV		Serial input inverter	0
		0	The serial input is not inverted.	

Table 228: IrDA Control Register (ICR - 0x4000 8024) bit description

Bit	Symbol	Value	Description	Reset value
		1	The serial input is inverted. This has no effect on the serial output.	
2	FIXPULSEEN		IrDA fixed pulse width mode.	0
		0	IrDA fixed pulse width mode disabled.	
		1	IrDA fixed pulse width mode enabled.	
5:3	PULSEDIV		Configures the pulse width when FixPulseEn = 1.	0
		0x0	3 / (16 × baud rate)	
		0x1	$2 \times T_{PCLK}$	
		0x2	$4 \times T_{PCLK}$	
		0x3	$8 \times T_{PCLK}$	
		0x4	$16 \times T_{PCLK}$	
		0x5	$32 \times T_{PCLK}$	
		0x6	$64 \times T_{PCLK}$	
		0x7	$128 \times T_{PCLK}$	
31:6	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

The PulseDiv bits in the ICR 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 229 shows the possible pulse widths.

Table 229: IrDA Pulse Width

FixPulseEn	PulseDiv	IrDA Transmitter Pulse width (µs)
0	x	3 / (16 × baud rate)
1	0	$2 \times T_{PCLK}$
1	1	$4 \times T_{PCLK}$
1	2	$8 \times T_{PCLK}$
1	3	$16 \times T_{PCLK}$
1	4	$32 \times T_{PCLK}$
1	5	$64 \times T_{PCLK}$
1	6	$128 \times T_{PCLK}$
1	7	$256 \times T_{PCLK}$

12.5.14 USART Fractional Divider Register

The USART Fractional Divider Register (FDR) 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 3 or greater.

Table 230. USART Fractional Divider Register (FDR - address 0x4000 8028) bit description

Bit	Function	Description	Reset value
3:0	DIVADDVAL	Baud rate generation pre-scaler divisor value. If this field is 0, fractional baud rate generator will not impact the USART baud rate.	0
7:4	MULVAL	Baud rate pre-scaler multiplier value. This field must be greater or equal 1 for USART to operate properly, regardless of whether the fractional baud rate generator is used or not.	1
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of USART disabled making sure that USART is fully software and hardware compatible with USARTs not equipped with this feature.

The USART baud rate can be calculated as:

$$UART_{baudrate} = \frac{PCLK}{16 \times (256 \times U0DLM + U0DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)} \quad (3)$$

Where UART_PCLK is the peripheral clock, DLM and DLL are the standard USART baud rate divider registers, and DIVADDVAL and MULVAL are USART 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 FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the FDR 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.

12.5.14.1 Baud rate calculation

The USART 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 a set of parameters yields a baud rate with a relative error of less than 1.1% from the desired one.

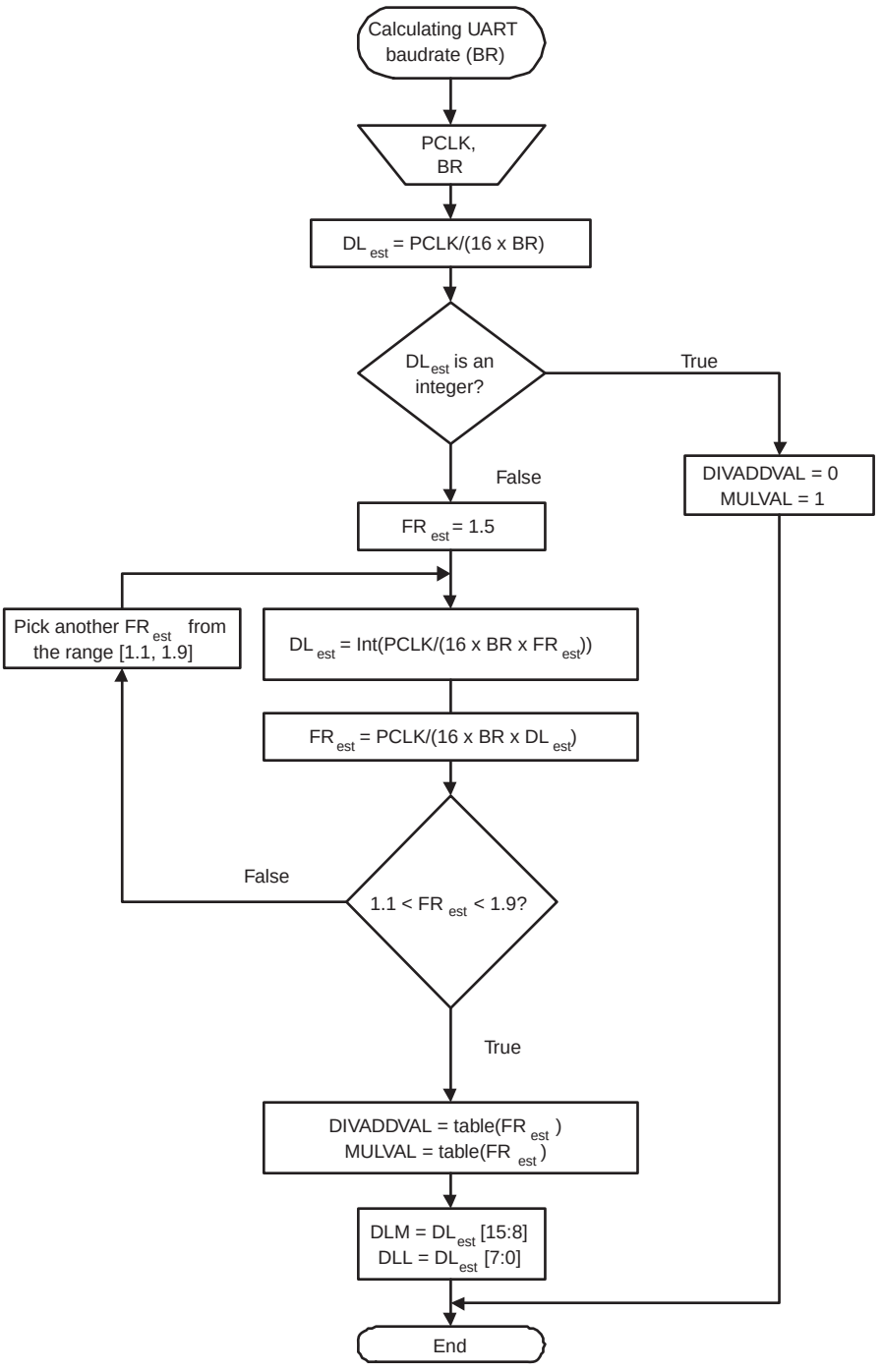


Fig 26. Algorithm for setting USART dividers

Table 231. 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

12.5.14.1.1 Example 1: UART_PCLK = 14.7456 MHz, BR = 9600

According to the provided algorithm DL_{est} = PCLK/(16 x BR) = 14.7456 MHz / (16 x 9600) = 96. Since this DL_{est} is an integer number, DIVADDVAL = 0, MULVAL = 1, DLM = 0, and DLL = 96.

12.5.14.1.2 Example 2: UART_PCLK = 12.0 MHz, BR = 115200

According to the provided algorithm DL_{est} = PCLK/(16 x BR) = 12 MHz / (16 x 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 FRest = 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 FRest = 1.628 in the look-up Table 231 is FR = 1.625. It is equivalent to DIVADDVAL = 5 and MULVAL = 8.

Based on these findings, the suggested USART setup would be: DLM = 0, DLL = 4, DIVADDVAL = 5, and MULVAL = 8. According to Equation 3, the USART's baud rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

12.5.15 USART Oversampling Register

In most applications, the USART 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 232. USART Oversampling Register (OSR - address 0x4000 802C) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Example: For a baud rate of 3.25 Mbps with a 24 MHz USART 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 USART. When Smart card mode is enabled, the initial value of OSINT and FDINT should be programmed as “00101110011” (372 minus one).

12.5.16 USART Transmit Enable Register

In addition to being equipped with full hardware flow control (auto-cts and auto-rts mechanisms described above), TER enables implementation of software flow control. When TxEn = 1, the USART transmitter will keep sending data as long as they are available. As soon as TxEn becomes 0, USART transmission will stop.

Although Table 233 describes how to use TxEn bit in order to achieve hardware flow control, it is strongly suggested to let the USART hardware implemented auto flow control features take care of this and limit the scope of TxEn to software flow control.

Table 233 describes how to use TXEn bit in order to achieve software flow control.

Table 233. USART Transmit Enable Register (TER - address 0x4000 8030) bit description

Bit	Symbol	Description	Reset Value
6:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	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 ($\overline{CTS}$) 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	-

12.5.17 UART Half-duplex enable register

Remark: The HDEN register should be disabled when in smart card mode or IrDA mode (smart card and IrDA by default run in half-duplex mode).

After reset the USART will be in full-duplex mode, meaning that both TX and RX work independently. After setting the HDEN bit, the USART will be in half-duplex mode. In this mode, the USART ensures that the receiver is locked when idle, or will enter a locked state after having received a complete ongoing character reception. Line conflicts must be handled in software. The behavior of the USART is unpredictable when data is presented for reception while data is being transmitted.

For this reason, the value of the HDEN register should not be modified while sending or receiving data, or data may be lost or corrupted.

Table 234. USART Half duplex enable register (HDEN - addresses 0x4000 8040) bit description

Bit	Symbol	Value	Description	Reset value
0	HDEN		Half-duplex mode enable	0
		0	Disable half-duplex mode.	
		1	Enable half-duplex mode.	
31:1	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-

12.5.18 Smart Card Interface Control register

This register allows the USART to be used in asynchronous smart card applications.

Table 235. Smart Card Interface Control register (SCICTRL - address 0x4000 8048) 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	
4:3	-	-	Reserved.	-
7:5	TXRETRY	-	When the protocol selection T bit (above) is 0, the field controls the maximum number of retransmissions that the USART 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 USART 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 (ETUs) by which the guard time after a character transmitted by the USART 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.5.19 USART RS485 Control register

The RS485CTRL register controls the configuration of the USART in RS-485/EIA-485 mode.

Table 236. USART RS485 Control register (RS485CTRL - address 0x4000 804C) bit description

Bit	Symbol	Value	Description	Reset value
0	NMMEN		NMM enable.	0
		0	RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled.	
		1	RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the USART to set the parity error and generate an interrupt.	
1	RXDIS		Receiver enable.	0
		0	The receiver is enabled.	

Table 236. USART RS485 Control register (RS485CTRL - address 0x4000 804C) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
		1	The receiver is disabled.	
2	AADEN		AAD enable.	0
		0	Auto Address Detect (AAD) is disabled.	
		1	Auto Address Detect (AAD) is enabled.	
3	SEL		Select direction control pin	0
		0	If direction control is enabled (bit DCTRL = 1), pin $\overline{RTS}$ is used for direction control.	
		1	If direction control is enabled (bit DCTRL = 1), pin $\overline{DTR}$ is used for direction control.	
4	DCTRL		Auto direction control enable.	0
		0	Disable Auto Direction Control.	
		1	Enable Auto Direction Control.	
5	OINV		Polarity control. This bit reverses the polarity of the direction control signal on the $\overline{RTS}$ (or $\overline{DTR}$) 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.5.20 USART RS-485 Address Match register

The RS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

Table 237. USART RS-485 Address Match register (RS485ADRMATCH - address 0x4000 8050) bit description

Bit	Symbol	Description	Reset value
7:0	ADRMATCH	Contains the address match value.	0x00
31:8	-	Reserved	-

12.5.21 USART RS-485 Delay value register

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 $\overline{RTS}$ (or $\overline{DTR}$). This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

Table 238. USART RS-485 Delay value register (RS485DLY - address 0x4000 8054) 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.	0x00
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.5.22 USART Synchronous mode control register

SYNCCTRL register controls the synchronous mode. When this mode is in effect, the USART generates or receives a bit clock on the SCLK pin and applies it to the transmit and receive shift registers. Synchronous mode should not be used with smart card mode.

Table 239. USART Synchronous mode control register (SYNCCTRL - address 0x4000 8058) 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 high er 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	SSDIS		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

Table 239. USART Synchronous mode control register (SYNCCTRL - address 0x4000 8058) bit description

Bit	Symbol	Value	Description	Reset value
		0	CSCEN is under software control.	
		1	Hardware clears CSCEN after each character is received.	
31:7	-		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 USART operates as follows:

1. The CSRC bit controls whether the USART 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 USART 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 USART sends and samples for start and stop bits as in other modes. When SSDIS is 1, the USART 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 USART samples serial data on RxD on rising edges or falling edges on SCLK. When SSDIS is 1, the USART 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 USART 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 LPC11Uxx USART is the clock source (CSRC=1), such half-duplex operation can lead to the rather artificial-seeming 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 LPC11Uxx USART should receive a character, software should write the SYNCCTRL register with CSCEN=1 and CCCLR=1. After the USART 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.

12.6 Functional description

12.6.1 RS-485/EIA-485 modes of operation

The RS-485/EIA-485 feature allows the USART to be configured as an addressable slave. The addressable slave is one of multiple slaves controlled by a single master.

The USART 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 USART 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

Setting the RS485CTRL bit 0 enables this mode. In this mode, an address is detected when a received byte causes the USART 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 USART 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 an 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 DIR pin as a direction control output signal.

Setting RS485CTRL bit 4 = '1' enables this feature.

Keep RS485CTRL bit 3 zero so that direction control, if enabled, will use the $\overline{\text{RTS}}$ pin.

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 the direction control pin 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 $\overline{\text{RTS}}$. 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 used.

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 RS485CTRL 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.

12.6.2 Smart card mode

Figure 27 shows a typical asynchronous smart card application.

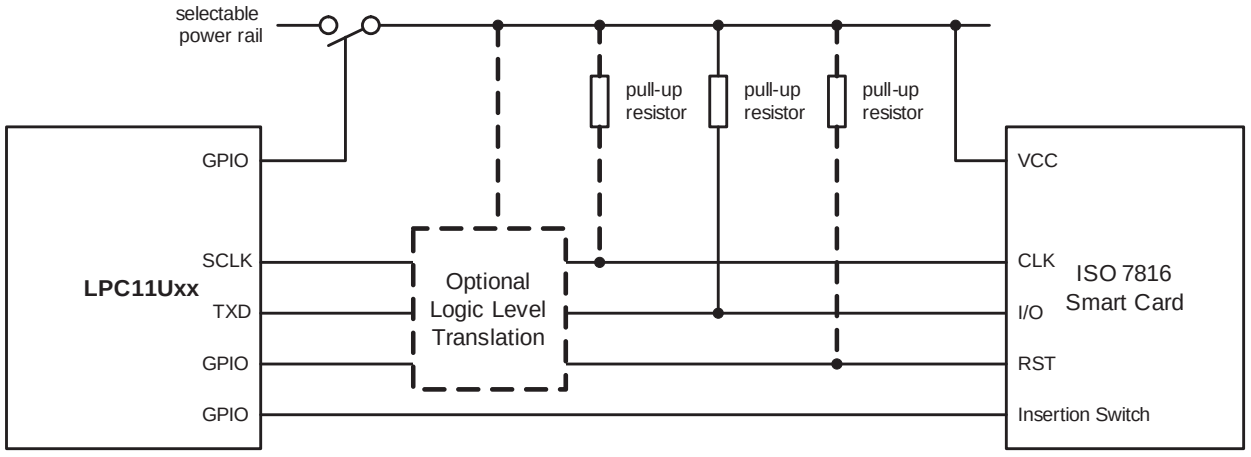


Fig 27. Typical smart card application

When the SCIEN bit in the SCICTRL register (Table 235) is set as described, the USART provides bidirectional serial data on the open-drain TXD pin. No RXD pin is used when SCIEN is 1. The USART SCLK pin will output synchronously with the data at the data bit

rate. Software must use timers to implement character and block waiting times (no hardware support via trigger signals is provided on the LPC11Uxx). GPIO pins can be used to control the smart card reset and power pins. Any power supplied to the card must be externally switched as card power supply requirements often exceed source currents possible on the LPC11Uxx. As the specific application may accommodate any of the available ISO 7816 class A, B, or C power requirements, be aware of the logic level tolerances and requirements when communicating or powering cards that use different power rails than the LPC11Uxx.

12.6.2.1 Smart card set-up procedure

A T = 0 protocol transfer consists of 8-bits of data, an even parity bit, and two guard bits that allow for the receiver of the particular transfer to flag parity errors through the NACK response (see Figure 28). Extra guard bits may be added according to card requirements. If no NACK is sent (provided the interface accepts them in SCICTRL), the next byte may be transmitted immediately after the last guard bit. If the NACK is sent, the transmitter will retry sending the byte until successfully received or until the SCICTRL retry limit has been met.

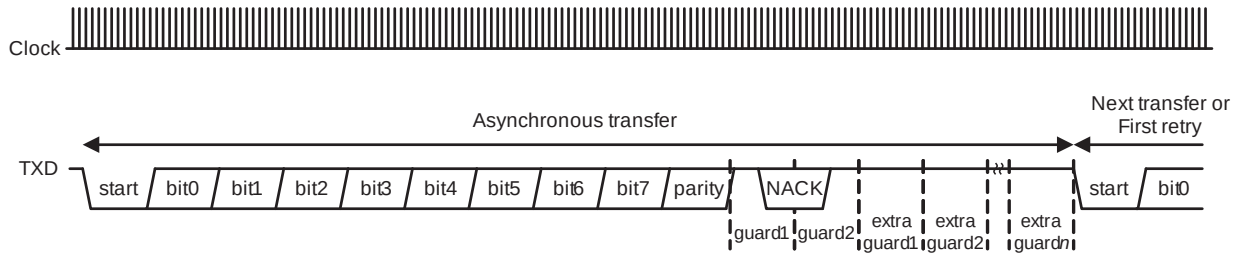


Fig 28. Smart card T = 0 waveform

The smart card must be set up with the following considerations:

- If necessary, program PRESETCTRL (Table 8) so that the USART is not continuously reset.
- Program one IOCON register to enable a USART TXD function.
- If the smart card requires a clock, program one IOCON register to select the USART SCLK function. The USART will use it as an output.
- Program UARTCLKDIV (Table 25) for an initial USART frequency of 3.58 MHz.
- Program the OSR (Section 12.5.15) for 372x oversampling.
- If necessary, program the DLM and DLL (Section 12.5.3) to 00 and 01 respectively, to pass the USART clock through without division.
- Program the LCR (Section 12.5.7) for 8-bit characters, parity enabled, even parity.
- Program the GPIO signals associated with the smart card so that (in this order):
  a. Reset is low.
  b. VCC is provided to the card (GPIO pins do not have the required 200 mA drive).
  c. VPP (if provided to the card) is at “idle” state.
- Program SCICTRL (Section 12.5.18) 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.
- Program SYSAHBCLKCTRL (Table 23) to enable the USART clock.

Thereafter, software should monitor card insertion, handle activation, wait for answer to reset as described in ISO7816-3.

12.7 Architecture

The architecture of the USART is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the USART.

The USART receiver block, RX, monitors the serial input line, RXD, for valid input. The USART RX Shift Register (RSR) accepts valid characters via RXD. After a valid character is assembled in the RSR, it is passed to the USART RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The USART transmitter block, TX, accepts data written by the CPU or host and buffers the data in the USART TX Holding Register FIFO (THR). The USART TX Shift Register (TSR) reads the data stored in the THR and assembles the data to transmit via the serial output pin, TXD1.

The USART Baud Rate Generator block, BRG, generates the timing enables used by the USART TX block. The BRG clock input source is USART_PCLK. The main clock is divided down per the divisor specified in the DLL and DLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

The interrupt interface contains registers IER and IIR. The interrupt interface receives several one clock wide enables from the TX and RX blocks.

Status information from the TX and RX is stored in the LSR. Control information for the TX and RX is stored in the LCR.

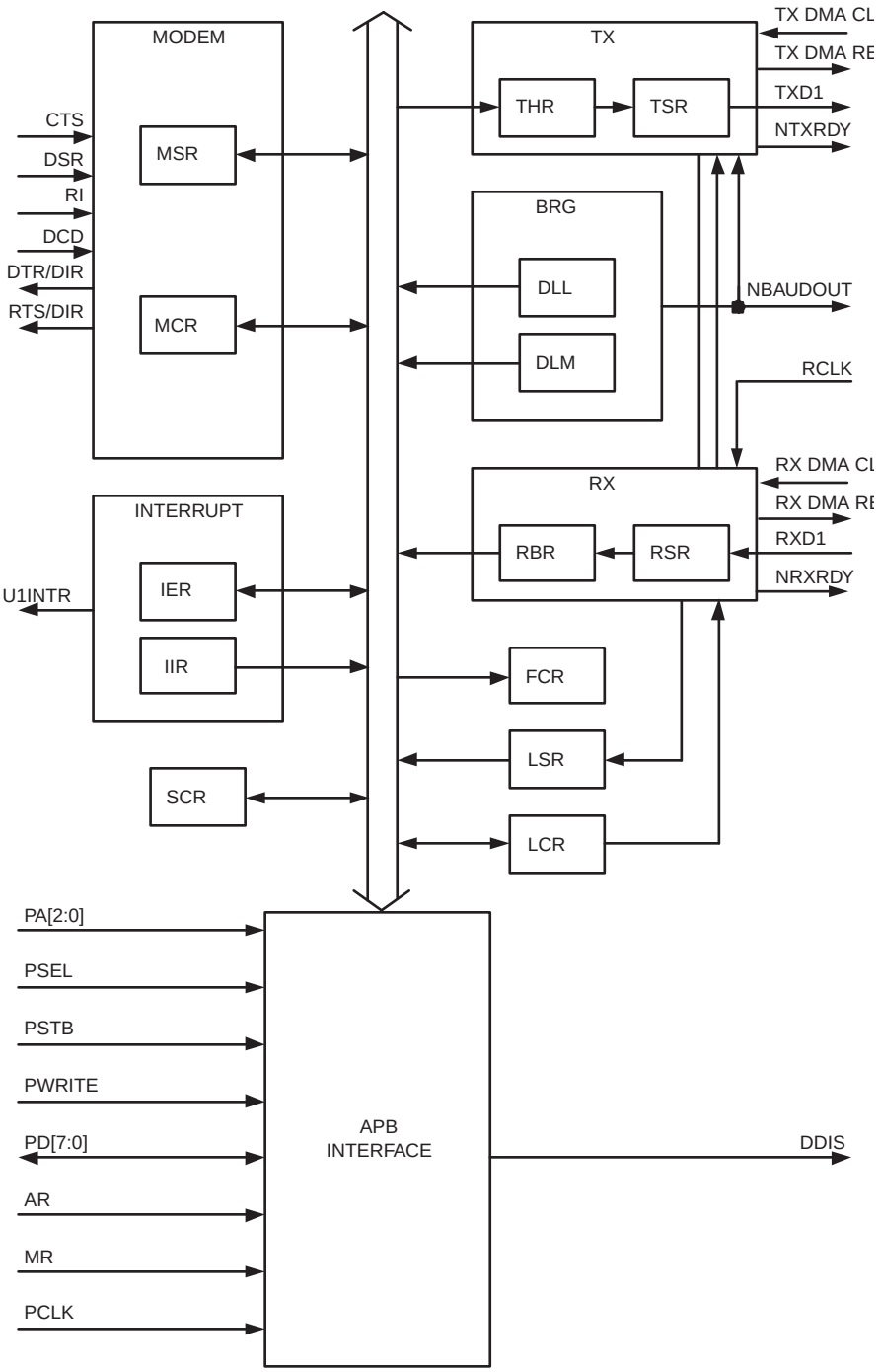


Fig 29. USART block diagram

13.1 How to read this chapter

Two SSP/SPI interfaces are available on all LPC11Uxx parts.

13.2 Basic configuration

The SSP0/1 are configured using the following registers:

1. Pins: The SSP/SPI pins must be configured in the IOCON register block.
2. Power: In the SYSAHBCLKCTRL register, set bit 11 for SSP0 and bit 18 for SSP1 (Table 23).
3. Peripheral clock: Enable the SSP0/SSP1 peripheral clocks by writing to the SSP0/1CLKDIV registers (Table 24/Table 26).
4. Reset: Before accessing the SSP/SPI block, ensure that the SSP0/1_RST_N bits (bit 0 and bit 2) in the PRESETCTRL register (Table 8) are set to 1. This de-asserts the reset signal to the SSP/SPI block.

13.3 Features

- Compatible with Motorola SPI, 4-wire TI SSI, and National Semiconductor Microwire buses.
- Synchronous Serial Communication.
- Supports master or slave operation.
- Eight-frame FIFOs for both transmit and receive.
- 4-bit to 16-bit frame.

13.4 General description

The SSP/SPI 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 bits 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.

13.5 Pin description

Table 240. SSP/SPI pin descriptions

Pin name	Type	Interface pin name/function			Pin description
		SPI	SSI	Microwire	
SCK0/1	I/O	SCK	CLK	SK	**Serial Clock.** 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 SSP/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 SSP/SPI interface either holds it in its inactive state or does not drive it (leaves it in high-impedance state).
SSEL0/1	I/O	SSEL	FS	CS	**Frame Sync/Slave Select.** When the SSP/SPI 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 data has been sent.The active state of this signal can be high or low depending upon the selected bus and mode. When the SSP/SPI interface 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	I/O	MISO	DR(M) DX(S)	SI(M) SO(S)	**Master In Slave Out.** The MISO signal transfers serial data from the slave to the master. When the SSP/SPI is a slave, serial data is output on this signal. When the SSP/SPI is a master, it clocks in serial data from this signal. When the SSP/SPI is a slave and is not selected by FS/SSEL, it does not drive this signal (leaves it in high-impedance state).
MOSI0/1	I/O	MOSI	DX(M) DR(S)	SO(M) SI(S)	**Master Out Slave In.** The MOSI signal transfers serial data from the master to the slave. When the SSP/SPI is a master, it outputs serial data on this signal. When the SSP/SPI is a slave, it clocks in serial data from this signal.

13.6 Register description

The register addresses of the SPI controllers are shown in Table 241.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

Remark: Register names use the SSP prefix to indicate that the SPI controllers have full SSP capabilities.

Table 241. Register overview: SSP/SPI0 (base address 0x4004 0000)

Name	Access	Address offset	Description	Reset value	Reference
CR0	R/W	0x000	Control Register 0. Selects the serial clock rate, bus type, and data size.	0	Table 243
CR1	R/W	0x004	Control Register 1. Selects master/slave and other modes.	0	Table 244
DR	R/W	0x008	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	0	Table 245
SR	RO	0x00C	Status Register	0x0000 0003	Table 246
CPSR	R/W	0x010	Clock Prescale Register	0	Table 247
IMSC	R/W	0x014	Interrupt Mask Set and Clear Register	0	Table 248
RIS	RO	0x018	Raw Interrupt Status Register	0x0000 0008	Table 249
MIS	RO	0x01C	Masked Interrupt Status Register	0	Table 250
ICR	WO	0x020	SSPICR Interrupt Clear Register	NA	Table 251

Table 242. Register overview: SSP/SPI1 (base address 0x4005 8000)

Name	Access	Address offset	Description	Reset value	Reference
CR0	R/W	0x000	Control Register 0. Selects the serial clock rate, bus type, and data size.	0	Table 243
CR1	R/W	0x004	Control Register 1. Selects master/slave and other modes.	0	Table 244
DR	R/W	0x008	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	0	Table 245
SR	RO	0x00C	Status Register	0x0000 0003	Table 246
CPSR	R/W	0x010	Clock Prescale Register	0	Table 247
IMSC	R/W	0x014	Interrupt Mask Set and Clear Register	0	Table 248
RIS	RO	0x018	Raw Interrupt Status Register	0x0000 0008	Table 249
MIS	RO	0x01C	Masked Interrupt Status Register	0	Table 250
ICR	WO	0x020	SSPICR Interrupt Clear Register	NA	Table 251

13.6.1 SSP/SPI Control Register 0

This register controls the basic operation of the SSP/SPI controller.

Table 243. SSP/SPI Control Register 0 (CR0 - address 0x4004 0000 (SSP0) and 0x4005 8000 (SSP1)) 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
		0x3	4-bit transfer	
		0x4	5-bit transfer	
		0x5	6-bit transfer	
		0x6	7-bit transfer	
		0x7	8-bit transfer	
		0x8	9-bit transfer	
		0x9	10-bit transfer	
		0xA	11-bit transfer	
		0xB	12-bit transfer	
		0xC	13-bit transfer	
		0xD	14-bit transfer	
		0xE	15-bit transfer	
		0xF	16-bit transfer	
5:4	FRF		Frame Format.	00
		0x0	SPI	
		0x1	TI	
		0x2	Microwire	
		0x3	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	SPI controller maintains the bus clock low between frames.	
		1	SPI controller maintains the bus clock high between frames.	
7	CPHA		Clock Out Phase. This bit is only used in SPI mode.	0
		0	SPI controller captures serial data on the first clock transition of the frame, that is, the transition **away from** the inter-frame state of the clock line.	
		1	SPI controller captures serial data on the second clock transition of the frame, that is, the transition **back to** 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 × [SCR+1]).	0x00
31:16	-	-	Reserved	-

13.6.2 SSP/SPI Control Register 1

This register controls certain aspects of the operation of the SSP/SPI controller.

Table 244. SSP/SPI Control Register 1 (CR1 - address 0x4004 0004 (SSP0) and 0x4005 8004 (SSP1)) 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		SPI Enable.	0
		0	The SPI controller is disabled.	
		1	The SPI controller will interact with other devices on the serial bus. Software should write the appropriate control information to the other SSP/SPI 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 SPI controller acts as a master on the bus, driving the SCLK, MOSI, and SSEL lines and receiving the MISO line.	
		1	The SPI 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 SPI controller from driving the transmit data line (MISO).	0
31:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.6.3 SSP/SPI Data Register

Software can write data to be transmitted to this register and read data that has been received.

Table 245. SSP/SPI Data Register (DR - address 0x4004 0008 (SSP0) and 0x4005 8008 (SSP1)) bit description

Bit	Symbol	Description	Reset Value
15:0	DATA	**Write:** 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 SPI 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 bit, software must right-justify the data written to this register. **Read:** 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 SPI controller returns data from the least recent frame in the Rx FIFO. If the data length is less than 16 bit, the data is right-justified in this field with higher order bits filled with 0s.	0x0000
31:16	-	Reserved.	-

13.6.4 SSP/SPI Status Register

This read-only register reflects the current status of the SPI controller.

Table 246. SSP/SPI Status Register (SR - address 0x4004 000C (SSP0) and 0x4005 800C (SSP1)) bit description

Bit	Symbol	Description	Reset Value
0	TFE	Transmit FIFO Empty. This bit is 1 is 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 SPI controller is idle, 1 if it is currently sending/receiving a frame and/or the Tx FIFO is not empty.	0
31:5	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.6.5 SSP/SPI Clock Prescale Register

This register controls the factor by which the Prescaler divides the SPI peripheral clock SPI_PCLK to yield the prescaler clock that is, in turn, divided by the SCR factor in the SSPCR0 registers, to determine the bit clock.

Table 247. SSP/SPI Clock Prescale Register (CPSR - address 0x4004 0010 (SSP0) and 0x4005 8010 (SSP1)) bit description

Bit	Symbol	Description	Reset Value
7:0	CPSDVSR	This even value between 2 and 254, by which SPI_PCLK is divided to yield the prescaler output clock. Bit 0 always reads as 0.	0
31:8	-	Reserved.	-

Important: the SSPnCPSR value must be properly initialized, or the SPI controller will not be able to transmit data correctly.

In Slave mode, the SPI clock rate provided by the master must not exceed 1/12 of the SPI peripheral clock selected in Table 24. The content of the SSPnCPSR register is not relevant.

In master mode, $CPSDVSR_{min}$ = 2 or larger (even numbers only).

13.6.6 SSP/SPI Interrupt Mask Set/Clear Register

This register controls whether each of the four possible interrupt conditions in the SPI controller are enabled.

Table 248. SSP/SPI Interrupt Mask Set/Clear register (IMSC - address 0x4004 0014 (SSP0) and 0x4005 8014 (SSP1)) 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 Time-out condition occurs. A Receive Time-out occurs when the Rx FIFO is not empty, and no has not been read for a time-out period. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]).	0
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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.6.7 SSP/SPI Raw Interrupt Status Register

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 IMSC registers.

Table 249. SSP/SPI Raw Interrupt Status register (RIS - address 0x4004 0018 (SSP0) and 0x4005 8018 (SSP1)) bit description

	Symbol	Description	Reset value
0	RORRIS	This bit is 1 if another frame was completely received while the RxFIFO 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 time-out period. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]).	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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.6.8 SSP/SPI Masked Interrupt Status Register

This read-only register contains a 1 for each interrupt condition that is asserted and enabled in the IMSC registers. When an SSP/SPI interrupt occurs, the interrupt service routine should read this register to determine the causes of the interrupt.

Table 250. SSP/SPI Masked Interrupt Status register (MIS - address 0x4004 001C (SSP0) and 0x4005 801C (SSP1)) bit description

Bit	Symbol	Description	Reset value
0	RORMIS	This bit is 1 if another frame was completely received while the RxFIFO 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 time-out period, and this interrupt is enabled. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]).	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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.6.9 SSP/SPI Interrupt Clear Register

Software can write one or more ones to this write-only register, to clear the corresponding interrupt conditions in the SPI 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 SSPIMSC registers.

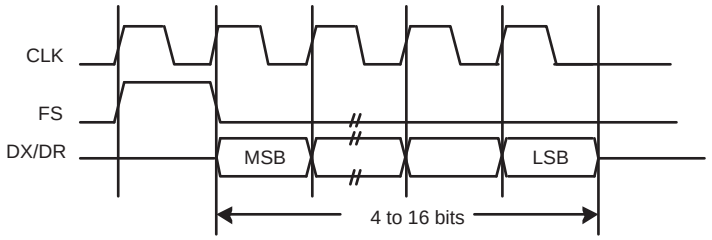
Table 251. SSP/SPI interrupt Clear Register (ICR - address 0x4004 0020 (SSP0) and 0x4005 8020 (SSP1)) bit description

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 timeout period interrupt. The timeout period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]).	NA
31:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

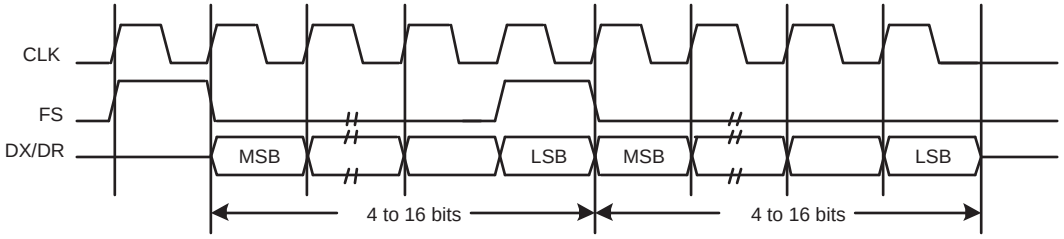
13.7 Functional description

13.7.1 Texas Instruments synchronous serial frame format

Figure 30 shows the 4-wire Texas Instruments synchronous serial frame format supported by the SPI module.



a. Single frame transfer



b. Continuous/back-to-back frames transfer

Fig 30. 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 in 3-state mode 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.

13.7.2 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.

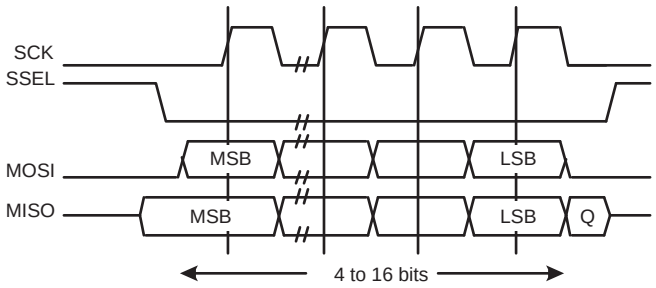
13.7.2.1 Clock Polarity (CPOL) and Phase (CPHA) control

When the CPOL clock polarity control bit is LOW, it produces a steady state low value on the SCK pin. If the CPOL clock polarity control bit is HIGH, a steady state high value is placed on the CLK pin when data is not being transferred.

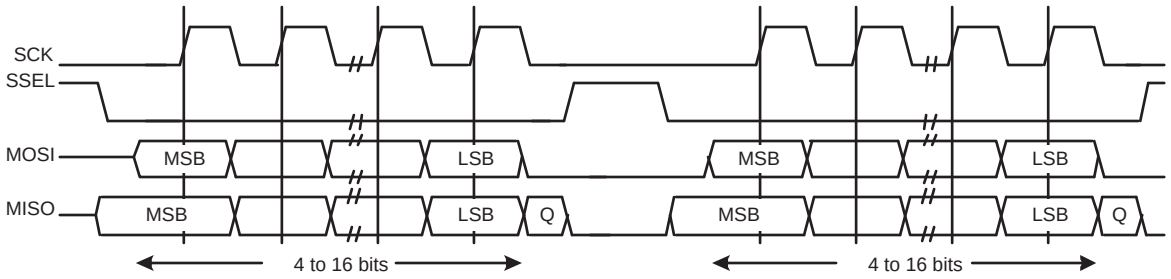
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is LOW, data is captured on the first clock edge transition. If the CPHA clock phase control bit is HIGH, data is captured on the second clock edge transition.

13.7.2.2 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 31.



a. Single transfer with CPOL=0 and CPHA=0



b. Continuous transfer with CPOL=0 and CPHA=0

Fig 31. 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/SPI 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 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.

13.7.2.3 SPI format with CPOL=0,CPHA=1

The transfer signal sequence for SPI format with CPOL = 0, CPHA = 1 is shown in Figure 32, which covers both single and continuous transfers.

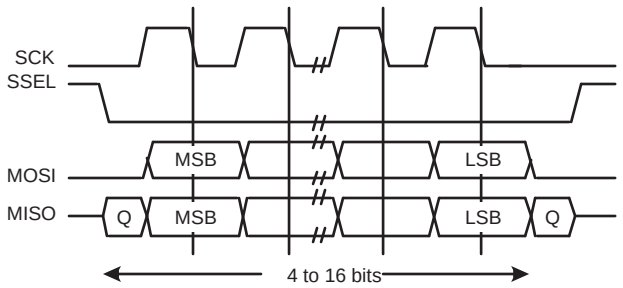


Fig 32. SPI frame format with CPOL=0 and CPHA=1

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/SPI 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.

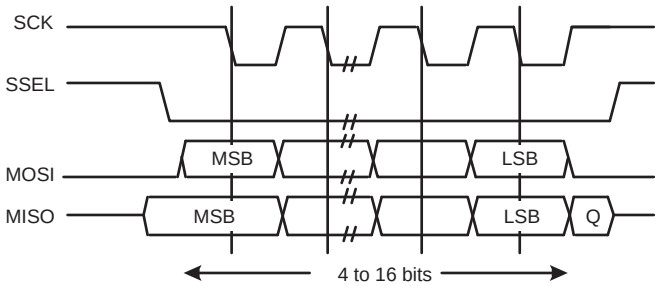
Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

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.

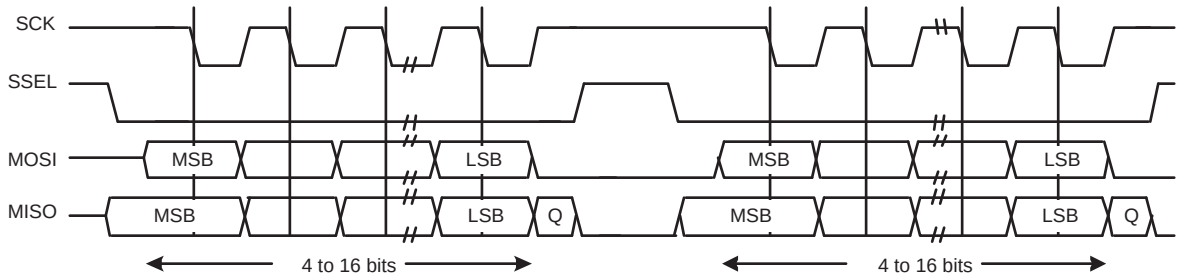
For continuous back-to-back transfers, the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

13.7.2.4 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 33.



a. Single transfer with CPOL=1 and CPHA=0



b. Continuous transfer with CPOL=1 and CPHA=0

Fig 33. SPI frame format with CPOL = 1 and CPHA = 0 (a) Single and b) Continuous Transfer)

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/SPI 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.

13.7.2.5 SPI format with CPOL = 1,CPHA = 1

The transfer signal sequence for SPI format with CPOL = 1, CPHA = 1 is shown in Figure 34, which covers both single and continuous transfers.

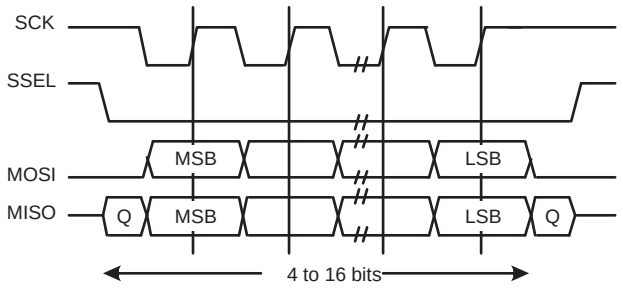


Fig 34. SPI Frame Format with CPOL = 1 and CPHA = 1

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/SPI 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.

13.7.3 Semiconductor Microwire frame format

Figure 35 shows the Microwire frame format for a single frame. Figure 36 shows the same format when back-to-back frames are transmitted.

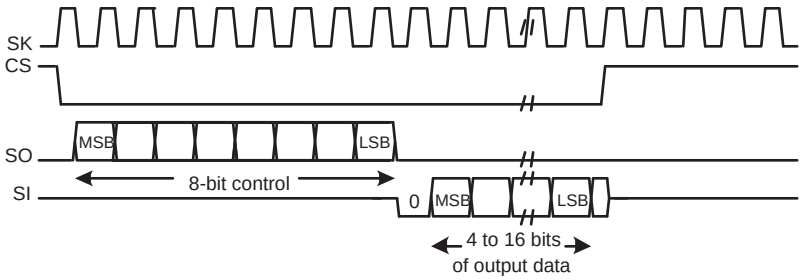


Fig 35. Microwire frame format (single transfer)

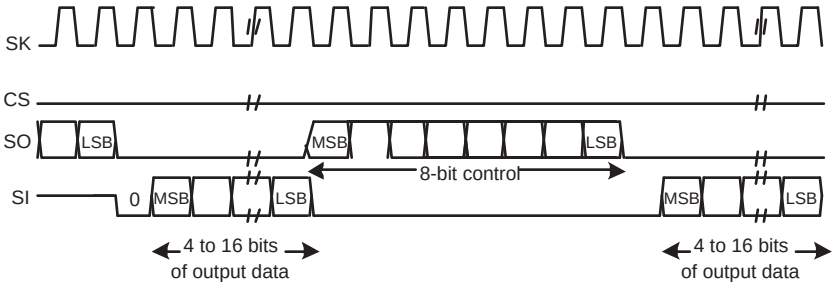


Fig 36. Microwire frame format (continuous transfers)

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/SPI to the off-chip slave device. During this transmission, no incoming data is received by the SSP/SPI. 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 bit 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/SPI. Each bit is driven onto SI line on the falling edge of SK. The SSP/SPI 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 shiftier, 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/SPI.

13.7.3.1 Setup and hold time requirements on CS with respect to SK in Microwire mode

In the Microwire mode, the SSP/SPI 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 37 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/SPI slave, CS must have a setup of at least two times the period of SK on which the SSP/SPI operates. With respect to the SK rising edge previous to this edge, CS must have a hold of at least one SK period.

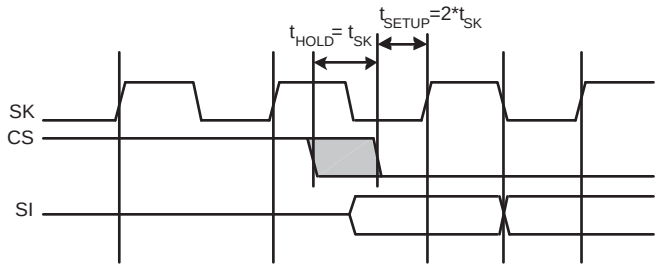


Fig 37. Microwire frame format setup and hold details

14.1 How to read this chapter

The I²C-bus block is identical for all LPC11Uxx parts.

14.2 Basic configuration

The I²C-bus interface is configured using the following registers:

1. Pins: The I2C pin functions and the I2C mode are configured in the IOCON register block (Table 65 and Table 66).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 5 (Table 23).
3. Reset: Before accessing the I2C block, ensure that the I2C_RST_N bit (bit 1) in the PRESETCTRL register (Table 8) is set to 1. This de-asserts the reset signal to the I2C block.

14.3 Features

- Standard I²C-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 I²C 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.
- Supports Fast-mode Plus.
- Optional recognition of up to four distinct slave addresses.
- Monitor mode allows observing all I²C-bus traffic, regardless of slave address.
- I²C-bus can be used for test and diagnostic purposes.
- The I²C-bus contains a standard I²C-compliant bus interface with two pins.

14.4 Applications

Interfaces to external I²C standard parts, such as serial RAMs, LCDs, tone generators, other microcontrollers, etc.

14.5 General description

A typical I²C-bus configuration is shown in Figure 38. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I²C-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.
- 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^2C bus will not be released.

The I^2C interface is byte oriented and has four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

The I^2C interface complies with the entire I^2C specification, supporting the ability to turn power off to the ARM Cortex-M0 without interfering with other devices on the same I^2C-bus.

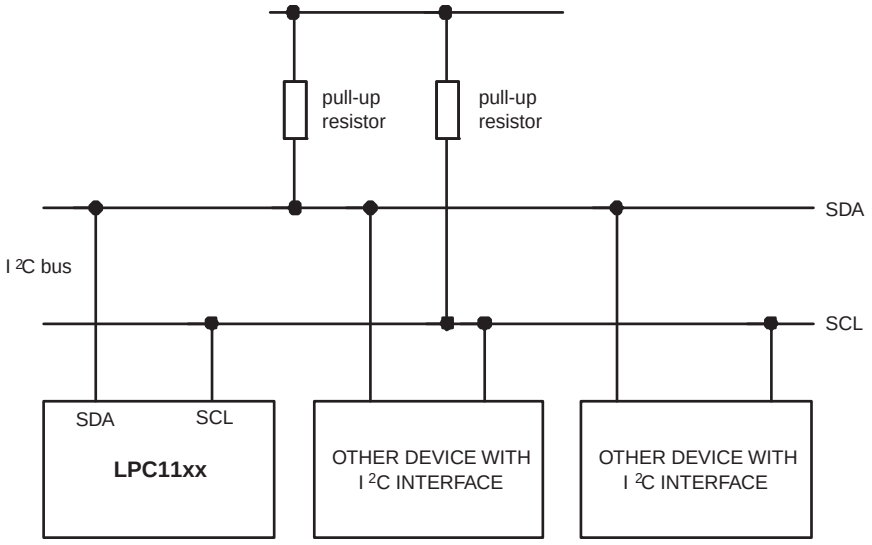


Fig 38. I^2C-bus configuration

14.5.1 I^2C Fast-mode Plus

Fast-Mode Plus supports a 1 Mbit/sec transfer rate to communicate with the I^2C-bus products which NXP Semiconductors is now providing.

14.6 Pin description

Table 252. I²C-bus pin description

Pin	Type	Description
SDA	Input/Output	I²C Serial Data
SCL	Input/Output	I²C Serial Clock

The I²C-bus pins must be configured through the IOCON_PIO0_4 (Table 65) and IOCON_PIO0_5 (Table 66) registers for Standard/ Fast-mode or Fast-mode Plus. In Fast-mode Plus, rates above 400 kHz and up to 1 MHz may be selected. The I²C-bus pins are open-drain outputs and fully compatible with the I²C-bus specification.

14.7 Register description

Table 253. Register overview: I²C (base address 0x4000 0000)

Name	Access	Address offset	Description	Reset value[1]	Reference
CONSET	R/W	0x000	**I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is set. Writing a zero has no effect on the corresponding bit in the I²C control register.	0x00	Table 254
STAT	RO	0x004	**I2C Status Register.** During I²C operation, this register provides detailed status codes that allow software to determine the next action needed.	0xF8	Table 255
DAT	R/W	0x008	**I2C Data Register.** 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.	0x00	Table 256
ADR0	R/W	0x00C	**I2C Slave Address Register 0.** 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.	0x00	Table 257
SCLH	R/W	0x010	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I²C clock.	0x04	Table 258
SCLL	R/W	0x014	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I²C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I²C master and certain times used in slave mode.	0x04	Table 259
CONCLR	WO	0x018	**I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is cleared. Writing a zero has no effect on the corresponding bit in the I²C control register.	NA	Table 261
MMCTRL	R/W	0x01C	**Monitor mode control register.**	0x00	Table 262
ADR1	R/W	0x020	**I2C Slave Address Register 1.** 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.	0x00	Table 263

Table 253. Register overview: I²C (base address 0x4000 0000) *…continued*

Name	Access	Address offset	Description	Reset value[1]	Reference
ADR2	R/W	0x024	**I2C Slave Address Register 2.** 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.	0x00	Table 263
ADR3	R/W	0x028	**I2C Slave Address Register 3.** 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.	0x00	Table 263
DATA_BUFFER	RO	0x02C	**Data buffer register.** The contents of the 8 MSBs of the I2DAT shift register will be transferred to the DATA_BUFFER automatically after every nine bits (8 bits of data plus ACK or NACK) has been received on the bus.	0x00	Table 264
MASK0	R/W	0x030	**I2C Slave address mask register 0**. 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’).	0x00	Table 265
MASK1	R/W	0x034	**I2C Slave address mask register 1**. 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’).	0x00	Table 265
MASK2	R/W	0x038	**I2C Slave address mask register 2**. 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’).	0x00	Table 265
MASK3	R/W	0x03C	**I2C Slave address mask register 3**. 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’).	0x00	Table 265

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

14.7.1 I²C Control Set register (CONSET)

The CONSET registers control setting of bits in the CON 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.

Table 254. I²C Control Set register (CONSET - address 0x4000 0000) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AA	Assert acknowledge flag.	
3	SI	I²C interrupt flag.	0
4	STO	STOP flag.	0

Table 254. I²C Control Set register (CONSET - address 0x4000 0000) bit description

Bit	Symbol	Description	Reset value
5	STA	START flag.	0
6	I2EN	I²C interface enable.	0
31:7	-	Reserved. The value read from a reserved bit is not defined.	-

I2EN I²C Interface Enable. When I2EN is 1, the I²C interface is enabled. I2EN can be cleared by writing 1 to the I2ENC bit in the CONCLR register. When I2EN is 0, the I²C interface is disabled.

When I2EN is “0”, the SDA and SCL input signals are ignored, the I²C 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 I²C-bus since, when I2EN is reset, the I²C-bus status is lost. The AA flag should be used instead.

STA is the START flag. Setting this bit causes the I²C 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 I²C 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 I²C 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 I²C interface is in an addressed slave mode.

STA can be cleared by writing 1 to the STAC bit in the CONCLR 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 I²C-bus if it the interface is in master mode, and transmits a START condition thereafter. If the I²C 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 I²C 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 I²C-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 I²C Interrupt Flag. This bit is set when the I²C 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 the CONCLR register.

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. The address in the Slave Address Register has been received.
2. The General Call address has been received while the General Call bit (GC) in the ADR register is set.
3. A data byte has been received while the I^2C is in the master receiver mode.
4. A data byte has been received while the I^2C is in the addressed slave receiver mode

The AA bit can be cleared by writing 1 to the AAC bit in the CONCLR 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^2C is in the master receiver mode.
2. A data byte has been received while the I^2C is in the addressed slave receiver mode.

14.7.2 I^2C Status register (STAT)

Each I^2C Status register reflects the condition of the corresponding I^2C interface. The I^2C Status register is Read-Only.

Table 255. I^2C Status register (STAT - 0x4000 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^2C interface.	0x1F
31:8	-	Reserved. The value read from a reserved bit is not defined.	-

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^2C 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 270 to Table 275.

14.7.3 I^2C Data register (DAT)

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 DAT register remains stable as long as the SI bit is set. Data in DAT register 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 the DAT register.

Table 256. I^2C Data register (DAT - 0x4000 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. The value read from a reserved bit is not defined.	-

14.7.4 I²C Slave Address register 0 (ADR0)

This register is readable and writable and are only used when an I²C interface is set to slave mode. In master mode, this register has no effect. The LSB of the ADR register is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

If this register contains 0x00, the I²C will not acknowledge any address on the bus. All four registers (ADR0 to ADR3) will be cleared to this disabled state on reset. See also Table 263.

Table 257. I²C Slave Address register 0 (ADR0- 0x4000 000C) bit description

Bit	Symbol	Description	Reset value
0	GC	General Call enable bit.	0
7:1	Address	The I²C device address for slave mode.	0x00
31:8	-	Reserved. The value read from a reserved bit is not defined.	-

14.7.5 I²C SCL HIGH and LOW duty cycle registers (SCLH and SCLL)

Table 258. I²C SCL HIGH Duty Cycle register (SCLH - address 0x4000 0010) bit description

Bit	Symbol	Description	Reset value
15:0	SCLH	Count for SCL HIGH time period selection.	0x0004
31:16	-	Reserved. The value read from a reserved bit is not defined.	-

Table 259. I²C SCL Low duty cycle register (SCLL - 0x4000 0014) bit description

Bit	Symbol	Description	Reset value
15:0	SCLL	Count for SCL low time period selection.	0x0004
31:16	-	Reserved. The value read from a reserved bit is not defined.	-

14.7.5.1 Selecting the appropriate I²C data rate and duty cycle

Software must set values for the registers SCLH and SCLL to select the appropriate data rate and duty cycle. SCLH defines the number of I2C_PCLK cycles for the SCL HIGH time, SCLL defines the number of I2C_PCLK cycles for the SCL low time. The frequency is determined by the following formula (I2C_PCLK is the frequency of the peripheral I2C clock):

(4)

$$I^2C_{bitfrequency} = \frac{I2CPCLK}{SCLH + SCLL}$$

The values for SCLL and SCLH must ensure that the data rate is in the appropriate I²C data rate range. Each register value must be greater than or equal to 4. Table 260 gives some examples of I²C-bus rates based on I2C_PCLK frequency and SCLL and SCLH values.

Table 260. SCLL + SCLH values for selected I²C clock values

I²C mode	I²C bit frequency	I2C_PCLK (MHz)								
		6	8	10	12	16	20	30	40	50
		SCLH + SCLL								
Standard mode	100 kHz	60	80	100	120	160	200	300	400	500
Fast-mode	400 kHz	15	20	25	30	40	50	75	100	125
Fast-mode Plus	1 MHz	-	8	10	12	16	20	30	40	50

SCLL and SCLH values should not necessarily be the same. Software can set different duty cycles on SCL by setting these two registers. For example, the I²C-bus specification defines the SCL low time and high time at different values for a Fast-mode and Fast-mode Plus I²C.

14.7.6 I²C Control Clear register (CONCLR)

The CONCLR register control clearing of bits in the CON 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 cleared. Writing a zero has no effect.

Table 261. I²C Control Clear register (CONCLR - 0x4000 0018) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AAC	Assert acknowledge Clear bit.	
3	SIC	I²C interrupt Clear bit.	0
4	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
5	STAC	START flag Clear bit.	0
6	I2ENC	I²C interface Disable bit.	0
7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
31:8	-	Reserved. The value read from a reserved bit is not defined.	-

AAC is the Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the CONSET register. Writing 0 has no effect.

SIC is the I²C Interrupt Clear bit. Writing a 1 to this bit clears the SI bit in the CONSET 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 CONSET register. Writing 0 has no effect.

I**2ENC** is the I²C Interface Disable bit. Writing a 1 to this bit clears the I2EN bit in the CONSET register. Writing 0 has no effect.

14.7.7 I²C Monitor mode control register (MMCTRL)

This register controls the Monitor mode which allows the I²C module to monitor traffic on the I²C bus without actually participating in traffic or interfering with the I²C bus.

Table 262. I²C Monitor mode control register (MMCTRL - 0x4000 001C) bit description

Bit	Symbol	Value	Description	Reset value
0	MM_ENA		Monitor mode enable.	0
		0	Monitor mode disabled.	
		1	The I²C module will enter monitor mode. In this mode the SDA output will be forced high. This will prevent the I²C module from outputting data of any kind (including ACK) onto the I²C 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 I²C 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 I²C clock line.	
		1	When this bit is set, the I²C 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 I²C module can "stretch" the clock line (hold it low) until it has had time to respond to an I²C interrupt.[1]	
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 described above. 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 I²C 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. The value read from reserved bits is not defined.	

[1] When the ENA_SCL bit is cleared and the I²C no longer has the ability to stall the bus, interrupt response time becomes important. To give the part more time to respond to an I²C interrupt under these conditions, a DATA _BUFFER register is used (Section 14.7.9) 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).

14.7.7.1 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 "thinks" 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.

14.7.7.2 Loss of arbitration in Monitor mode

In monitor mode, the I2C 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. This will most probably result in a lost-arbitration state as far as our module is concerned.

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. In addition, hardware may be designed into the module to block some/all loss of arbitration states from occurring if those state would either prevent a desired interrupt from occurring or cause an unwanted interrupt to occur. Whether any such hardware will be added is still to be determined.

14.7.8 I2C Slave Address registers (ADR[1, 2, 3])

These registers are readable and writable and are only used when an I2C interface is set to slave mode. In master mode, this register has no effect. The LSB of the ADR register is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

If these registers contain 0x00, the I2C will not acknowledge any address on the bus. All four registers will be cleared to this disabled state on reset (also see Table 257).

Table 263. I2C Slave Address registers (ADR[1, 2, 3]- 0x4000 00[20, 24, 28]) bit description

Bit	Symbol	Description	Reset value
0	GC	General Call enable bit.	0
7:1	Address	The I2C device address for slave mode.	0x00
31:8	-	Reserved. The value read from a reserved bit is not defined.	0

14.7.9 I2C Data buffer register (DATA_BUFFER)

In monitor mode, the I2C module may lose the ability to stretch the clock (stall the bus) 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 DAT 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 DATA_BUFFER register will be added. The contents of the 8 MSBs of the DAT shift register will be transferred to the DATA_BUFFER automatically after every nine bits (8 bits of data plus ACK or NACK) has been received on the bus. This means that the processor will have nine 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 the DAT register directly, as usual, and the behavior of DAT will not be altered in any way.

Although the DATA_BUFFER register is primarily intended for use in monitor mode with the ENA_SCL bit = ‘0’, it will be available for reading at any time under any mode of operation.

Table 264. I²C Data buffer register (DATA_BUFFER - 0x4000 002C) bit description

Bit	Symbol	Description	Reset value
7:0	Data	This register holds contents of the 8 MSBs of the DAT shift register.	0
31:8	-	Reserved. The value read from a reserved bit is not defined.	0

14.7.10 I²C Mask registers (MASK[0, 1, 2, 3])

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 ADRn register associated with that mask register. In other words, bits in an ADRn register which are masked are not taken into account in determining an address match.

On reset, all mask register bits are cleared to '0'.

The mask register has no effect on comparison to the General Call address ("0000000").

Bits(31:8) and bit(0) of the mask registers are unused and should not be written to. These bits will always read back as zeros.

When an address-match interrupt occurs, the processor will have to read the data register (DAT) to determine what the received address was that actually caused the match.

Table 265. I²C Mask registers (MASK[0, 1, 2, 3] - 0x4000 00[30, 34, 38, 3C]) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved. User software should not write ones to reserved bits. This bit reads always back as 0.	0
7:1	MASK	Mask bits.	0x00
31:8	-	Reserved. The value read from reserved bits is undefined.	0

14.8 Functional description

Figure 39 shows how the on-chip I²C-bus interface is implemented, and the following text describes the individual blocks.

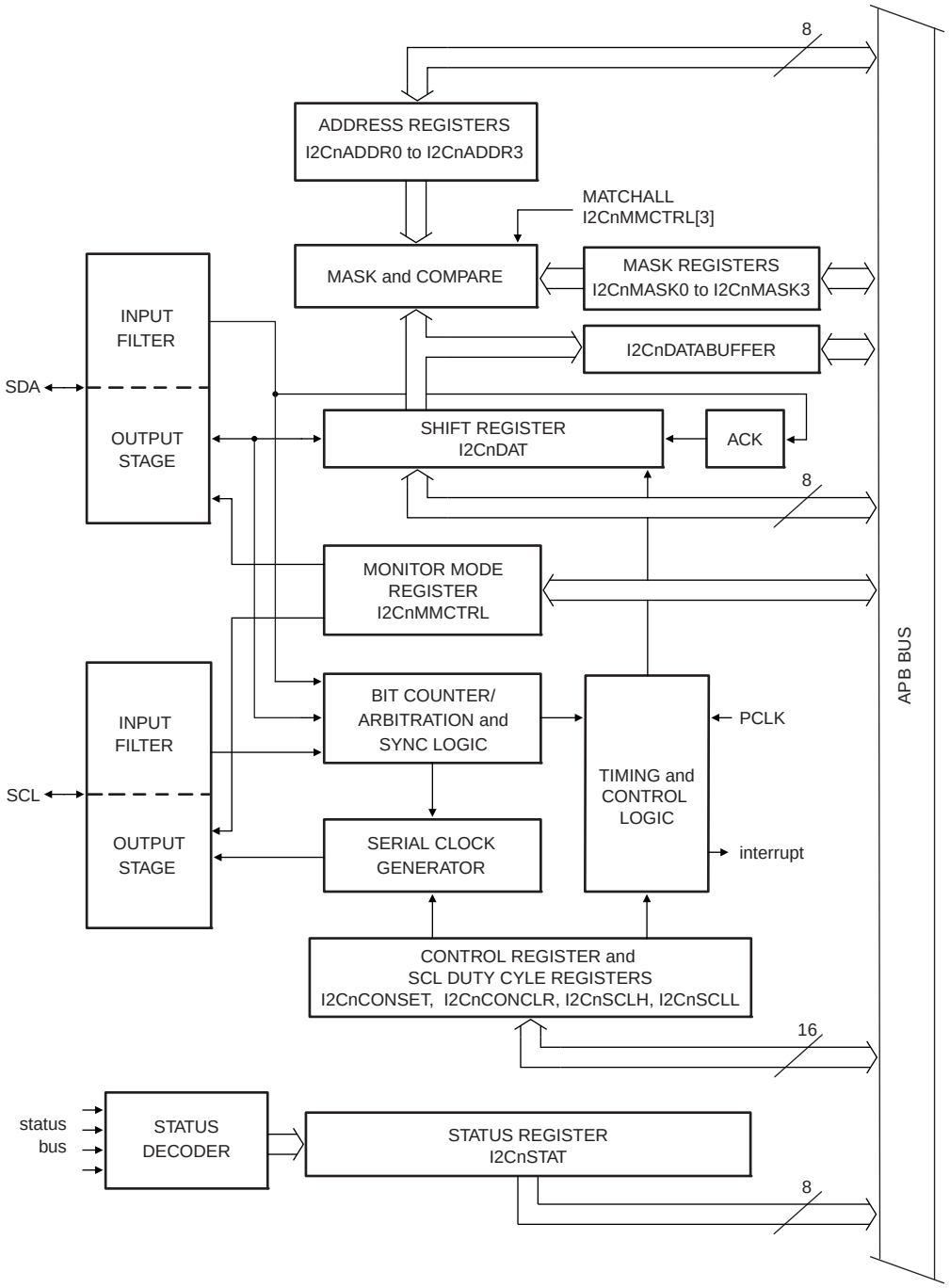


Fig 39. I²C serial interface block diagram

14.8.1 Input filters and output stages

Input signals are synchronized with the internal clock, and spikes shorter than three clocks are filtered out.

The output for I²C is a special pad designed to conform to the I²C specification.

14.8.2 Address Registers, ADR0 to ADR3

These registers may be loaded with the 7-bit slave address (7 most significant bits) to which the I^2C 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 DAT register at the state where the own slave address has been received.

14.8.3 Address mask registers, MASK0 to MASK3

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 ADRn register associated with that mask register. In other words, bits in an ADRn 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 (DAT) to determine what the received address was that actually caused the match.

14.8.4 Comparator

The comparator compares the received 7-bit slave address with its own slave address (7 most significant bits in ADR). It also compares the first received 8-bit byte with the General Call address (0x00). If an equality is found, the appropriate status bits are set and an interrupt is requested.

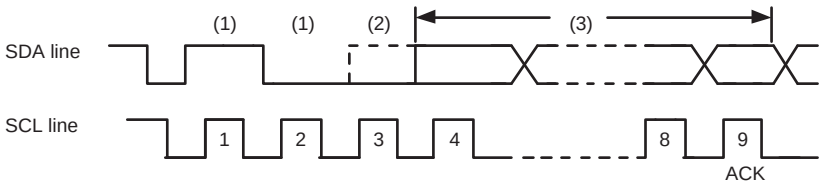
14.8.5 Shift register, DAT

This 8-bit register contains a byte of serial data to be transmitted or a byte which has just been received. Data in DAT 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 DAT. While data is being shifted out, data on the bus is simultaneously being shifted in; DAT 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 DAT.

14.8.6 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^2C-bus. If another device on the bus overrules a logic 1 and pulls the SDA line low, arbitration is lost, and the I^2C block immediately changes from master transmitter to slave receiver. The I^2C 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^2C 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^2C block generates no further clock pulses. Figure 40 shows the arbitration procedure.



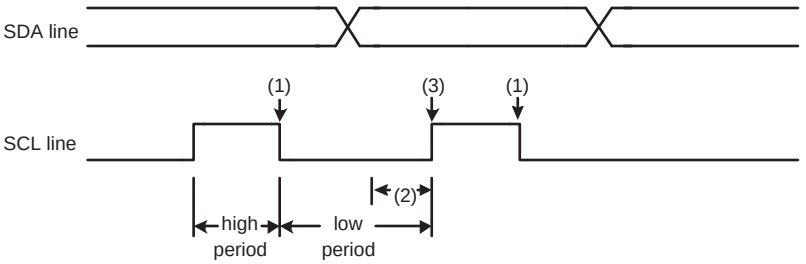
(1) Another device transmits serial data.

(2) Another device overrules a logic (dotted line) transmitted this I2C master by pulling the SDA line low. Arbitration is lost, and this I2C enters Slave Receiver mode.

(3) This I2C is in Slave Receiver mode but still generates clock pulses until the current byte has been transmitted. This I2C will not generate clock pulses for the next byte. Data on SDA originates from the new master once it has won arbitration.

Fig 40. 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 41 shows the synchronization procedure.



(1) Another device pulls the SCL line low before this I2C has timed a complete high time. The other device effectively determines the (shorter) HIGH period.

(2) Another device continues to pull the SCL line low after this I2C has timed a complete low time and released SCL. The I2C clock generator is forced to wait until SCL goes HIGH. The other device effectively determines the (longer) LOW period.

(3) The SCL line is released , and the clock generator begins timing the HIGH time.

Fig 41. Serial clock synchronization

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 I2C 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.

14.8.7 Serial clock generator

This programmable clock pulse generator provides the SCL clock pulses when the I2C block is in the master transmitter or master receiver mode. It is switched off when the I2C block is in slave mode. The I2C output clock frequency and duty cycle is programmable

via the I2C 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.

14.8.8 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 DAT, 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 I2C-bus status.

14.8.9 Control register, CONSET and CONCLR

The I2C control register contains bits used to control the following I2C block functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.

The contents of the I2C control register may be read as CONSET. Writing to CONSET will set bits in the I2C control register that correspond to ones in the value written. Conversely, writing to CONCLR will clear bits in the I2C control register that correspond to ones in the value written.

14.8.10 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 I2C-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 I2C 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).

14.9 I2C operating modes

In a given application, the I2C block may operate as a master, a slave, or both. In the slave mode, the I2C 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 I2C block switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

14.9.1 Master Transmitter mode

In this mode data is transmitted from master to slave. Before the master transmitter mode can be entered, the CONSET register must be initialized as shown in Table 266. I2EN must be set to 1 to enable the I2C function. If the AA bit is 0, the I2C 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 CONCLR register. THe STA bit should be cleared after writing the slave address.

Table 266. CONSET 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^2C interface will enter master transmitter mode when software sets the STA bit. The I^2C 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 STAT 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 DAT register, and then clear the SI bit. SI is cleared by writing a 1 to the SIC bit in the CONCLR 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 270 to Table 275.

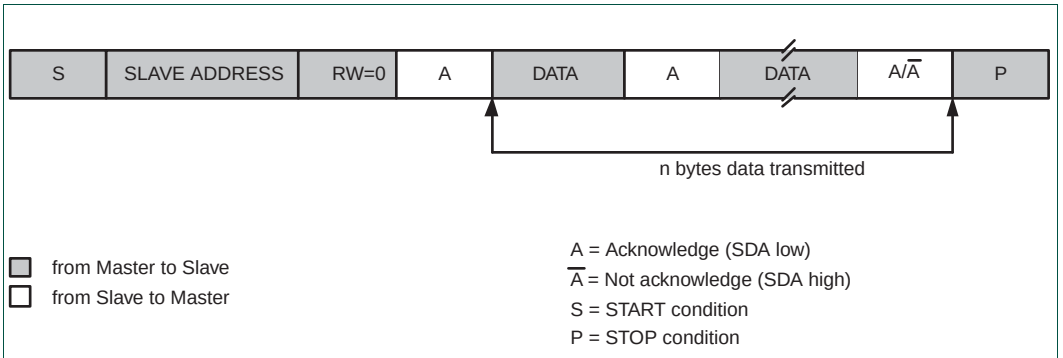


Fig 42. Format in the Master Transmitter mode

14.9.2 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^2C Data register (DAT), 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 271.

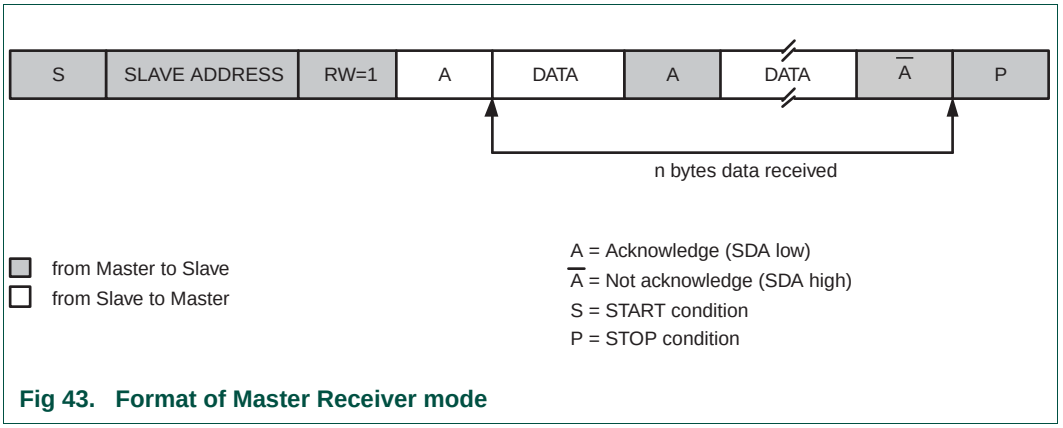


Fig 43. Format of Master Receiver mode

After a Repeated START condition, I^2C may switch to the master transmitter mode.

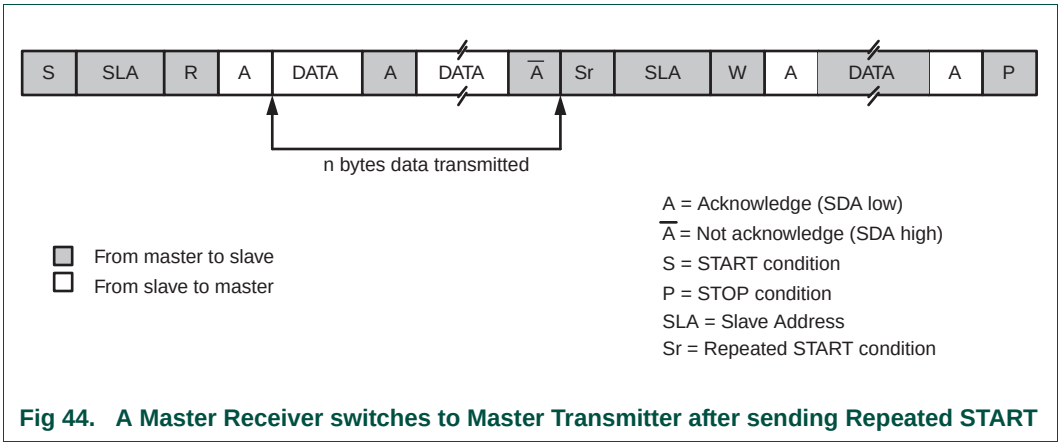


Fig 44. A Master Receiver switches to Master Transmitter after sending Repeated START

14.9.3 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 (ADR0-3) and write the I^2C Control Set register (CONSET) as shown in Table 267.

Table 267. CONSET 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^2C function. AA bit must be set to 1 to acknowledge its own slave address or the General Call address. The STA, STO and SI bits are set to 0.

After ADR and CONSET are initialized, the I^2C interface waits until it is addressed by its own address or general 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 (STAT). Refer to Table 274 for the status codes and actions.

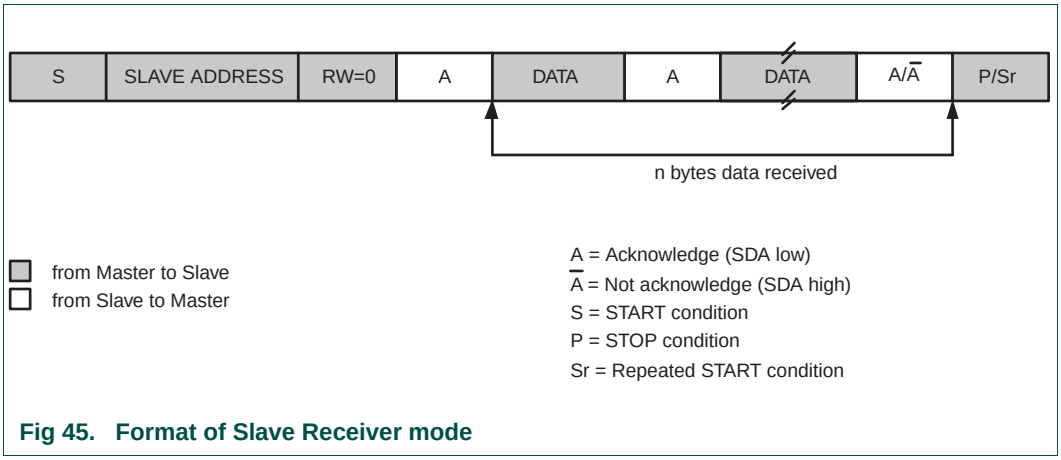


Fig 45. Format of Slave Receiver mode

14.9.4 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^2C may operate as a master and as a slave. In the slave mode, the I^2C hardware looks for its own slave address 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^2C interface switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

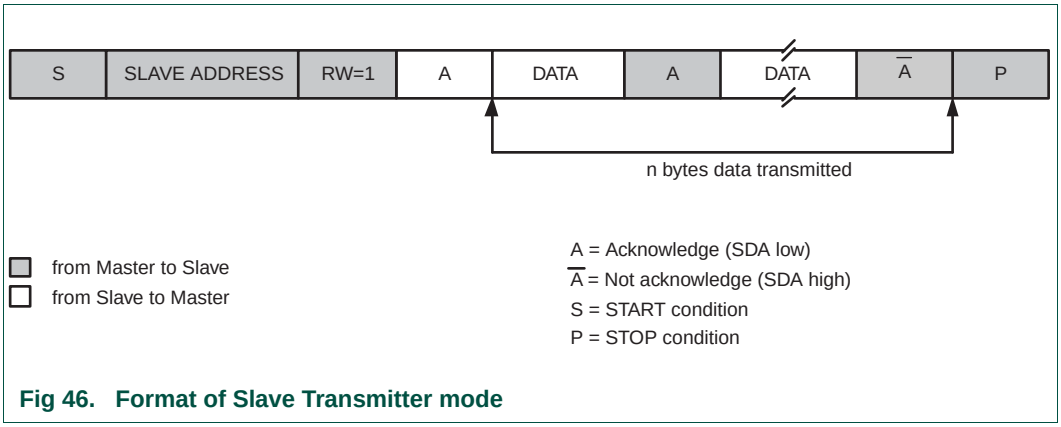


Fig 46. Format of Slave Transmitter mode

14.10 Details of I^2C 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 47, Figure 48, Figure 49, Figure 50, and Figure 51. Table 268 lists abbreviations used in these figures when describing the I^2C operating modes.

Table 268. Abbreviations used to describe an I^2C 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)
$\overline{A}$	Not acknowledge bit (HIGH level at SDA)
Data	8-bit data byte
P	STOP condition

In Figure 47 to Figure 51, circles are used to indicate when the serial interrupt flag is set. The numbers in the circles show the status code held in the STAT 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 STAT 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 270 to Table 276.

14.10.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 47). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

Table 269. CONSET 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^2C rate must also be configured in the SCLL and SCLH registers. I2EN must be set to logic 1 to enable the I^2C block. If the AA bit is reset, the I^2C 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^2C interface cannot enter slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The I^2C logic will now test the I^2C-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 (STAT) will be 0x08. This status code is used by the interrupt service routine to enter the appropriate state service routine that loads DAT with the slave address and the data direction bit (SLA+W). The SI bit in CON 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 STAT 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 270. After a Repeated START condition (state 0x10). The I^2C block may switch to the master receiver mode by loading DAT with SLA+R).

Table 270. Master Transmitter mode

Status Code (I2CSTAT)	Status of the I^2C-bus and hardware	Application software response					Next action taken by I^2C hardware
		To/From DAT	To CON				
			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+R will be transmitted; the I^2C 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 DAT action or	1	0	0	X	Repeated START will be transmitted.
		No DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT 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 DAT action or	1	0	0	X	Repeated START will be transmitted.
		No DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x28	Data byte in DAT 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 DAT action or	1	0	0	X	Repeated START will be transmitted.
		No DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x30	Data byte in DAT 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 DAT action or	1	0	0	X	Repeated START will be transmitted.
		No DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT 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 DAT action or	0	0	0	X	I^2C-bus will be released; not addressed slave will be entered.
		No DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free.

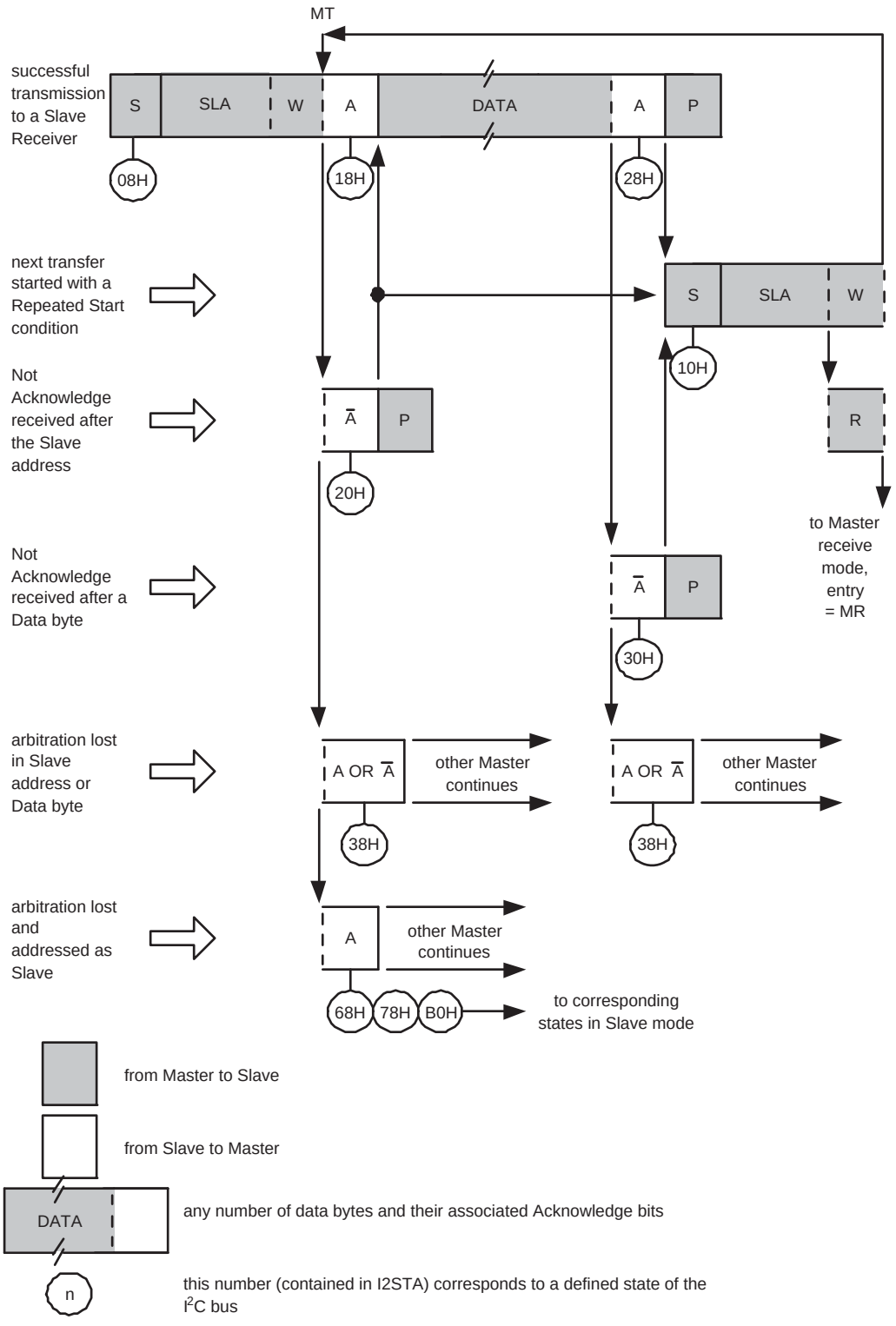


Fig 47. Format and states in the Master Transmitter mode

14.10.2 Master Receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see Figure 48). The transfer is initialized as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load DAT with the 7-bit slave address and the data direction bit (SLA+R). The SI bit in CON 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 STAT 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 271. After a Repeated START condition (state 0x10), the I^2C block may switch to the master transmitter mode by loading DAT with SLA+W.

Table 271. Master Receiver mode

Status Code (STAT)	Status of the I^2C-bus and hardware	Application software response					Next action taken by I^2C hardware
		To/From DAT	To CON				
			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^2C block will be switched to MST/TRX mode.
0x38	Arbitration lost in NOT ACK bit.	No DAT action or	0	0	0	X	I^2C-bus will be released; the I^2C block will enter slave mode.
		No DAT 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 DAT action or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		No DAT 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 DAT action or	1	0	0	X	Repeated START condition will be transmitted.
		No DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No DAT 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.

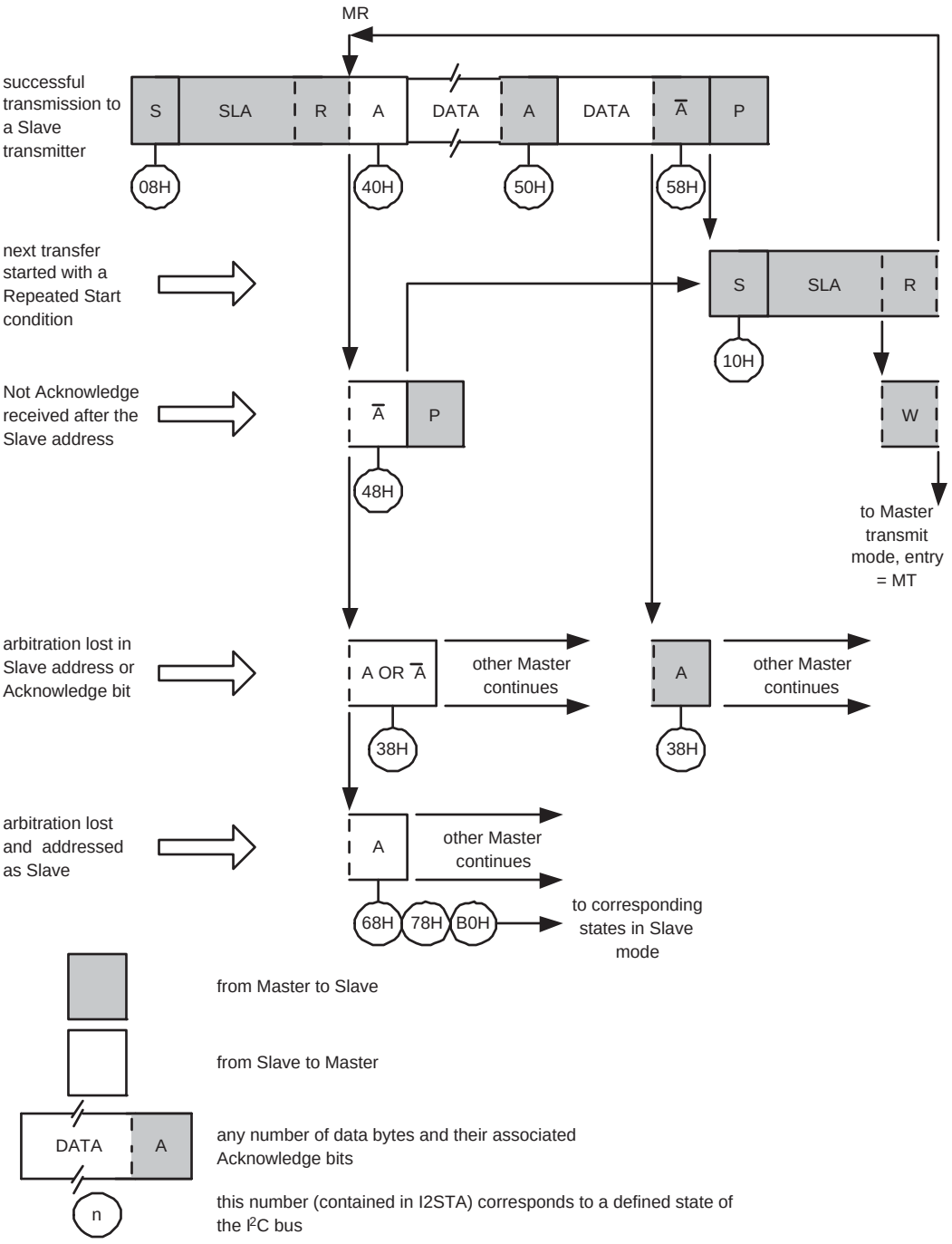


Fig 48. Format and states in the Master Receiver mode

14.10.3 Slave Receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 49). To initiate the slave receiver mode, ADR and CON must be loaded as follows:

Table 272. ADR usage in Slave Receiver mode

Bit	7	6	5	4	3	2	1	0
Symbol	own slave 7-bit address							GC

The upper 7 bits are the address to which the I^2C block will respond when addressed by a master. If the LSB (GC) is set, the I^2C block will respond to the General Call address (0x00); otherwise it ignores the General Call address.

Table 273. CONSET 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^2C-bus rate settings do not affect the I^2C block in the slave mode. I2EN must be set to logic 1 to enable the I^2C block. The AA bit must be set to enable the I^2C block to acknowledge its own slave address or the General Call address. STA, STO, and SI must be reset.

When ADR and CON have been initialized, the I^2C 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^2C 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 STAT. 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 274. The slave receiver mode may also be entered if arbitration is lost while the I^2C block is in the master mode (see status 0x68 and 0x78).

If the AA bit is reset during a transfer, the I^2C block will return a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the I^2C block does not respond to its own slave address or a General Call address. However, the I^2C-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^2C block from the I^2C-bus.

Table 274. Slave Receiver mode

Status Code (STAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From DAT	To CON				
			STA	STO	SI	AA	
0x60	Own SLA+W has been received; ACK has been returned.	No DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT 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 DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT 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 DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT 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 DAT action or	X	0	0	0	Data byte will be received and NOT ACK will be returned.
		No DAT action	X	0	0	1	Data byte will be received and ACK will be returned.
0x80	Previously addressed with own SLV 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 ADR[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 ADR[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 274. Slave Receiver mode *…continued*

Status Code (STAT)	Status of the I^2C-bus and hardware	Application software response					Next action taken by I^2C hardware
		To/From DAT	To CON				
			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 ADR[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 ADR[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 SLV/REC or SLV/TRX.	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 ADR[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 ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.

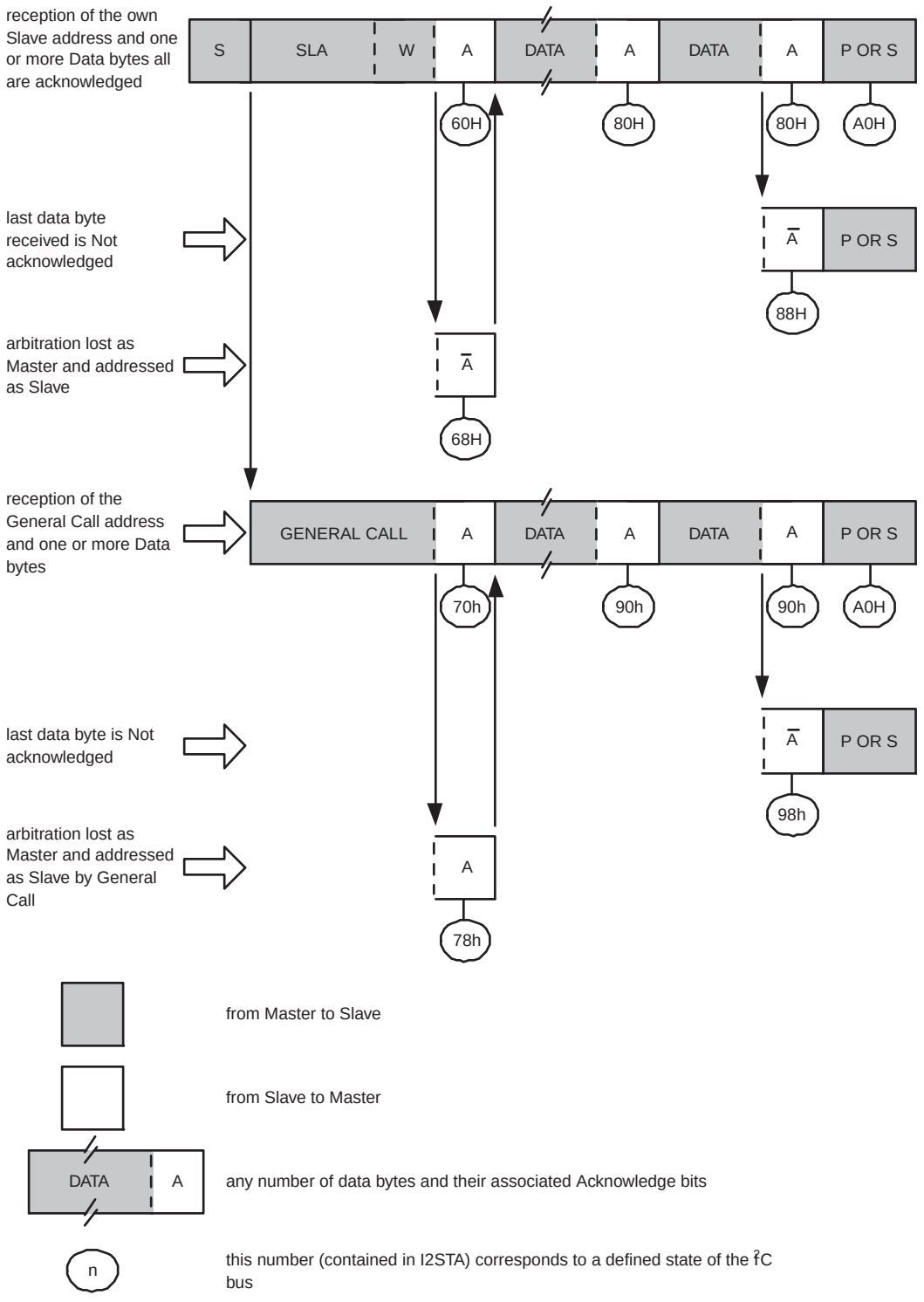


Fig 49. Format and states in the Slave Receiver mode

14.10.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 50). Data transfer is initialized as in the slave receiver mode. When ADR and CON have been initialized, the I^2C 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^2C 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 STAT. 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 275. The slave transmitter mode may also be entered if arbitration is lost while the I^2C block is in the master mode (see state 0xB0).

If the AA bit is reset during a transfer, the I^2C block will transmit the last byte of the transfer and enter state 0xC0 or 0xC8. The I^2C 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^2C block does not respond to its own slave address or a General Call address. However, the I^2C-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^2C block from the I^2C-bus.

Table 275. Slave Transmitter mode

Status Code (STAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From DAT	To CON				
			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 DAT 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 DAT has been transmitted; NOT ACK has been received.	No DAT action or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General call address.
		No DAT action or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1.
		No DAT 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 DAT action	1	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free.
0xC8	Last data byte in DAT has been transmitted (AA = 0); ACK has been received.	No DAT action or	0	0	0	0	Switched to not addressed SLV mode; no recognition of own SLA or General call address.
		No DAT action or	0	0	0	1	Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1.
		No DAT 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 DAT action	1	0	0	01	Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR.0 = logic 1. A START condition will be transmitted when the bus becomes free.

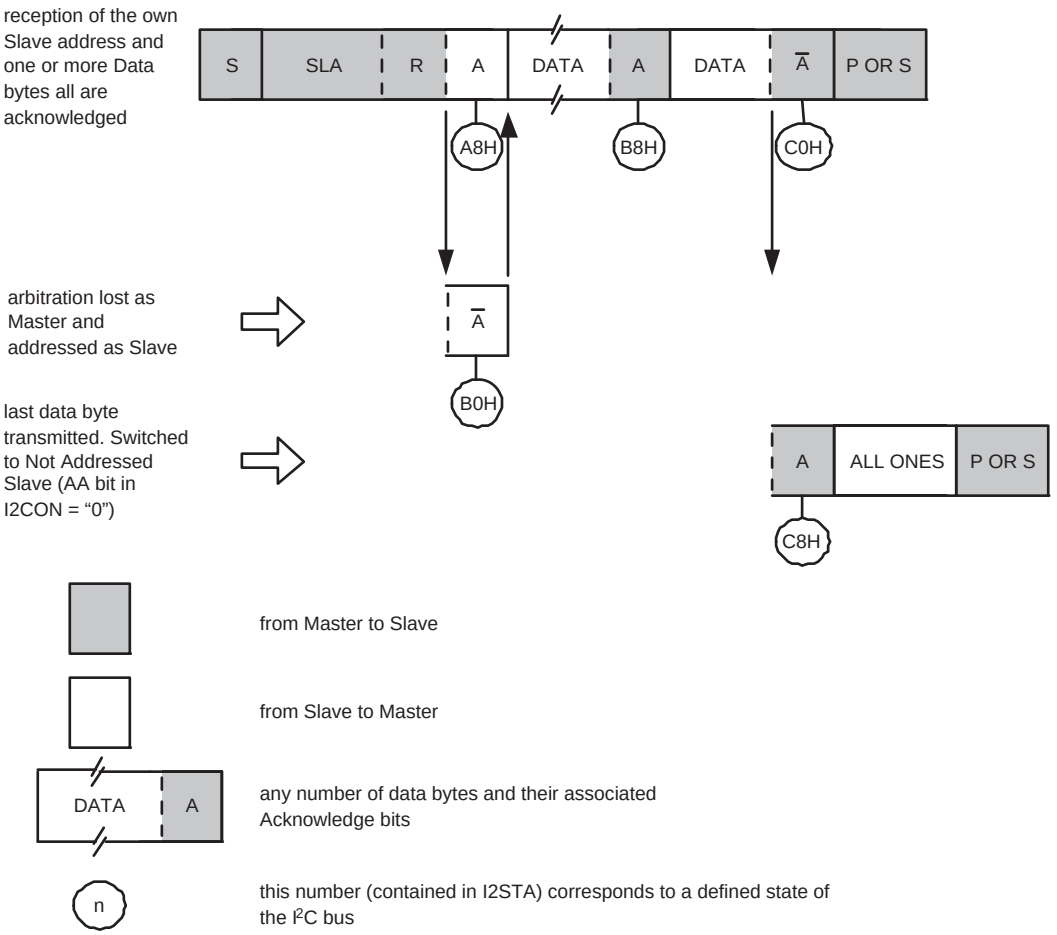


Fig 50. Format and states in the Slave Transmitter mode

14.10.5 Miscellaneous states

There are two STAT codes that do not correspond to a defined I^2C hardware state (see Table 276). These are discussed below.

14.10.5.1 STAT = 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 I^2C block is not involved in a serial transfer.

14.10.5.2 STAT = 0x00

This status code indicates that a bus error has occurred during an I^2C 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 I^2C 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 CON are affected). The SDA and SCL lines are released (a STOP condition is not transmitted).

Table 276. Miscellaneous States

Status Code (STAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From DAT	To CON				
			STA	STO	SI	AA	
0xF8	No relevant state information available; SI = 0.	No DAT action	No CON 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 DAT action	0	1	0	X	Only the internal hardware is affected in the MST or addressed SLV modes. In all cases, the bus is released and the I²C block is switched to the not addressed SLV mode. STO is reset.

14.10.6 Some special cases

The I²C hardware has facilities to handle the following special cases that may occur during a serial transfer:

- Simultaneous Repeated START conditions from two masters
- Data transfer after loss of arbitration
- Forced access to the I²C-bus
- I²C-bus obstructed by a LOW level on SCL or SDA
- Bus error

14.10.6.1 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 51). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

If the I²C hardware detects a Repeated START condition on the I²C-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 I²C block will transmit a normal START condition (state 0x08), and a retry of the total serial data transfer can commence.

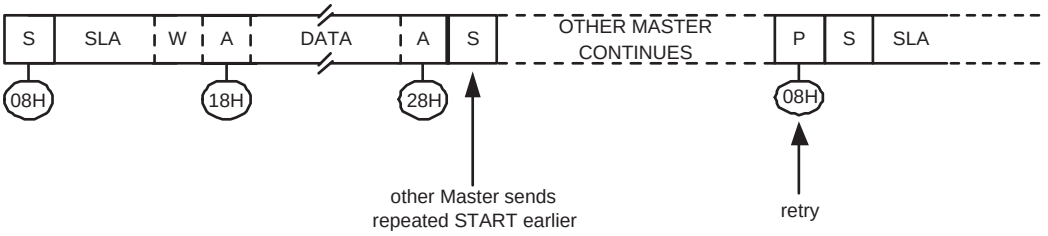


Fig 51. Simultaneous Repeated START conditions from two masters

14.10.6.2 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see Figure 40). Loss of arbitration is indicated by the following states in STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 47 and Figure 48).

If the STA flag in CON 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.

14.10.6.3 Forced access to the I²C-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 I²C-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 I²C-bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I²C hardware behaves as if a STOP condition was received and is able to transmit a START condition. The STO flag is cleared by hardware (see Figure 52).

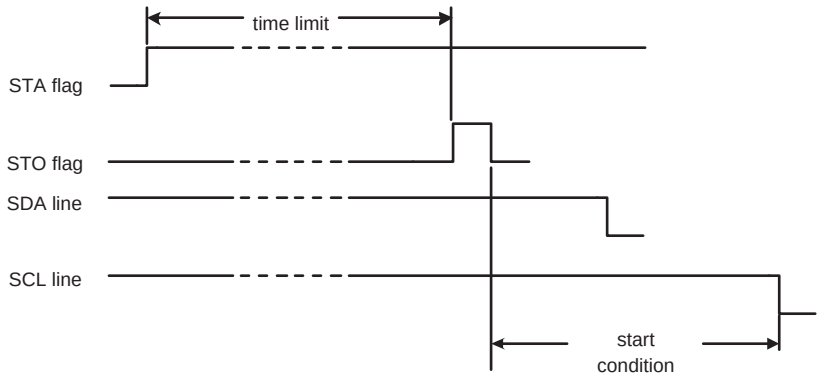
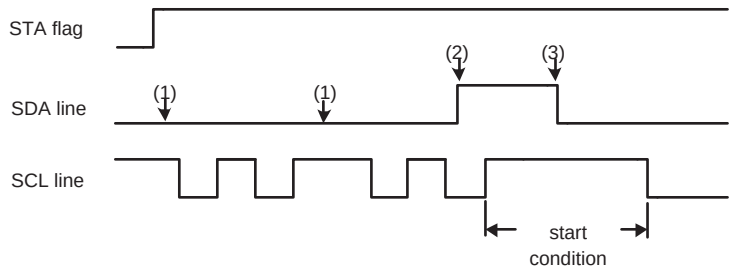


Fig 52. Forced access to a busy I²C-bus

14.10.6.4 I²C-bus obstructed by a LOW level on SCL or SDA

An I²C-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 (see Figure 53). The I²C interface does not include a dedicated time-out 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 I²C peripherals are synchronized.



(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 53. Recovering from a bus obstruction caused by a LOW level on SDA

14.10.6.5 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 I²C 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 I²C 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 276.

14.10.7 I²C state service routines

This section provides examples of operations that must be performed by various I²C state service routines. This includes:

- Initialization of the I²C block after a Reset.
- I²C Interrupt Service
- The 26 state service routines providing support for all four I²C operating modes.

14.10.8 Initialization

In the initialization example, the I^2C 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:

- ADR is loaded with the part's own slave address and the General Call bit (GC)
- The I^2C interrupt enable and interrupt priority bits are set
- The slave mode is enabled by simultaneously setting the I2EN and AA bits in CON and the serial clock frequency (for master modes) is defined by is defined by loading the SCLH and SCLL registers. The master routines must be started in the main program.

The I^2C hardware now begins checking the I^2C-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 STAT is loaded with the appropriate state information.

14.10.9 I^2C interrupt service

When the I^2C interrupt is entered, STAT contains a status code which identifies one of the 26 state services to be executed.

14.10.10 The state service routines

Each state routine is part of the I^2C interrupt routine and handles one of the 26 states.

14.10.11 Adapting state services to an application

The state service examples show the typical actions that must be performed in response to the 26 I^2C state codes. If one or more of the four I^2C 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 time-out during I^2C operations, in order to trap an inoperative bus or a lost service routine.

14.11 Software example

14.11.1 Initialization routine

Example to initialize I^2C Interface as a Slave and/or Master.

1. Load ADR with own Slave Address, enable General Call recognition if needed.
2. Enable I^2C interrupt.
3. Write 0x44 to CONSET to set the I2EN and AA bits, enabling Slave functions. For Master only functions, write 0x40 to CONSET.

14.11.2 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 CONSET 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

14.11.3 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 CONSET 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

14.11.4 I^2C interrupt routine

Determine the I^2C state and which state routine will be used to handle it.

1. Read the I^2C status from STA.
2. Use the status value to branch to one of 26 possible state routines.

14.11.5 Non mode specific states

14.11.5.1 State: 0x00

Bus Error. Enter not addressed Slave mode and release bus.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.5.2 Master States

State 08 and State 10 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.

14.11.5.3 State: 0x08

A START condition has been transmitted. The Slave Address + R/W bit will be transmitted, an ACK bit will be received.

1. Write Slave Address with R/W bit to DAT.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR 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

14.11.5.4 State: 0x10

A Repeated START condition has been transmitted. The Slave Address + R/W bit will be transmitted, an ACK bit will be received.

1. Write Slave Address with R/W bit to DAT.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR 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

14.11.6 Master Transmitter states

14.11.6.1 State: 0x18

Previous state was State 8 or State 10, Slave Address + Write has been transmitted, ACK has been received. The first data byte will be transmitted, an ACK bit will be received.

1. Load DAT with first data byte from Master Transmit buffer.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Increment Master Transmit buffer pointer.
5. Exit

14.11.6.2 State: 0x20

Slave Address + Write has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.6.3 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 CONSET to set the STO and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Exit
5. Load DAT with next data byte from Master Transmit buffer.

6. Write 0x04 to CONSET to set the AA bit.
7. Write 0x08 to CONCLR to clear the SI flag.
8. Increment Master Transmit buffer pointer
9. Exit

14.11.6.4 State: 0x30

Data has been transmitted, NOT ACK received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.6.5 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 CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.7 Master Receive states

14.11.7.1 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 CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.7.2 State: 0x48

Slave Address + Read has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.7.3 State: 0x50

Data has been received, ACK has been returned. Data will be read from DAT. 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 DAT into Master Receive buffer.
2. Decrement the Master data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to CONCLR to clear the SI flag and the AA bit.

4. Exit
5. Write 0x04 to CONSET to set the AA bit.
6. Write 0x08 to CONCLR to clear the SI flag.
7. Increment Master Receive buffer pointer
8. Exit

14.11.7.4 State: 0x58

Data has been received, NOT ACK has been returned. Data will be read from DAT. A STOP condition will be transmitted.

1. Read data byte from DAT into Master Receive buffer.
2. Write 0x14 to CONSET to set the STO and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Exit

14.11.8 Slave Receiver states

14.11.8.1 State: 0x60

Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

14.11.8.2 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 CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit.

14.11.8.3 State: 0x70

General call has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.

4. Initialize Slave data counter.
5. Exit

14.11.8.4 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 CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

14.11.8.5 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 DAT 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 CONCLR to clear the SI flag and the AA bit.
4. Exit.
5. Write 0x04 to CONSET to set the AA bit.
6. Write 0x08 to CONCLR to clear the SI flag.
7. Increment Slave Receive buffer pointer.
8. Exit

14.11.8.6 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 CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.8.7 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 DAT into the Slave Receive buffer.
2. Write 0x0C to CONCLR to clear the SI flag and the AA bit.
3. Exit

14.11.8.8 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 CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.8.9 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 CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

14.11.9 Slave Transmitter states

14.11.9.1 State: 0xA8

Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received.

1. Load DAT from Slave Transmit buffer with first data byte.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

14.11.9.2 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 DAT from Slave Transmit buffer with first data byte.
2. Write 0x24 to CONSET to set the STA and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

14.11.9.3 State: 0xB8

Data has been transmitted, ACK has been received. Data will be transmitted, ACK bit will be received.

1. Load DAT from Slave Transmit buffer with data byte.

2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Increment Slave Transmit buffer pointer.
5. Exit

14.11.9.4 State: 0xC0

Data has been transmitted, NOT ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit.

14.11.9.5 State: 0xC8

The last data byte has been transmitted, ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

15.1 How to read this chapter

CT16B0/1 are available on all LPC11Uxx parts.

15.2 Basic configuration

The CT16B0/1 counter/timers are configured through the following registers:

- Pins: The CT16B0/1 pins must be configured in the IOCON register block.
- Power: In the SYSAHBCLKCTRL register, set bit 7 and 8 in Table 23.
- The peripheral clock is determined by the system clock (see Table 22).

15.3 Features

- Two 16-bit counter/timers with a programmable 16-bit prescaler.
- Counter or timer operation
- Two 16-bit capture channels that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- The timer and prescaler may be configured to be cleared on a designated capture event. This feature permits easy pulse-width measurement by clearing the timer on the leading edge of an input pulse and capturing the timer value on the trailing edge.
- Four 16-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.
- Two 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.
- For each timer, up to four match registers can be configured as PWM allowing to use up to two match outputs as single edge controlled PWM outputs.

15.4 Applications

- Interval timer for counting internal events
- Pulse Width Demodulator via capture input
- Free running timer
- Pulse Width Modulator via match outputs

15.5 Description

Each Counter/timer 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. Each counter/timer also includes one capture input to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, two match registers can be used to provide a single-edge controlled PWM output on the match output pins. It is recommended to use the match registers that are not pinned out to control the PWM cycle length.

Remark: The 16-bit counter/timer0 (CT16B0) and the 16-bit counter/timer1 (CT16B1) are functionally identical except for the peripheral base address.

15.6 Pin description

Table 277 gives a brief summary of each of the counter/timer related pins.

Table 277. Counter/timer pin description

Pin	Type	Description
CT16B0_CAP0 CT16B1_CAP0	Input	Capture Signal: A transition on a capture pin can be configured to load the Capture Register with the value in the counter/timer and optionally generate an interrupt. Counter/Timer block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 15.7.11.
CT16B0_MAT[1:0] CT16B1_MAT[1:0]	Output	External Match Outputs of CT16B0/1: When a match register of CT16B0/1 (MR1:0) equals the timer counter (TC), this output can either toggle, go LOW, go HIGH, or do nothing. The External Match Register (EMR) and the PWM Control Register (PWMCON) control the functionality of this output.

15.7 Register description

The 16-bit counter/timer0 contains the registers shown in Table 278 and the 16-bit counter/timer1 contains the registers shown in Table 279. More detailed descriptions follow.

Table 278. Register overview: 16-bit counter/timer 0 CT16B0 (base address 0x4000 C000)

Name	Access	Address offset	Description	Reset value[1]	Reference
IR	R/W	0x000	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	0	Table 280
TCR	R/W	0x004	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0	Table 281
TC	R/W	0x008	Timer Counter. The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0	Table 282
PR	R/W	0x00C	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0	Table 283
PC	R/W	0x010	Prescale Counter. The 16-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.	0	Table 284
MCR	R/W	0x014	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0	Table 285
MR0	R/W	0x018	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.	0	Table 286
MR1	R/W	0x01C	Match Register 1. See MR0 description.	0	Table 286
MR2	R/W	0x020	Match Register 2. See MR0 description.	0	Table 286
MR3	R/W	0x024	Match Register 3. See MR0 description.	0	Table 286
CCR	R/W	0x028	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.	0	Table 287
CR0	RO	0x02C	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CT16B0_CAP0 input.	0	Table 288
-	-	0x030	Reserved.	-	-
-	-	0x034	Reserved.	-	-
-	-	0x038	Reserved.	-	-
-	-	0x03C	Reserved.	-	-
EMR	R/W	0x03C	External Match Register. The EMR controls the match function and the external match pins CT16B0_MAT[1:0] and CT16B1_MAT[1:0].	0	Table 289
-	-	0x040 - 0x06C	Reserved.	-	-
CTCR	R/W	0x070	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0	Table 291
PWMC	R/W	0x074	PWM Control Register. The PWMCON enables PWM mode for the external match pins CT16B0_MAT[1:0] and CT16B1_MAT[1:0].	0	Table 292

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

Table 279. Register overview: 16-bit counter/timer 1 CT16B1 (base address 0x4001 0000)

Name	Access	Address	Description	Reset value[1]	Reference
IR	R/W	0x000	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	0	Table 280
TCR	R/W	0x004	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0	Table 281
TC	R/W	0x008	Timer Counter. The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0	Table 282
PR	R/W	0x00C	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0	Table 283
PC	R/W	0x010	Prescale Counter. The 16-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.	0	Table 284
MCR	R/W	0x014	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0	Table 285
MR0	R/W	0x018	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.	0	Table 286
MR1	R/W	0x01C	Match Register 1. See MR0 description.	0	Table 286
MR2	R/W	0x020	Match Register 2. See MR0 description.	0	Table 286
MR3	R/W	0x024	Match Register 3. See MR0 description.	0	Table 286
CCR	R/W	0x028	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.	0	Table 287
CR0	RO	0x02C	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CT16B1_CAP0 input.	0	Table 288
-	-	0x030	Reserved.	-	-
-	-	0x034	Reserved.	-	-
-	-	0x038	Reserved.	-	-
EMR	R/W	0x03C	External Match Register. The EMR controls the match function and the external match pins CT16B0_MAT[1:0] and CT16B1_MAT[1:0].	0	Table 289

Table 279. Register overview: 16-bit counter/timer 1 CT16B1 (base address 0x4001 0000) *…continued*

Name	Access	Address	Description	Reset value[1]	Reference
-	-	0x040 - 0x06C	reserved	-	-
CTCR	R/W	0x070	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0	Table 291
PWMC	R/W	0x074	PWM Control Register. The PWMCON enables PWM mode for the external match pins CT16B0_MAT[1:0] and CT16B1_MAT[1:0].	0	Table 292

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

15.7.1 Interrupt Register

The Interrupt Register consists of four bits for the match interrupts and two 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.

Table 280. Interrupt Register (IR, address 0x4000 C000 (CT16B0) and 0x4001 0000 (CT16B1)) bit description

Bit	Symbol	Description	Reset value
0	MR0INT	Interrupt flag for match channel 0.	0
1	MR1INT	Interrupt flag for match channel 1.	0
2	MR2INT	Interrupt flag for match channel 2.	0
3	MR3INT	Interrupt flag for match channel 3.	0
4	CR0INT	Interrupt flag for capture channel 0 event.	0
31:5	-	Reserved	-

15.7.2 Timer Control Register

The Timer Control Register (TCR) is used to control the operation of the counter/timer.

Table 281. Timer Control Register (TCR, address 0x4000 C004 (CT16B0) and 0x4001 0004 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	CEN		Counter enable.	0
		0	The counters are disabled.	
		1	The Timer Counter and Prescale Counter are enabled for counting.	
1	CRST		Counter reset.	0
		0	Do nothing.	
		1	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.	
31:2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

15.7.3 Timer Counter

The 16-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC will count up to the value 0x0000 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.

Table 282: Timer counter registers (TC, address 0x4000 C008 (CT16B0) and 0x4001 0008 (CT16B1)) bit description

Bit	Symbol	Description	Reset value
15:0	TC	Timer counter value.	0
31:16	-	Reserved.	-

15.7.4 Prescale Register

The 16-bit Prescale Register specifies the maximum value for the Prescale Counter.

Table 283: Prescale registers (PR, address 0x4000 C00C (CT16B0) and 0x4001 000C (CT16B1)) bit description

Bit	Symbol	Description	Reset value
15:0	PCVAL	Prescale value.	0
31:16	-	Reserved.	-

15.7.5 Prescale Counter register

The 16-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 between the resolution of the timer and 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 TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, ...

Table 284: Prescale counter registers (PC, address 0x4000 C010 (CT16B0) and 0x4001 0010 (CT16B1)) bit description

Bit	Symbol	Description	Reset value
15:0	PC	Prescale counter value.	0
31:16	-	Reserved.	-

15.7.6 Match Control Register

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 285.

Table 285. Match Control Register (MCR, address 0x4000 C014 (CT16B0) and 0x4001 0014 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	MR0I		Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
1	MR0R		Reset on MR0: the TC will be reset if MR0 matches it.	0
		1	Enabled	
		0	Disabled	
2	MR0S		Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		1	Enabled	
		0	Disabled	
3	MR1I		Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
4	MR1R		Reset on MR1: the TC will be reset if MR1 matches it.	0
		1	Enabled	
		0	Disabled	
5	MR1S		Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.	0
		1	Enabled	
		0	Disabled	
6	MR2I		Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
7	MR2R		Reset on MR2: the TC will be reset if MR2 matches it.	0
		1	Enabled	
		0	Disabled	
8	MR2S		Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.	0
		1	Enabled	
		0	Disabled	
9	MR3I		Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
10	MR3R		Reset on MR3: the TC will be reset if MR3 matches it.	0
		1	Enabled	
		0	Disabled	

Table 285. Match Control Register (MCR, address 0x4000 C014 (CT16B0) and 0x4001 0014 (CT16B1)) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
11	MR3S		Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.	0
		1	Enabled	
		0	Disabled	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

15.7.7 Match Registers

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.

Table 286: Match registers (MR0 to 3, addresses 0x4000 C018 to 24 (CT16B0) and 0x4001 0018 to 24 (CT16B1)) bit description

Bit	Symbol	Description	Reset value
15:0	MATCH	Timer counter match value.	0
31:16	-	Reserved.	-

15.7.8 Capture Control Register

The Capture Control Register is used to control whether the Capture Register is loaded with the value in the Counter/timer 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.

Table 287. Capture Control Register (CCR, address 0x4000 C028 (CT16B0) and 0x4001 0028 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	CAP0RE		Capture on CT16Bn_CAP0 rising edge: a sequence of 0 then 1 on CT16Bn_CAP0 will cause CR0 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	
1	CAP0FE		Capture on CT16Bn_CAP0 falling edge: a sequence of 1 then 0 on CT16Bn_CAP0 will cause CR0 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	

Table 287. Capture Control Register (CCR, address 0x4000 C028 (CT16B0) and 0x4001 0028 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
2	CAP0I		Interrupt on CT16Bn_CAP0 event: a CR0 load due to a CT16Bn_CAP0 event will generate an interrupt.	0
		1	Enabled.	
		0	Disabled.	
31:3	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

15.7.9 Capture Registers

Each Capture register is associated with a device pin and may be loaded with the counter/timer 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.

Table 288: Capture registers (CR0, addresses 0x4000 C02C (CT16B0) and 0x4001 002C (CT16B1)) bit description

Bit	Symbol	Description	Reset value
15:0	CAP	Timer counter capture value.	0
31:16	-	Reserved.	-

15.7.10 External Match Register

The External Match Register provides both control and status of the external match pins CT16Bn_MAT[1:0].

If the match outputs are configured as PWM output, the function of the external match registers is determined by the PWM rules (Section 15.7.13 “Rules for single edge controlled PWM outputs” on page 313).

Table 289. External Match Register (EMR, address 0x4000 C03C (CT16B0) and 0x4001 003C (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	EM0		External Match 0. This bit reflects the state of output CT16B0_MAT0/CT16B1_MAT0, whether or not this output is connected to its pin. When a match occurs between the TC and MR0, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[5:4] control the functionality of this output. This bit is driven to the CT16B0_MAT0/CT16B1_MAT0 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
1	EM1		External Match 1. This bit reflects the state of output CT16B0_MAT1/CT16B1_MAT1, whether or not this output is connected to its pin. When a match occurs between the TC and MR1, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[7:6] control the functionality of this output. This bit is driven to the CT16B0_MAT0/CT16B1_MAT0 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
2	EM2		External Match 2. This bit reflects the state of match channel 2. When a match occurs between the TC and MR2, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[9:8] control the functionality of this output.	0
3	EM3		External Match 3. This bit reflects the state of output of match channel 3. When a match occurs between the TC and MR3, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[11:10] control the functionality of this output.	0
5:4	EMC0		External Match Control 0. Determines the functionality of External Match 0. Table 290 shows the encoding of these bits.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT16Bi_MAT0 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16Bi_MAT0 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
7:6	EMC1		External Match Control 1. Determines the functionality of External Match 1.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT16Bi_MAT1 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16Bi_MAT1 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
9:8	EMC2		External Match Control 2. Determines the functionality of External Match 2.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT16Bi_MAT2 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16Bi_MAT2 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	

Table 289. External Match Register (EMR, address 0x4000 C03C (CT16B0) and 0x4001 003C (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
11:10	EMC3		External Match Control 3. Determines the functionality of External Match 3. Table 290 shows the encoding of these bits.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT16Bi_MAT3 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16Bi_MAT3 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-

Table 290. 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 (CT16Bn_MATm pin is LOW if pinned out).
10	Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out).
11	Toggle the corresponding External Match bit/output.

15.7.11 Count Control Register

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edges 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 half of the PCLK clock. Consequently, the duration of the HIGH/LOW levels on the same CAP input in this case can not be shorter than 1/PCLK.

Bits 7:4 of this register are also used to enable and configure the capture-clears-timer feature. This feature allows for a designated edge on a particular CAP input to reset the timer to all zeros. Using this mechanism to clear the timer on the leading edge of an input pulse and performing a capture on the trailing edge, permits direct pulse-width measurement using a single capture input without the need to perform a subtraction operation in software.

Table 291. Count Control Register (CTCR, address 0x4000 C070 (CT16B0) and 0x4001 0070 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	CTM		Counter/Timer Mode. This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC). **Remark:** If Counter mode is selected in the CTCR, bits 2:0 in the Capture Control Register (CCR) must be programmed as 000.	0
		0x0	Timer Mode: every rising PCLK edge	
		0x1	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		0x2	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		0x3	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	
3:2	CIS		Count Input Select. In counter mode (when bits 1:0 in this register are not 00), these bits select which CAP pin or comparator output is sampled for clocking. Values 0x1 to 0x3 are reserved.	0
		0x0	CT16Bn_CAP0	
4	ENCC		Setting this bit to 1 enables clearing of the timer and the prescaler when the capture-edge event specified in bits 7:5 occurs.	0
7:5	SELCC		When bit 4 is a 1, these bits select which capture input edge will cause the timer and prescaler to be cleared. These bits have no effect when bit 4 is low. Values 0x2 to 0x7 are reserved.	0
		0x0	Rising Edge of CAP0 clears the timer (if bit 4 is set)	
		0x1	Falling Edge of CAP0 clears the timer (if bit 4 is set)	
31:8	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-

15.7.12 PWM Control register

The PWM Control Register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the External Match Register (EMR).

For each timer, a maximum of three single edge controlled PWM outputs can be selected on the CT16Bn_MAT[1:0] outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

Table 292. PWM Control Register (PWMC, address 0x4000 C074 (CT16B0) and 0x4001 0074 (CT16B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	PWMEN0		PWM mode enable for channel0.	0
		0	CT16Bi_MAT0 is controlled by EM0.	
		1	PWM mode is enabled for CT16Bi_MAT0.	
1	PWMEN1		PWM mode enable for channel1.	0
		0	CT16Bi_MAT01 is controlled by EM1.	
		1	PWM mode is enabled for CT16Bi_MAT1.	
2	PWMEN2		PWM mode enable for channel2.	0
		0	CT16Bi_MAT2 is controlled by EM2.	
		1	PWM mode is enabled for CT16Bi_MAT2.	
3	PWMEN3		PWM mode enable for channel3.	0
		0	CT16Bi_MAT3 is controlled by EM3.	
		1	PWM mode is enabled for CT16Bi_MAT3.	
31:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-

15.7.13 Rules for single edge controlled PWM outputs

1. All single edge controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
2. Each PWM output will go HIGH when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
3. If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal will be cleared on the next start of the next PWM cycle.
4. If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output will be reset to LOW on the next clock tick. Therefore, the PWM output will always consist of a one clock tick wide positive pulse with a period determined by the PWM cycle length (i.e. the timer reload value).
5. If a match register is set to zero, then the PWM output will go to HIGH the first time the timer goes back to zero and will stay HIGH continuously.

Note: When the match outputs are selected to perform as PWM outputs, the timer reset (MRnR) and timer stop (MRnS) bits in the Match Control Register MCR must be set to zero except for the match register setting the PWM cycle length. For this register, set the MRnR bit to one to enable the timer reset when the timer value matches the value of the corresponding match register.

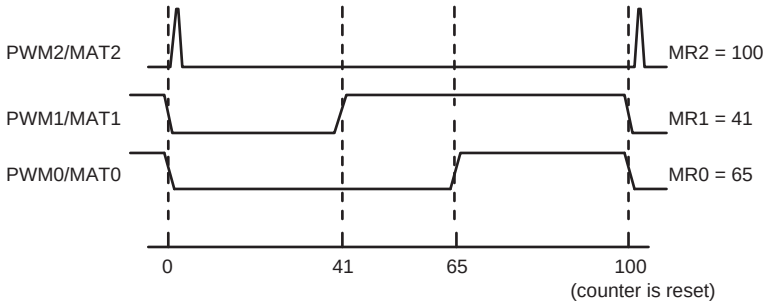


Fig 54. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.

15.8 Example timer operation

Figure 55 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 56 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.

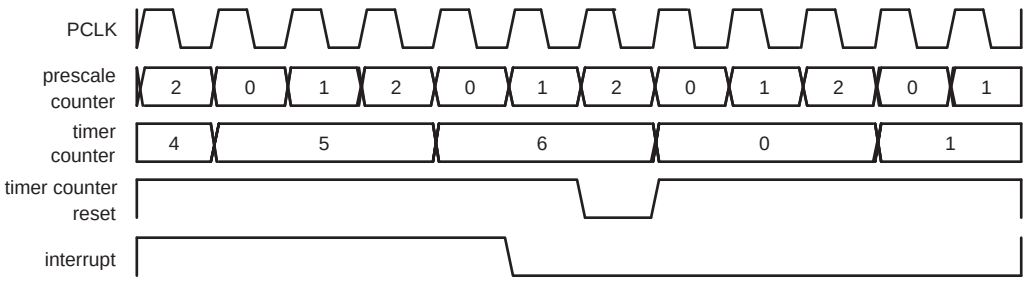


Fig 55. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled

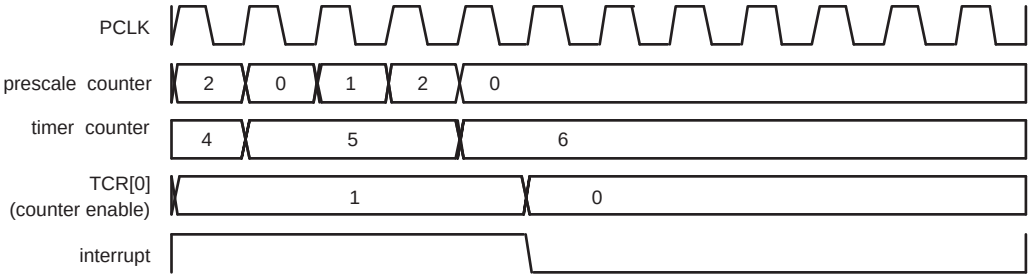


Fig 56. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled

15.9 Architecture

The block diagram for counter/timer0 and counter/timer1 is shown in Figure 57.

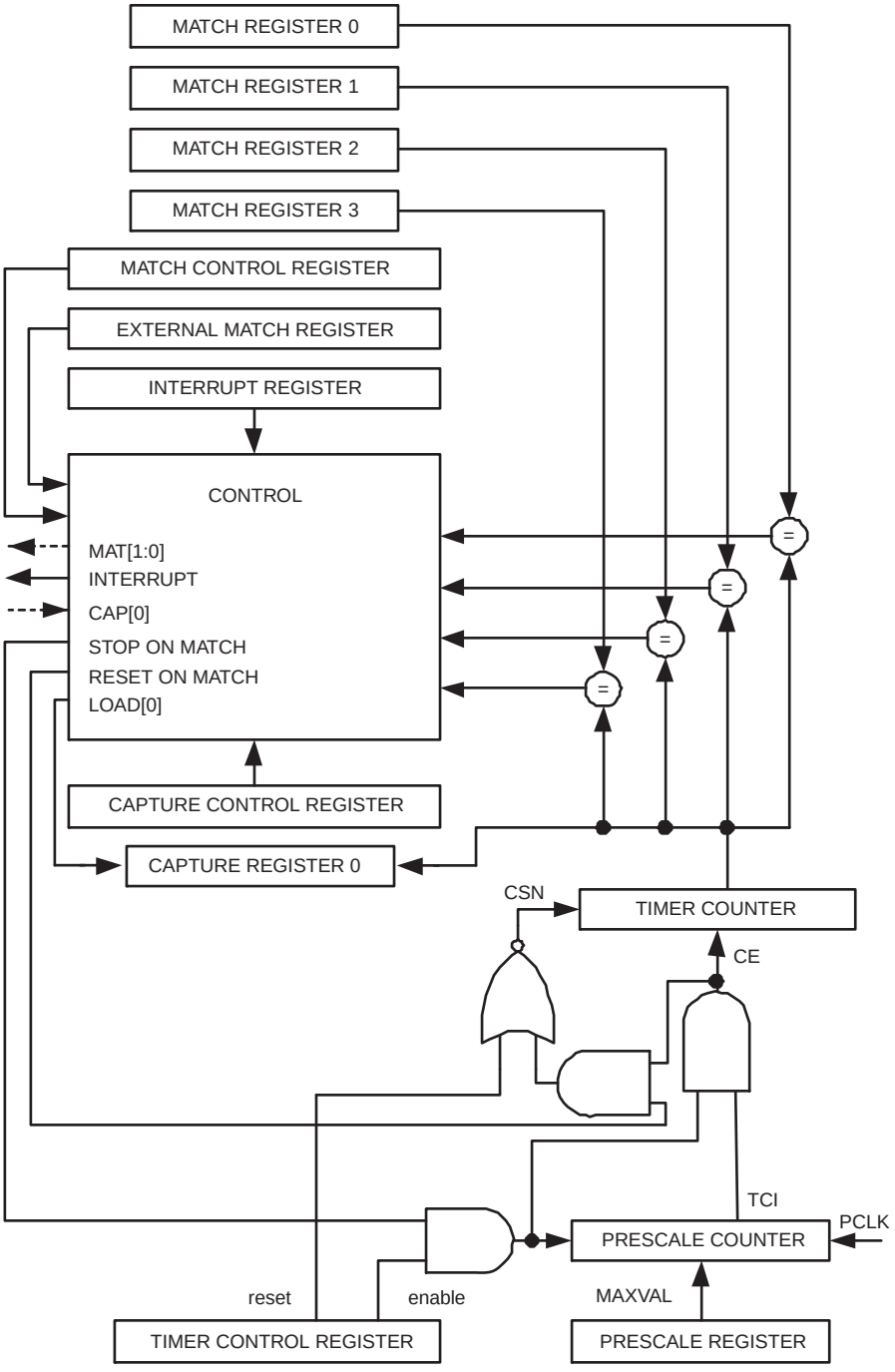


Fig 57. 16-bit counter/timer block diagram

16.1 How to read this chapter

CT32B0/1 are available on all LPC11Uxx parts. The CT32B1CAP1 input is only available on the TFBGA48 package. For all other packages, the registers controlling the CT32B1_CAP1 input are reserved.

16.2 Basic configuration

The CT32B0/1 counter/timers are configured through the following registers:

- Pins: The CT32B0/1 pins must be configured in the IOCON register block.
- Power: In the SYSAHBCLKCTRL register, set bit 9 and 10 in Table 23.
- The peripheral clock is determined by the system clock (see Table 22).

16.3 Features

- Two 32-bit counter/timers with a programmable 32-bit prescaler.
- Counter or timer operation.
- Four 32-bit capture channels that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- The timer and prescaler may be configured to be cleared on a designated capture event. This feature permits easy pulse-width measurement by clearing the timer on the leading edge of an input pulse and capturing the timer value on the trailing edge.
- 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.
- 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.
- For each timer, up to four match registers can be configured as PWM allowing to use up to three match outputs as single edge controlled PWM outputs.

16.4 Applications

- Interval timer for counting internal events
- Pulse Width Demodulator via capture input
- Free running timer

- Pulse Width Modulator via match outputs

16.5 General description

Each Counter/timer 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. Each counter/timer also includes one capture input to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, three match registers can be used to provide a single-edge controlled PWM output on the match output pins. One match register is used to control the PWM cycle length.

16.6 Pin description

Table 293 gives a brief summary of each of the counter/timer related pins.

Table 293. Counter/timer pin description

Pin	Type	Description
CT32B0_CAP0] CT32B1_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. The counter/timer block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 16.7.11 “Count Control Register” on page 325.
CT32B0_MAT[3:0] CT32B1_MAT[3:0]	Output	External Match Output of CT32B0/1: 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) and the PWM Control register (PWMCON) control the functionality of this output.

16.7 Register description

32-bit counter/timer0 contains the registers shown in Table 294 and 32-bit counter/timer1 contains the registers shown in Table 295. More detailed descriptions follow.

Table 294. Register overview: 32-bit counter/timer 0 CT32B0 (base address 0x4001 4000)

Name	Access	Address offset	Description	Reset value[1]	Reference
IR	R/W	0x000	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	0	Table 296
TCR	R/W	0x004	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0	Table 297
TC	R/W	0x008	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0	Table 298
PR	R/W	0x00C	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0	Table 299

Table 294. Register overview: 32-bit counter/timer 0 CT32B0 (base address 0x4001 4000) *…continued*

Name	Access	Address offset	Description	Reset value[1]	Reference
PC	R/W	0x010	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.	0	Table 300
MCR	R/W	0x014	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0	Table 301
MR0	R/W	0x018	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.	0	Table 302
MR1	R/W	0x01C	Match Register 1. See MR0 description.	0	Table 302
MR2	R/W	0x020	Match Register 2. See MR0 description.	0	Table 302
MR3	R/W	0x024	Match Register 3. See MR0 description.	0	Table 302
CCR	R/W	0x028	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.	0	Table 303
CR0	RO	0x02C	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CT32B0_CAP0 input.	0	Table 304
-	-	0x030	Reserved.	-	-
-	-	0x034	Reserved.	-	-
-	-	0x038	Reserved.	-	-
EMR	R/W	0x03C	External Match Register. The EMR controls the match function and the external match pins CT32Bn_MAT[3:0].	0	Table 305
-	-	0x040 - 0x06C	Reserved.	-	-
CTCR	R/W	0x070	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0	Table 307
PWMC	R/W	0x074	PWM Control Register. The PWMCON enables PWM mode for the external match pins CT32Bn_MAT[3:0].	0	Table 308

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

Table 295. Register overview: 32-bit counter/timer 1 CT32B1 (base address 0x4001 8000)

Name	Access	Address offset	Description	Reset value[1]	Reference
IR	R/W	0x000	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	0	Table 296
TCR	R/W	0x004	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0	Table 297
TC	R/W	0x008	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0	Table 298
PR	R/W	0x00C	Prescale Register. When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0	Table 299
PC	R/W	0x010	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.	0	Table 300
MCR	R/W	0x014	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0	Table 301
MR0	R/W	0x018	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.	0	Table 302
MR1	R/W	0x01C	Match Register 1. See MR0 description.	0	Table 302
MR2	R/W	0x020	Match Register 2. See MR0 description.	0	Table 302
MR3	R/W	0x024	Match Register 3. See MR0 description.	0	Table 302
CCR	R/W	0x028	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.	0	Table 303
CR0	RO	0x02C	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CT32B1_CAP0 input.	0	Table 304
CR1	RO	0x030	Capture Register 1. CR1 is loaded with the value of TC when there is an event on the CT32B1_CAP1 input.	0	-
-	-	0x034	Reserved.	-	-
-	-	0x038	Reserved.	-	-
EMR	R/W	0x03C	External Match Register. The EMR controls the match function and the external match pins CT32Bn_MAT[3:0].	0	Table 305
-	-	0x040 - 0x06C	Reserved.	-	-
CTCR	R/W	0x070	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0	Table 307
PWMC	R/W	0x074	PWM Control Register. The PWMCON enables PWM mode for the external match pins CT32Bn_MAT[3:0].	0	Table 308

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

16.7.1 Interrupt Register

The Interrupt Register consists of four bits for the match interrupts and four 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.

Table 296: Interrupt Register (IR, address 0x4001 4000 (CT32B0) and IR, address 0x4001 8000) bit description

Bit	Symbol	Description	Reset value
0	MR0INT	Interrupt flag for match channel 0.	0
1	MR1INT	Interrupt flag for match channel 1.	0
2	MR2INT	Interrupt flag for match channel 2.	0
3	MR3INT	Interrupt flag for match channel 3.	0
4	CR0INT	Interrupt flag for capture channel 0 event.	0
5	CR1INT	Interrupt flag for capture channel 1 event.	0
31:6	-	Reserved	-

16.7.2 Timer Control Register

The Timer Control Register (TCR) is used to control the operation of the counter/timer.

Table 297: Timer Control Register (TCR, address 0x4001 4004 (CT32B0) and 0x4001 8004 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	CEN		Counter enable.	0
		0	The counters are disabled.	
		1	The Timer Counter and Prescale Counter are enabled for counting.	
1	CRST		Counter reset.	0
		0	Do nothing.	
		1	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.	
31:2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

16.7.3 Timer Counter registers

The 32-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC 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.

Table 298: Timer counter registers (TC, address 0x4001 4008 (CT32B0) and 0x4001 8008 (CT32B1)) bit description

Bit	Symbol	Description	Reset value
31:0	TC	Timer counter value.	0

16.7.4 Prescale Register

The 32-bit Prescale Register specifies the maximum value for the Prescale Counter.

Table 299: Prescale registers (PR, address 0x4001 400C (CT32B0) and 0x4001 800C (CT32B1)) bit description

Bit	Symbol	Description	Reset value
31:0	PCVAL	Prescaler value.	0

16.7.5 Prescale Counter Register

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 between the resolution of the timer and 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 TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, etc.

Table 300: Prescale registers (PC, address 0x4001 4010 (CT32B0) and 0x4001 8010 (CT32B1)) bit description

Bit	Symbol	Description	Reset value
31:0	PC	Prescale counter value.	0

16.7.6 Match Control Register

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 301.

Table 301: Match Control Register (MCR, address 0x4001 4014 (CT32B0) and 0x4001 8014 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	MR0I		Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
1	MR0R		Reset on MR0: the TC will be reset if MR0 matches it.	0
		1	Enabled	
		0	Disabled	
2	MR0S		Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		1	Enabled	
		0	Disabled	
3	MR1I		Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	

Table 301: Match Control Register (MCR, address 0x4001 4014 (CT32B0) and 0x4001 8014 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
4	MR1R		Reset on MR1: the TC will be reset if MR1 matches it.	0
		1	Enabled	
		0	Disabled	
5	MR1S		Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.	0
		1	Enabled	
		0	Disabled	
6	MR2I		Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
7	MR2R		Reset on MR2: the TC will be reset if MR2 matches it.	0
		1	Enabled	
		0	Disabled	
8	MR2S		Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.	0
		1	Enabled	
		0	Disabled	
9	MR3I		Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0
		1	Enabled	
		0	Disabled	
10	MR3R		Reset on MR3: the TC will be reset if MR3 matches it.	0
		1	Enabled	
		0	Disabled	
11	MR3S		Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.	0
		1	Enabled	
		0	Disabled	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

16.7.7 Match Registers

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.

Table 302: Match registers (MR0 to 3, addresses 0x4001 4018 to 24 (CT32B0) and 0x4001 8018 to 24 (CT32B1)) bit description

Bit	Symbol	Description	Reset value
31:0	MATCH	Timer counter match value.	0

16.7.8 Capture Control Register (CCR)

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.

Table 303: Capture Control Register (CCR, address 0x4001 4028 (CT32B0) and 0x4001 8028 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	CAP0RE		Capture on CT32Bn_CAP0 rising edge: a sequence of 0 then 1 on CT32Bn_CAP0 will cause CR0 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	
1	CAP0FE		Capture on CT32Bn_CAP0 falling edge: a sequence of 1 then 0 on CT32Bn_CAP0 will cause CR0 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	
2	CAP0I		Interrupt on CT32Bn_CAP0 event: a CR0 load due to a CT32Bn_CAP0 event will generate an interrupt.	0
		1	Enabled.	
		0	Disabled.	
3	CAP1RE		Capture on CT32Bn_CAP1 rising edge: a sequence of 0 then 1 on CT32Bn_CAP1 will cause CR1 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	
4	CAP1FE		Capture on CT32Bn_CAP1 falling edge: a sequence of 1 then 0 on CT32Bn_CAP1 will cause CR1 to be loaded with the contents of TC.	0
		1	Enabled.	
		0	Disabled.	
5	CAP1I		Interrupt on CT32Bn_CAP1 event: a CR1 load due to a CT32Bn_CAP1 event will generate an interrupt.	0
		1	Enabled.	
		0	Disabled.	
316	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

16.7.9 Capture Register

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.

Table 304: Capture registers (CR0, addresses 0x4001 402C (CT32B0) and 0x4001 802C to 30 (CT32B1)) bit description

Bit	Symbol	Description	Reset value
31:0	CAP	Timer counter capture value.	0

16.7.10 External Match Register

The External Match Register provides both control and status of the external match pins CAP32Bn_MAT[3:0].

If the match outputs are configured as PWM output, the function of the external match registers is determined by the PWM rules (Section 16.7.13 “Rules for single edge controlled PWM outputs” on page 327).

Table 305: External Match Register (EMR, address 0x4001 403C (CT32B0) and 0x4001 803C (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	EM0		External Match 0. This bit reflects the state of output CT32Bn_MAT0, whether or not this output is connected to its pin. When a match occurs between the TC and MR0, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[5:4] control the functionality of this output. This bit is driven to the CT32B0_MAT0/CT32B1_MAT0 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
1	EM1		External Match 1. This bit reflects the state of output CT32Bn_MAT1, whether or not this output is connected to its pin. When a match occurs between the TC and MR1, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[7:6] control the functionality of this output. This bit is driven to the CT32B0_MAT1/CT32B1_MAT1 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
2	EM2		External Match 2. This bit reflects the state of output CT32Bn_MAT2, whether or not this output is connected to its pin. When a match occurs between the TC and MR2, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[9:8] control the functionality of this output. This bit is driven to the CT32B0_MAT2/CT32B1_MAT2 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
3	EM3		External Match 3. This bit reflects the state of output CT32Bn_MAT3, whether or not this output is connected to its pin. When a match occurs between the TC and MR3, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[11:10] control the functionality of this output. This bit is driven to the CT32B3_MAT0/CT32B1_MAT3 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH).	0
5:4	EMC0		External Match Control 0. Determines the functionality of External Match 0.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT32Bi_MAT0 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT32Bi_MAT0 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
7:6	EMC1		External Match Control 1. Determines the functionality of External Match 1.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT32Bi_MAT1 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT32Bi_MAT1 pin is HIGH if pinned out).	

Table 305: External Match Register (EMR, address 0x4001 403C (CT32B0) and 0x4001 803C (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
		0x3	Toggle the corresponding External Match bit/output.	
9:8	EMC2		External Match Control 2. Determines the functionality of External Match 2.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT32Bi_MAT2 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT32Bi_MAT2 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
11:10	EMC3		External Match Control 3. Determines the functionality of External Match 3.	00
		0x0	Do Nothing.	
		0x1	Clear the corresponding External Match bit/output to 0 (CT32Bi_MAT3 pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT32Bi_MAT3 pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 306. 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 (CT32Bn_MATm pin is LOW if pinned out).
10	Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out).
11	Toggle the corresponding External Match bit/output.

16.7.11 Count Control Register

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edges 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 half of the PCLK clock. Consequently, duration of the HIGH/LOWLOW levels on the same CAP input in this case can not be shorter than 1/PCLK.

Bits 7:4 of this register are also used to enable and configure the capture-clears-timer feature. This feature allows for a designated edge on a particular CAP input to reset the timer to all zeros. Using this mechanism to clear the timer on the leading edge of an input pulse and performing a capture on the trailing edge, permits direct pulse-width measurement using a single capture input without the need to perform a subtraction operation in software.

Table 307: Count Control Register (CTCR, address 0x4001 4070 (CT32B0) and 0x4001 8070 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	CTM		Counter/Timer Mode. This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC). **Remark:** If Counter mode is selected in the CTCR, bits 2:0 in the Capture Control Register (CCR) must be programmed as 000.	00
		0x0	Timer Mode: every rising PCLK edge	
		0x1	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		0x2	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		0x3	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	
3:2	CIS		Count Input Select. In counter mode (when bits 1:0 in this register are not 00), these bits select which CAP pin or comparator output is sampled for clocking. **Remark:** If Counter mode is selected in the CTCR, the 3 bits for that input in the Capture Control Register (CCR) must be programmed as 000. Values 0x2 to 0x3 are reserved.	00
		0x0	CT32Bn_CAP0	
		0x1	CT32Bn_CAP1	
4	ENCC		Setting this bit to 1 enables clearing of the timer and the prescaler when the capture-edge event specified in bits 7:5 occurs.	0
7:5	SEICC		When bit 4 is a 1, these bits select which capture input edge will cause the timer and prescaler to be cleared. These bits have no effect when bit 4 is low. Values 0x3 to 0x7 are reserved.	
		0x0	Rising Edge of CAP0 clears the timer (if bit 4 is set)	
		0x1	Falling Edge of CAP0 clears the timer (if bit 4 is set)	
		0x2	Rising Edge of CAP1 clears the timer (if bit 4 is set)	
		0x3	Falling Edge of CAP1 clears the timer (if bit 4 is set)	
31:8	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-

16.7.12 PWM Control Register

The PWM Control Register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the External Match Register (EMR).

For each timer, a maximum of three single edge controlled PWM outputs can be selected on the MATn.2:0 outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

Table 308: PWM Control Register (PWMC, 0x4001 4074 (CT32B0) and 0x4001 8074 (CT32B1)) bit description

Bit	Symbol	Value	Description	Reset value
0	PWMEN0		PWM mode enable for channel0.	0
		0	CT32Bi_MAT0 is controlled by EM0.	
		1	PWM mode is enabled for CT32Bi_MAT0.	
1	PWMEN1		PWM mode enable for channel1.	0
		0	CT32Bi_MAT01 is controlled by EM1.	
		1	PWM mode is enabled for CT32Bi_MAT1.	
2	PWMEN2		PWM mode enable for channel2.	0
		0	CT32Bi_MAT2 is controlled by EM2.	
		1	PWM mode is enabled for CT32Bi_MAT2.	
3	PWMEN3		PWM mode enable for channel3. **Note:** It is recommended to use match channel 3 to set the PWM cycle.	0
		0	CT32Bi_MAT3 is controlled by EM3.	
		1	PWM mode is enabled for CT132Bi_MAT3.	
31:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

16.7.13 Rules for single edge controlled PWM outputs

1. All single edge controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
2. Each PWM output will go HIGH when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
3. If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal will be cleared with the start of the next PWM cycle.

4. If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output will be reset to LOW on the next clock tick after the timer reaches the match value. Therefore, the PWM output will always consist of a one clock tick wide positive pulse with a period determined by the PWM cycle length (i.e. the timer reload value).
5. If a match register is set to zero, then the PWM output will go to HIGH the first time the timer goes back to zero and will stay HIGH continuously.

Note: When the match outputs are selected to perform as PWM outputs, the timer reset (MRnR) and timer stop (MRnS) bits in the Match Control Register MCR must be set to zero except for the match register setting the PWM cycle length. For this register, set the MRnR bit to one to enable the timer reset when the timer value matches the value of the corresponding match register.

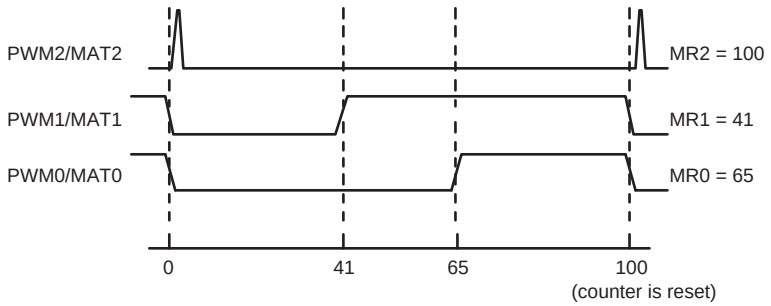


Fig 58. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.

16.8 Example timer operation

Figure 59 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 60 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.

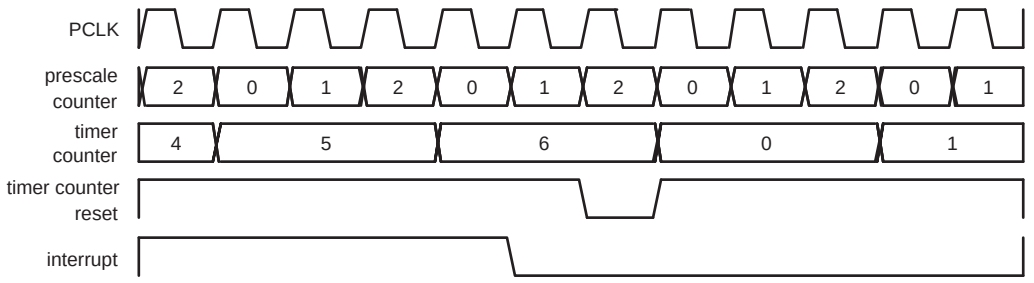


Fig 59. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled

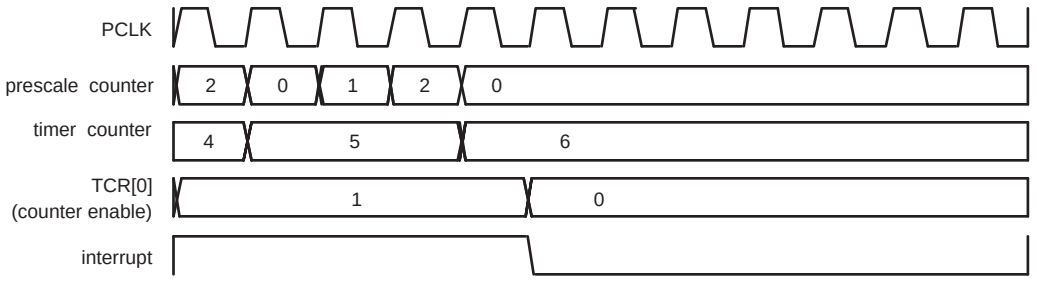


Fig 60. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled

16.9 Architecture

The block diagram for 32-bit counter/timer0 and 32-bit counter/timer1 is shown in Figure 61.

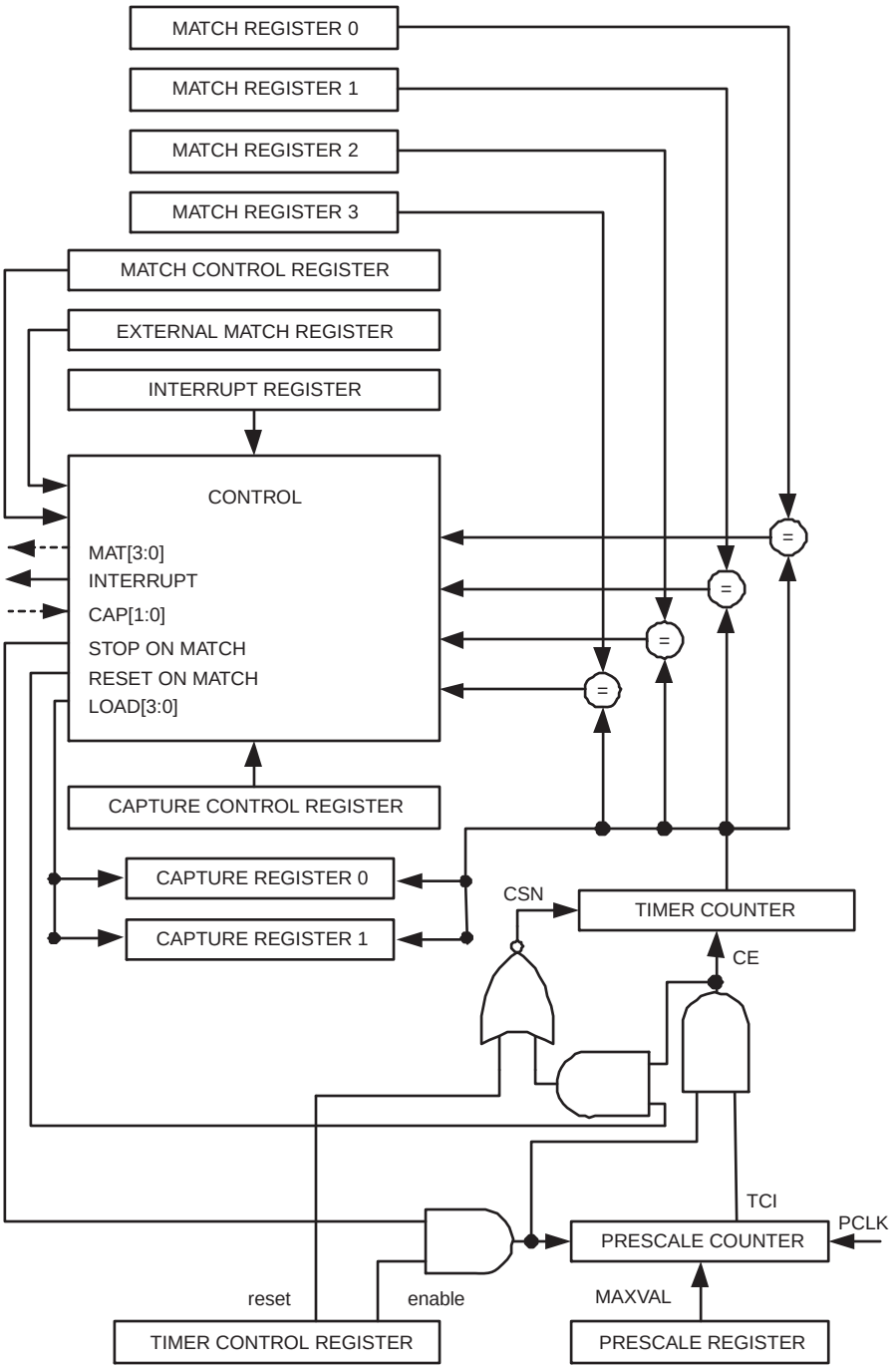


Fig 61. 32-bit counter/timer block diagram

17.1 How to read this chapter

The WWDT is identical on all LPC11Uxx parts.

17.2 Basic configuration

The WWDT is configured through the following registers:

- Power to the register interface (WWDT PCLK clock): In the SYSAHBCLKCTRL register, set bit 15 in Table 23.
- Enable the WWDT clock source (the watchdog oscillator or the IRC) in the PDRUNCFG register (Table 46).
- For waking up from a WWDT interrupt, enable the watchdog interrupt for wake-up in the STARTERP1 register (Table 43).

17.3 Features

- Internally resets chip if not reloaded during the programmable time-out period.
- Optional windowed operation requires reload to occur between a minimum and maximum time-out period, both programmable.
- Optional warning interrupt can be generated at a programmable time prior to watchdog time-out.
- 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.
- Incorrect feed sequence causes immediate watchdog event if 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.
- The Watchdog clock (WDCLK) source can be selected as the Internal High frequency oscillator (IRC) or the WatchDog oscillator.
- The Watchdog timer can be configured to run in Deep-sleep or Power-down mode when using the watchdog oscillator as the clock source.
- Debug mode.

17.4 Applications

The purpose of the Watchdog Timer is to reset or interrupt the microcontroller within a programmable time if it enters an erroneous state. When enabled, a watchdog reset and/or will be generated if the user program fails to “feed” (reload) the Watchdog within a predetermined amount of time.

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.

17.5 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 the TC register.
- Set the Watchdog timer operating mode in the MOD register.
- Set a value for the watchdog window time in the WINDOW register if windowed operation is desired.
- Set a value for the watchdog warning interrupt in the WARNINT register if a warning interrupt is desired.
- Enable the Watchdog by writing 0xAA followed by 0x55 to the FEED 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 for an 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 WARNINT register.

17.5.1 Block diagram

The block diagram of the Watchdog is shown below in the Figure 62. The synchronization logic (PCLK - WDCLK) is not shown in the block diagram.

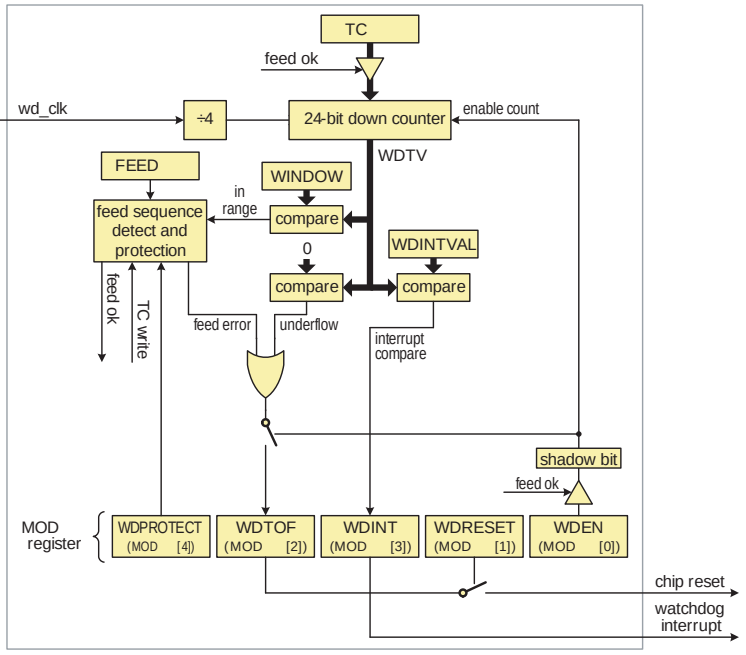


Fig 62. Watchdog block diagram

17.6 Clocking and power control

The watchdog timer block uses two clocks: PCLK and WDCLK. PCLK is used for the APB accesses to the watchdog registers and is derived from the system clock (see Figure 5). The WDCLK is used for the watchdog timer counting and is derived from the wdt_clk in Figure 5. Either the IRC or the watchdog oscillator can be used as wdt_clk in Active mode, Sleep mode, and Deep-sleep modes. In Power-down mode only the watchdog oscillator is available.

The synchronization logic between the two clock domains works as follows: When the MOD and TC registers are updated by APB operations, the new value will take effect in 3 WDCLK cycles on the logic in the WDCLK clock domain.

When the watchdog timer is counting on WDCLK, the synchronization logic will first lock the value of the counter on WDCLK and then synchronize it with PCLK, so that the CPU can read the WDTV register.

Remark: Because of the synchronization step, software must add a delay of three WDCLK clock cycles between the feed sequence and the time the WDPROTECT bit is enabled in the MOD register. The length of the delay depends on the selected watchdog clock WDCLK.

17.7 Using the WWDT lock features

The WWDT supports several lock features which can be enabled to ensure that the WWDT is running at all times:

- Accidental overwrite of the WWDT clock source
- Changing the WWDT clock source
- Changing the WWDT reload value

17.7.1 Accidental overwrite of the WWDT clock

If bit 31 of the WWDT CLKSEL register (Table 315) is set, writes to bit 0 of the CLKSEL register, the clock source select bit, will be ignored and the clock source will not change.

17.7.2 Changing the WWDT clock source

If bit 5 in the WWDT MOD register is set, the current clock source as selected in the CLKSEL register is locked and can not be changed either by software or by hardware when Sleep, Deep-sleep or Power-down modes are entered. Therefore, the user must ensure that the appropriate WWDT clock source for each power mode is selected **before** setting bit 5 in the MOD register:

- Active or Sleep modes: Both the IRC or the watchdog oscillator are allowed.
- Deep-sleep mode: Both the IRC and the watchdog oscillator are allowed. However, using the IRC during Deep-sleep mode will increase the power consumption. To minimize power consumption, use the watchdog oscillator as clock source.
- Power-down mode: Only the watchdog oscillator is allowed as clock source for the WWDT. Therefore, before setting bit 5 and locking the clock source, the WWDT clock source must be set to the watchdog oscillator. Otherwise, the part may not be able to enter Power-down mode.
- Deep power-down mode: No clock locking mechanisms are in effect as neither the WWDT nor any of the clocks are running. However, an additional lock bit in the PMU can be set to prevent the part from even entering Deep power-down mode (see Table 53).

The clock source lock mechanism can only be disabled by a reset of any type.

17.7.3 Changing the WWDT reload value

If bit 4 is set in the WWDT MOD register, the watchdog time-out value (TC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW.

The reload overwrite lock mechanism can only be disabled by a reset of any type.

17.8 Register description

The Watchdog Timer contains the registers shown in Table 309.

Table 309. Register overview: Watchdog timer (base address 0x4000 4000)

Name	Access	Address offset	Description	Reset Value[1]	Reference
MOD	R/W	0x000	Watchdog mode register. This register contains the basic mode and status of the Watchdog Timer.	0	Table 310
TC	R/W	0x004	Watchdog timer constant register. This 24-bit register determines the time-out value.	0xFF	Table 312
FEED	WO	0x008	Watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer with the value contained in WDTC.	NA	Table 313
TV	RO	0x00C	Watchdog timer value register. This 24-bit register reads out the current value of the Watchdog timer.	0xFF	Table 314
CLKSEL	R/W	0x010	Watchdog clock select register.	0	Table 315
WARNINT	R/W	0x014	Watchdog Warning Interrupt compare value.	0	Table 316
WINDOW	R/W	0x018	Watchdog Window compare value.	0xFF FFFF	Table 317

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

17.8.1 Watchdog mode register

The WDMOD register controls the operation of the Watchdog. Note that a watchdog feed must be performed before any changes to the WDMOD register take effect.

Table 310. Watchdog mode register (MOD - 0x4000 4000) bit description

Bit	Symbol	Value	Description	Reset value
0	WDEN		Watchdog enable bit. Once this bit has been written with a 1, it cannot be rewritten with a 0.	0
		0	The watchdog timer is stopped.	
		1	The watchdog timer is running.	
1	WDRESET		Watchdog reset enable bit. Once this bit has been written with a 1 it cannot be rewritten with a 0.	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		Warning interrupt flag. Set when the timer reaches the value in WDWARNINT. Cleared by software.	0

Table 310. Watchdog mode register (MOD - 0x4000 4000) bit description

Bit	Symbol	Value	Description	Reset value
4	WDPROTECT		Watchdog update mode. This bit can be set once by software and is only cleared by a reset.	0
		0	The watchdog time-out value (TC) can be changed at any time.	
		1	The watchdog time-out value (TC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW.	
5	LOCK		A 1 in this bit prevents disabling or powering down the clock source selected by bit 0 of the WDCLKSRC register and also prevents switching to a clock source that is disabled or powered down. This bit can be set once by software and is only cleared by any reset. **Remark:** If this bit is one and the WWDT clock source is the IRC when Deep-sleep or Power-down modes are entered, the IRC remains running thereby increasing power consumption in Deep-sleep mode and potentially preventing the part from entering Power-down mode correctly (see Section 17.7).	0
31:6	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	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 PROTECT =1 and an attempt is made to write to the TC 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 WARNINT. This flag is cleared when any reset occurs, and is cleared by software by writing a 0 to this bit.

In all power modes except Deep power-down mode, a Watchdog reset or interrupt can occur when the watchdog is running and has an operating clock source. The watchdog oscillator or the IRC can be selected to keep running in Sleep and Deep-sleep modes. In Power-down mode, only the watchdog oscillator is allowed. If a watchdog interrupt occurs in Sleep, Deep-sleep mode, or Power-down mode and the WWDT interrupt is enabled in the NVIC, the device will wake up. Note that in Deep-sleep and Power-down modes, the WWDT interrupt must be enabled in the STARTERP1 register in addition to the NVIC.

Table 311. 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 WDWARNINT will set the WDINT 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 WDWARNINT will set the WDINT 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 WDWINDOW will also cause a watchdog reset.

17.8.2 Watchdog Timer Constant register

The TC register determines the time-out value. Every time a feed sequence occurs the value in the TC is loaded into the Watchdog timer. The TC resets to 0x00 00FF. Writing a value below 0xFF will cause 0x00 00FF to be loaded into the TC. 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 TC before the watchdog counter is below the values of WDWARNINT and WDWINDOW will cause a watchdog reset and set the WDTOF flag.

Table 312. Watchdog Timer Constant register (TC - 0x4000 4004) bit description

Bit	Symbol	Description	Reset Value
23:0	COUNT	Watchdog time-out value.	0x00 00FF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.8.3 Watchdog Feed register

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.

It is good practise to disable interrupts around a feed sequence, if the application is such that some/any interrupt might result in rescheduling processor control away from the current task in the middle of the feed, and then lead to some other access to the WDT before control is returned to the interrupted task.

Table 313. Watchdog Feed register (FEED - 0x4000 4008) bit description

Bit	Symbol	Description	Reset Value
7:0	FEED	Feed value should be 0xAA followed by 0x55.	NA
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.8.4 Watchdog Timer Value register

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 314. Watchdog Timer Value register (TV - 0x4000 400C) bit description

Bit	Symbol	Description	Reset Value
23:0	COUNT	Counter timer value.	0x00 00FF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.8.5 Watchdog Clock Select register

The LOCK bit in this register prevents software from changing the clock source inadvertently. Once the LOCK bit is set, software cannot change the clock source until this register has been reset from any reset source.

Table 315. Watchdog Clock Select register (CLKSEL - 0x4000 4010) bit description

Bit	Symbol	Value	Description	Reset Value
0	CLKSEL		Selects source of WDT clock	0
		0	IRC	
		1	Watchdog oscillator (WDOSC)	
30:1	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
31	LOCK		If this bit is set to one, writing to this register does not affect bit 0 (that is the clock source cannot be changed). The clock source can only by changed after a reset from any source.	0

17.8.6 Watchdog Timer Warning Interrupt register

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 WARNINT, 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 WARNINT is 0, the interrupt will occur at the same time as the watchdog event.

Table 316. Watchdog Timer Warning Interrupt register (WARNINT - 0x4000 4014) bit description

Bit	Symbol	Description	Reset Value
9:0	WARNINT	Watchdog warning interrupt compare value.	0
31:10	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.8.7 Watchdog Timer Window register

The WDWINDOW register determines the highest WDTV value allowed when a watchdog feed is performed. If a feed sequence occurs when WDTV is greater than the value in WDWINDOW, a watchdog event will occur.

WDWINDOW resets to the maximum possible WDTV value, so windowing is not in effect.

Table 317. Watchdog Timer Window register (WINDOW - 0x4000 4018) bit description

Bit	Symbol	Description	Reset Value
23:0	WINDOW	Watchdog window value.	0xFF FFFF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.9 Watchdog timing examples

The following figures illustrate several aspects of Watchdog Timer operation.

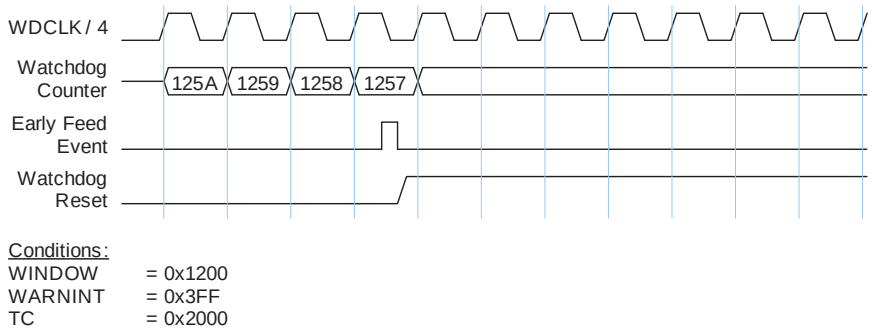


Fig 63. Early Watchdog Feed with Windowed Mode Enabled

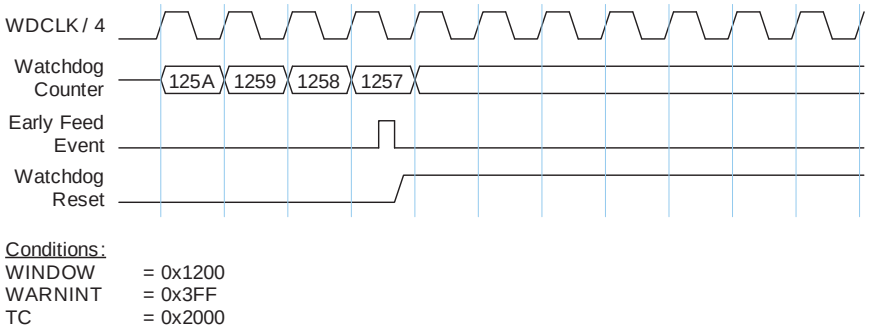


Fig 64. Correct Watchdog Feed with Windowed Mode Enabled

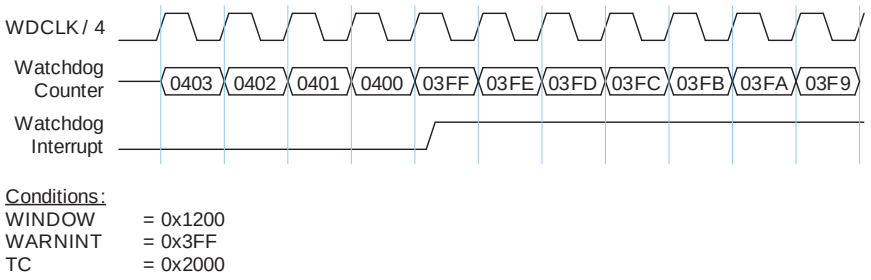


Fig 65. Watchdog Warning Interrupt

18.1 How to read this chapter

The system tick timer (SysTick timer) is part of the ARM Cortex-M0 core and is identical for all LPC11Uxx.

18.2 Basic configuration

The system tick timer is configured using the following registers:

1. Pins: The system tick timer uses no external pins.
2. Power: The system tick timer is enabled through the SysTick control register (Table 422). The system tick timer clock is fixed to half the frequency of the system clock.
3. Enable the clock source for the SysTick timer in the SYST_CSR register (Table 422).

18.3 Features

- Simple 24-bit timer.
- Uses dedicated exception vector.
- Clocked internally by the system clock or the system clock/2.

18.4 General description

The block diagram of the SysTick timer is shown below in the Figure 66.

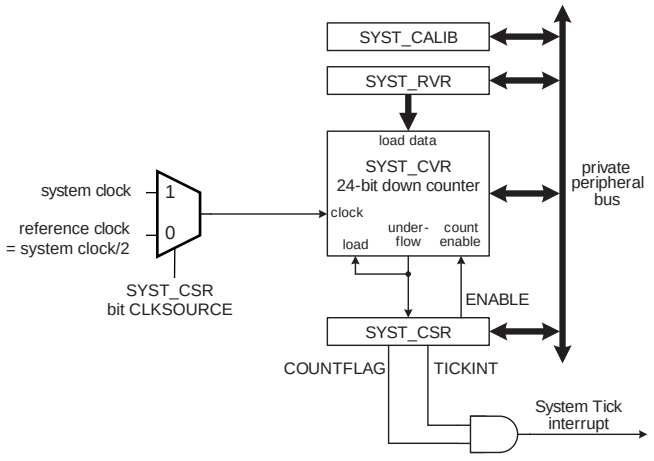


Fig 66. System tick timer block diagram

The SysTick timer is an integral part of the Cortex-M0. The SysTick timer is intended to generate a fixed 10 millisecond interrupt for use by an operating system or other system management software.

Since the SysTick timer is a part of the Cortex-M0, it facilitates porting of software by providing a standard timer that is available on Cortex-M0 based devices. The SysTick timer can be used for:

- An RTOS tick timer which fires at a programmable rate (for example 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the core clock.
- A simple counter. Software can use this to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNTFLAG bit-field in the control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

Refer to the *Cortex-M0 User Guide* for details.

18.5 Register description

The systick timer registers are located on the ARM Cortex-M0 private peripheral bus (see Figure 3), and are part of the ARM Cortex-M0 core peripherals. For details, see Section 23.5.4.

Table 318. Register overview: SysTick timer (base address 0xE000 E000)

Name	Access	Address offset	Description	Reset value[1]	Reference
SYST_CSR	R/W	0x010	System Timer Control and status register	0x000 0000	Table 319
SYST_RVR	R/W	0x014	System Timer Reload value register	0	Table 320
SYST_CVR	R/W	0x018	System Timer Current value register	0	Table 321
SYST_CALIB	R/W	0x01C	System Timer Calibration value register	0x4	Table 322

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

18.5.1 System Timer Control and status register

The SYST_CSR register contains control information for the SysTick timer and provides a status flag. This register is part of the ARM Cortex-M0 core system timer register block. For a bit description of this register, see Section 23.5.4.

This register determines the clock source for the system tick timer.

Table 319. SysTick Timer Control and status register (SYST_CSR - 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 system clock (CPU) clock is selected. When 0, the system clock/2 is selected as the reference clock.	0
15:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
16	COUNTFLAG	Returns 1 if the SysTick timer counted to 0 since the last read of this register.	0
31:17	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

18.5.2 System Timer Reload value register

The SYST_RVR register is set to the value that will be loaded into the SysTick timer whenever it counts down to zero. This register is loaded by software as part of timer initialization. The SYST_CALIB register may be read and used as the value for SYST_RVR register if the CPU is running at the frequency intended for use with the SYST_CALIB value.

Table 320. System Timer Reload value register (SYST_RVR - 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

18.5.3 System Timer Current value register

The SYST_CVR register returns the current count from the System Tick counter when it is read by software.

Table 321. System Timer Current value register (SYST_CVR - 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

18.5.4 System Timer Calibration value register (SYST_CALIB - 0xE000 E01C)

The value of the SYST_CALIB register is driven by the value of the SYSTCKCAL register in the system configuration block (see Table 36).

Table 322. System Timer Calibration value register (SYST_CALIB - 0xE000 E01C) bit description

Bit	Symbol	Value	Description	Reset value
23:0	TENMS		See Table 425.	0x4
29:24	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	SKEW		See Table 425.	0
31	NOREF		See Table 425.	0

18.6 Functional description

The SysTick 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 SysTick timer is clocked from the CPU clock (the system clock, see Figure 3) or from the reference clock, which is fixed to half the frequency of the CPU clock. In order to generate recurring interrupts at a specific interval, the SYST_RVR register must be initialized with the correct value for the desired interval. A default value is provided in the SYST_CALIB register and may be changed by software. The default value gives a 10 millisecond interrupt rate if the CPU clock is set to 50 MHz.

18.7 Example timer calculations

To use the system tick timer, do the following:

1. Program the SYST_RVR register with the reload value RELOAD to obtain the desired time interval.
2. Clear the SYST_CVR register by writing to it. This ensures that the timer will count from the SYST_RVR value rather than an arbitrary value when the timer is enabled.
3. Program the SYST_SCR register with the value 0x7 which enables the SysTick timer and the SysTick timer interrupt.

The following example illustrates selecting the SysTick timer reload value to obtain a 10 ms time interval with the LPC11Uxx system clock set to 50 MHz.

Example (system clock = 50 MHz)

The system tick clock = system clock = 50 MHz. Bit CLKSOURCE in the SYST_CSR register set to 1 (system clock).

RELOAD = (system tick clock frequency $\times$ 10 ms) -1 = (50 MHz $\times$ 10 ms) -1 = 500000 -1 = 499999 = 0x0007A11F.

19.1 How to read this chapter

The ADC block is identical for all LPC11Uxx parts.

19.2 Basic configuration

The ADC is configured using the following registers:

1. Pins: The ADC pin functions are configured in the IOCON register block (Section 7.4).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 13 (Table 23). Power to the ADC is controlled through the PDRUNCFG register (Table 46).

Remark: Basic clocking for the A/D converters is determined by the APB clock (PCLK). A programmable divider is included in the A/D converter to scale this clock to the 4.5 MHz (max) clock needed by the successive approximation process. An accurate conversion requires 11 clock cycles.

19.3 Features

- 10-bit successive approximation Analog-to-Digital Converter (ADC).
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range 0 to 3.6 V. Do not exceed the V_{DD} voltage level.
- 10-bit conversion time $\geq$ 2.44 μs.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.
- Individual result registers for each A/D channel to reduce interrupt overhead.

19.4 Pin description

Table 323 gives a brief summary of the ADC related pins.

Table 323. ADC pin description

Pin	Type	Description
AD[7:0]	Input	**Analog Inputs.** The A/D converter cell can measure the voltage on any of these input signals. **Remark:** While the pins are 5 V tolerant in digital mode, the maximum input voltage must not exceed V_{DD} when the pins are configured as analog inputs.
V_{DD}	Input	V_{REF}; Reference voltage.

The ADC function must be selected via the IOCON registers in order to get accurate voltage readings on the monitored pin. For a pin hosting an ADC input, it is not possible to have a have a digital function selected and yet get valid ADC readings. An inside circuit disconnects ADC hardware from the associated pin whenever a digital function is selected on that pin.

19.5 Register description

The ADC contains registers organized as shown in Table 324.

Table 324. Register overview: ADC (base address 0x4001 C000)

Name	Access	Address offset	Description	Reset Value[1]	Reference
CR	R/W	0x000	A/D Control Register. The CR register must be written to select the operating mode before A/D conversion can occur.	0x0000 0000	Table 325
GDR	R/W	0x004	A/D Global Data Register. Contains the result of the most recent A/D conversion.	NA	Table 326
-	-	0x008	Reserved.	-	-
INTEN	R/W	0x00C	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.	0x0000 0100	Table 327
DR0	R/W	0x010	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0	NA	Table 328
DR1	R/W	0x014	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	NA	Table 328
DR2	R/W	0x018	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	NA	Table 328
DR3	R/W	0x01C	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	NA	Table 328
DR4	R/W	0x020	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	NA	Table 328
DR5	R/W	0x024	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	NA	Table 328
DR6	R/W	0x028	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	NA	Table 328
DR7	R/W	0x02C	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 328
STAT	RO	0x030	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 flag.	0	Table 329

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

19.5.1 A/D Control Register (CR - 0x4001 C000)

The A/D Control Register provides bits to select A/D channels to be converted, A/D timing, A/D modes, and the A/D start trigger.

Table 325. A/D Control Register (CR - address 0x4001 C000) bit description

Bit	Symbol	Value	Description	Reset Value
7:0	SEL		Selects which of the AD7:0 pins is (are) to be sampled and converted. Bit 0 selects Pin AD0, bit 1 selects pin AD1,..., and bit 7 selects pin AD7. In software-controlled mode (BURST = 0), only one channel can be selected, i.e. only one of these bits should be 1. In hardware scan mode (BURST = 1), any numbers of channels can be selected, i.e any or all bits can be set to 1. If all bits are set to 0, channel 0 is selected automatically (SEL = 0x01).	0x00
15:8	CLKDIV		The APB clock (PCLK) is divided by CLKDIV +1 to produce the clock for the ADC, which should be less than or equal to 4.5 MHz. Typically, software should program the smallest value in this field that yields a clock of 4.5 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	0
16	BURST		Burst mode **Remark:** If BURST is set to 1, the ADGINTEN bit in the INTEN register (Table 327) must be set to 0.	0
		0	Software-controlled mode: Conversions are software-controlled and require 11 clocks.	
		1	Hardware scan mode: The AD converter does repeated conversions at the rate selected by the CLKS field, scanning (if necessary) through the pins selected by 1s in the SEL field. The first conversion after the start corresponds to the least-significant bit set to 1 in the SEL field, then the next higher bits (pins) set to 1 are scanned if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion in progress when this bit is cleared will be completed. **Important:** START bits must be 000 when BURST = 1 or conversions will not start.	
19:17	CLKS		This field selects the number of clocks used for each conversion in Burst mode, and the number of bits of accuracy of the result in the LS bits of ADDR, between 11 clocks (10 bits) and 4 clocks (3 bits).	000
		0x0	11 clocks / 10 bits	
		0x1	10 clocks / 9 bits	
		0x2	9 clocks / 8 bits	
		0x3	8 clocks / 7 bits	
		0x4	7 clocks / 6 bits	
		0x5	6 clocks / 5 bits	
		0x6	5 clocks / 4 bits	
		0x7	4 clocks / 3 bits	
23:20	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 325. A/D Control Register (CR - address 0x4001 C000) bit description

Bit	Symbol	Value	Description	Reset Value
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
		0x0	No start (this value should be used when clearing PDN to 0).	
		0x1	Start conversion now.	
		0x2	Start conversion when the edge selected by bit 27 occurs on PIO0_2/SSEL/CT16B0_CAP0.	
		0x3	Start conversion when the edge selected by bit 27 occurs on PIO1_5/DIR/CT32B0_CAP0.	
		0x4	Start conversion when the edge selected by bit 27 occurs on CT32B0_MAT0[1].	
		0x5	Start conversion when the edge selected by bit 27 occurs on CT32B0_MAT1[1].	
		0x6	Start conversion when the edge selected by bit 27 occurs on CT16B0_MAT0[1].	
		0x7	Start conversion when the edge selected by bit 27 occurs on CT16B0_MAT1[1].	
27	EDGE		This bit is significant only when the START field contains 010-111. In these cases:	0
		0	Start conversion on a rising edge on the selected CAP/MAT signal.	
		1	Start conversion on a falling edge on the selected CAP/MAT signal.	
31:28	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] Note that this does not require that the timer match function appear on a device pin.

19.5.2 A/D Global Data Register (GDR - 0x4001 C004)

The A/D Global Data Register contains the result of the most recent A/D conversion. This includes the data, DONE, and Overrun flags, and the number of the A/D channel to which the data relates.

Table 326. A/D Global Data Register (GDR - address 0x4001 C004) bit description

Bit	Symbol	Description	Reset Value
5:0	-	Reserved. These bits always read as zeros.	0
15:6	V_VREF	When DONE is 1, this field contains a binary fraction representing the voltage on the ADn pin selected by the SEL field, divided by the voltage on the V_{DD} pin. Zero in the field indicates that the voltage on the ADn pin was less than, equal to, or close to that on V_{SS}, while 0x3FF indicates that the voltage on ADn was close to, equal to, or greater than that on V_{REF}.	X
23:16	-	Reserved. These bits always read as zeros.	0
26:24	CHN	These bits contain the channel from which the result bits V_VREF were converted.	X
29:27	-	Reserved. These bits always read as zeros.	0
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 V_VREF bits.	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

19.5.3 A/D Interrupt Enable Register (INTEN - 0x4001 C00C)

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 327. A/D Interrupt Enable Register (INTEN - address 0x4001 C00C) bit description

Bit	Symbol	Description	Reset Value
7:0	ADINTEN	These bits allow control over which A/D channels generate interrupts for conversion completion. When bit 0 is one, completion of a conversion on A/D channel 0 will generate an interrupt, when bit 1 is one, completion of a conversion on A/D channel 1 will generate an interrupt, etc.	0x00
8	ADGINTEN	When 1, enables the global DONE flag in ADDR to generate an interrupt. When 0, only the individual A/D channels enabled by ADINTEN 7:0 will generate interrupts. **Remark:** This bit must be set to 0 in burst mode (BURST = 1 in the CR register).	1
31:9	-	Reserved. Unused, always 0.	0

19.5.4 A/D Data Registers (DR0 to DR7 - 0x4001 C010 to 0x4001 C02C)

The A/D Data Register hold the result when an A/D conversion is complete, and also include the flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

Table 328. A/D Data Registers (DR0 to DR7 - addresses 0x4001 C010 to 0x4001 C02C) bit description

Bit	Symbol	Description	Reset Value
5:0	-	Reserved.	0
15:6	V_VREF	When DONE is 1, this field contains a binary fraction representing the voltage on the ADn pin, divided by the voltage on the V_{REF} pin. Zero in the field indicates that the voltage on the ADn pin was less than, equal to, or close to that on V_{REF}, while 0x3FF indicates that the voltage on AD input was close to, equal to, or greater than that on V_{REF}.	NA
29:16	-	Reserved.	0
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 V_VREF 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.	0

19.5.5 A/D Status Register (STAT - 0x4001 C030)

The A/D Status register allows checking the status of all A/D channels simultaneously. The DONE and OVERRUN flags appearing in the DRn 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 329. A/D Status Register (STAT - address 0x4001 C030) bit description

Bit	Symbol	Description	Reset Value
7:0	DONE	These bits mirror the DONE status flags that appear in the result register for each A/D channel n.	0
15:8	OVERRUN	These bits mirror the OVERRRUN status flags that appear in the result register for each A/D channel n. Reading ADSTAT allows checking the status of all A/D channels simultaneously.	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. Unused, always 0.	0

19.6 Operation

19.6.1 Hardware-triggered conversion

If the BURST bit in the ADCR0 is 0 and the START field contains 010-111, the A/D converter will start a conversion when a transition occurs on a selected pin or timer match signal.

19.6.2 Interrupts

An interrupt is requested to the interrupt controller when the ADINT bit in the ADSTAT register is 1. The ADINT bit is one when any of the DONE bits of A/D channels that are enabled for interrupts (via the ADINTEN register) are one. Software can use the Interrupt Enable bit in the interrupt controller that corresponds to the ADC to control whether this results in an interrupt. The result register for an A/D channel that is generating an interrupt must be read in order to clear the corresponding DONE flag.

19.6.3 Accuracy vs. digital receiver

While the A/D converter can be used to measure the voltage on any ADC input pin, regardless of the pin's setting in the IOCON block, selecting the ADC in the IOCON registers function improves the conversion accuracy by disabling the pin's digital receiver (see also Section 7.3.7).

20.1 How to read this chapter

See Table 330 for different flash configurations and functionality.

Table 330. LPC11Uxx flash configurations

Type number	Flash	EEPROM	ISP via UART	ISP via USB MSC
LPC11U12FBD48/201	16 kB	no	yes	no
LPC11U12FHN33/201	16 kB	no	yes	no
LPC11U13FBD48/201	24 kB	no	yes	no
LPC11U14FBD48/201	32 kB	no	yes	no
LPC11U14FHN33/201	32 kB	no	yes	no
LPC11U14FHI33/201	32 kB	no	yes	no
LPC11U14FET48/201	32 kB	no	yes	no
LPC11U23FBD48/301	24 kB	1 kB	yes	yes
LPC11U24FHI33/301	32 kB	2 kB	yes	yes
LPC11U24FBD48/301	32 kB	2 kB	yes	yes
LPC11U24FET48/301	32 kB	2 kB	yes	yes
LPC11U24FHN33/401	32 kB	4 kB	yes	yes
LPC11U24FBD48/401	32 kB	4 kB	yes	yes
LPC11U24FBD64/401	32 kB	4 kB	yes	yes

Remark: In addition to the ISP and IAP commands, a register can be accessed in the flash controller block to configure flash memory access times, see Section 20.16.4.1.

20.2 Bootloader

The bootloader controls initial operation after reset and also provides the means to program 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.

The bootloader version can be read by ISP/IAP calls (see Section 20.13.12 or Section 20.14.6).

20.3 Features

- In-System Programming: In-System programming (ISP) is programming or reprogramming the on-chip flash memory, using the bootloader software and the UART 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 access times can be configured through a register in the flash controller block.

- Erase time for one sector is 100 ms ± 5%. Programming time for one block of 256 bytes is 1 ms ± 5%.
- The LPC11U2x supports ISP from the USB port through enumeration as a Mass Storage Class (MSC) Device when connected to a USB host interface (Windows operating system only).

20.4 Description

The bootloader code is executed every time the part is powered on or reset (see Figure 67). The loader can execute the ISP command handler or the user application code. A LOW level during reset at the PIO0_1 pin is considered an external hardware request to start the ISP command handler (or, on the LPC11U2x, the USB device handler) without checking for a valid user code first.

Assuming that power supply pins are at their nominal levels when the rising edge on RESET pin is generated, it may take up to 3 ms before PIO0_1 is sampled and the decision whether to continue with user code or ISP handler is made. If PIO0_1 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 (PIO0_1 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.

For the LPC11U2x parts, the state of PIO0_3 determines whether the UART or USB interface will be used:

- If PIO0_3 is sampled HIGH, the bootloader connects the LPC1U2x as a MSC USB device to a PC host. The LPC11U2x flash memory space is represented as a drive in the host's Windows operating system.
- If PIO0_3 is sampled LOW, the bootloader configures the UART serial port using pins PIO0_18 and PIO0_19 for RXD and TXD and calls the ISP command handler.

Remark: The sampling of pin PIO0_1 can be disabled through programming flash location 0x0000 02FC (see Section 20.12.1).

20.5 Memory map after any reset

The boot block is 16 kB in size and is located in the memory region starting from the address 0x1FFF 0000. The bootloader is designed to run from this memory area, but both the ISP and IAP software use parts of the on-chip RAM. The RAM usage is described later in this chapter. The interrupt vectors residing in the boot block of the on-chip flash memory also become active after reset, i.e., the bottom 512 bytes of the boot block are also visible in the memory region starting from the address 0x0000 0000.

20.6 Flash content protection mechanism

The LPC11Uxx is equipped with the Error Correction Code (ECC) capable Flash memory. The purpose of an error correction module is twofold. Firstly, it decodes data words read from the memory into output data words. Secondly, it encodes data words to be written to the memory. The error correction capability consists of single bit error correction with Hamming code.

The operation of ECC is transparent to the running application. The ECC content itself is stored in a flash memory not accessible by user's code to either read from it or write into it on its own. A byte of ECC corresponds to every consecutive 128 bits of the user accessible Flash. Consequently, Flash bytes from 0x0000 0000 to 0x0000 000F are protected by the first ECC byte, Flash bytes from 0x0000 0010 to 0x0000 001F are protected by the second ECC byte, etc.

Whenever the CPU requests a read from user's Flash, both 128 bits of raw data containing the specified memory location and the matching ECC byte are evaluated. If the ECC mechanism detects a single error in the fetched data, a correction will be applied before data are provided to the CPU. When a write request into the user's Flash is made, write of user specified content is accompanied by a matching ECC value calculated and stored in the ECC memory.

When a sector of Flash memory is erased, the corresponding ECC bytes are also erased. Once an ECC byte is written, it can not be updated unless it is erased first. Therefore, for the implemented ECC mechanism to perform properly, data must be written into the flash memory in groups of 16 bytes (or multiples of 16), aligned as described above.

20.7 Criterion for Valid User Code

The reserved ARM Cortex-M0 exception vector location 7 (offset 0x0000 001C 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 bootloader 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 the serial port (UART).

If the UART is selected, 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 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. 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. In USART ISP mode, the LPC11Uxx is clocked by the IRC and the crystal frequency is ignored.

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 20.13 “ISP commands” on page 361.

20.8 ISP/IAP 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.

20.8.1 ISP command format

"Command Parameter_0 Parameter_1 ... Parameter_n<CR><LF>" "Data" (Data only for Write commands).

20.8.2 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).

20.8.3 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.

20.8.4 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 (stop) is sent to stop the flow of data. Data flow is resumed by sending the ASCII control character DC1 (start). The host should also support the same flow control scheme.

20.8.5 ISP command abort

Commands can be aborted by sending the ASCII control character "ESC". 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.

20.8.6 Interrupts during ISP

The boot block interrupt vectors located in the boot block of the flash are active after any reset.

20.8.7 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.

20.8.8 RAM used by ISP command handler

ISP commands use on-chip RAM from 0x1000 017C to 0x1000 025B. 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 bytes. The maximum stack usage is 256 bytes and grows downwards.

20.8.9 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 and grows downwards.

20.9 USB communication protocol

The LPC11U2x is enumerated as a Mass Storage Class (MSC) device to a PC or another embedded system. In order to connect via the USB interface, the LPC11U2x must use the external crystal at a frequency of 12 MHz. The MSC device presents an easy integration with the PC's Windows operating system. The LPC11U2x flash memory space is represented as a drive in the host file system. The entire available user flash is mapped to a file of the size of the LPC11U2x flash in the host's folder with the default name 'firmware.bin'. The 'firmware.bin' file can be deleted and a new file can be copied into the directory, thereby updating the user code in flash. Note that the filename of the new flash image file is not important. After a reset or a power cycle, the new file is visible in the host's file system under it's default name 'firmware.bin'.

Remark: USB ISP commands are supported for the Windows operating system only.

The code read protection (CRP, see Table 331) level determines how the flash is reprogrammed:

If CRP1 or CRP2 is enabled, the user flash is erased when the file is deleted.

If CRP1 is enabled or no CRP is selected, the user flash is erased and reprogrammed when the new file is copied. However, only the area occupied by the new file is erased and reprogrammed.

Remark: The only Windows commands supported for the LPC11U2x flash image folder are copy and delete.

Three Code Read Protection (CRP) levels can be enabled for flash images updated through USB (see Section 20.12 for details). The volume label on the MSCD indicates the CRP status.

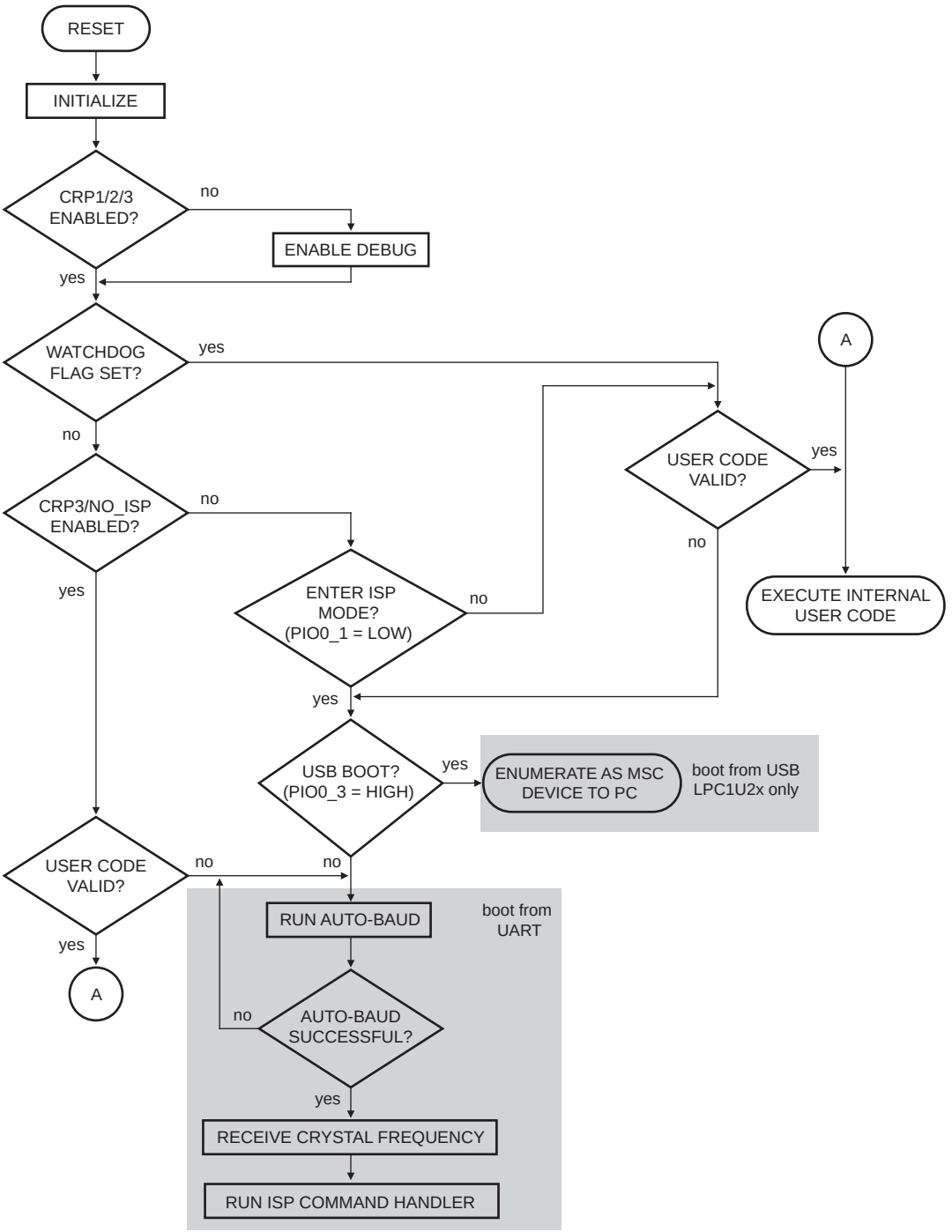
Table 331. CRP levels for USB boot images

CRP status	Volume label	Description
No CRP	CRP DISABLD	The user flash can be read or written.
CRP1	CRP1 ENABLD	The user flash content cannot be read but can be updated. The flash memory sectors are updated depending on the new firmware image.
CRP2	CRP2 ENABLD	The user flash content cannot be read but can be updated. The entire user flash memory is erased before writing the new firmware image.
CRP3	CRP3 ENABLD	The user flash content cannot be read or updated. The bootloader always executes the user application if valid.

20.9.1 Usage note

When programming flash images via Flash Magic or Serial Wire Debugger (SWD), the user code valid signature is automatically inserted by the programming utility. When using USB ISP, the user code valid signature must be either part of the vector table, or the axf or binary file must be post-processed to insert the checksum.

20.10 Boot process flowchart



(1) For details on handling the crystal frequency, see Section 20.14.8.

(2) For details on available ISP commands based on the CRP settings, see Section 20.12.

Fig 67. Boot process flowchart

20.11 Sector numbers

Some IAP and ISP commands operate on sectors and specify sector numbers. The following table shows the correspondence between sector numbers and memory addresses for LPC11Uxx devices.

Table 332. LPC11Uxx flash sectors

Sector number	Sector size [kB]	Address range	LPC11U12	LPC11U13/ LPC11U23	LPC11U14/ LPC11U24
0	4	0x0000 0000 - 0x0000 0FFF	yes	yes	yes
1	4	0x0000 1000 - 0x0000 1FFF	yes	yes	yes
2	4	0x0000 2000 - 0x0000 2FFF	yes	yes	yes
3	4	0x0000 3000 - 0x0000 3FFF	yes	yes	yes
4	4	0x0000 4000 - 0x0000 4FFF	-	yes	yes
5	4	0x0000 5000 - 0x0000 5FFF	-	yes	yes
6	4	0x0000 6000 - 0x0000 6FFF	-	-	yes
7	4	0x0000 7000 - 0x0000 7FFF	-	-	yes

20.12 Code Read Protection (CRP)

Code Read Protection is a mechanism that allows the 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 333. Code Read Protection (CRP) options

Name	Pattern programmed in 0x0000 02FC	Description
NO_ISP	0x4E69 7370	Prevents sampling of pin PIO0_1 for entering ISP mode. PIO0_1 is available for other uses.
CRP1	0x12345678	Access to chip via the SWD pins is disabled. This mode allows partial flash update using the following ISP commands and restrictions: • Write to RAM command should not access RAM below 0x1000 0300. Access to addresses below 0x1000 0200 is disabled. • Copy RAM to flash command can not write to Sector 0. • Erase command can erase Sector 0 only when all sectors are selected for erase. • Compare command is disabled. • Read Memory command is disabled. This mode is useful when CRP is required and flash field updates are needed but all sectors can not be erased. Since compare command is disabled in case of partial updates the secondary loader should implement checksum mechanism to verify the integrity of the flash.
CRP2	0x87654321	Access to chip via the SWD pins is disabled. The following ISP commands are disabled: • Read Memory • Write to RAM • Go • Copy RAM to flash • Compare When CRP2 is enabled the ISP erase command only allows erasure of all user sectors.
CRP3	0x43218765	Access to chip via the SWD pins is disabled. ISP entry by pulling PIO0_1 LOW is disabled if a valid user code is present in flash sector 0. This mode effectively disables ISP override using PIO0_1 pin. It is up to the user's application to provide a flash update mechanism using IAP calls or call reinvoke ISP command to enable flash update via UART0. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Table 334. Code Read Protection hardware/software interaction

CRP option	User Code Valid	PIO0_1 pin at reset	SWD enabled	LPC11U1x enters ISP mode	partial flash Update in ISP mode
None	No	x	Yes	Yes	Yes
None	Yes	High	Yes	No	NA
None	Yes	Low	Yes	Yes	Yes
CRP1	Yes	High	No	No	NA
CRP1	Yes	Low	No	Yes	Yes

Table 334. Code Read Protection hardware/software interaction *…continued*

CRP option	User Code Valid	PIO0_1 pin at reset	SWD enabled	LPC11U1x enters ISP mode	partial flash Update in ISP mode
CRP2	Yes	High	No	No	NA
CRP2	Yes	Low	No	Yes	No
CRP3	Yes	x	No	No	NA
CRP1	No	x	No	Yes	Yes
CRP2	No	x	No	Yes	No
CRP3	No	x	No	Yes	No

Table 335. ISP commands allowed for different CRP levels

ISP command	CRP1	CRP2	CRP3 (no entry in ISP mode allowed)
Unlock	yes	yes	n/a
Set Baud Rate	yes	yes	n/a
Echo	yes	yes	n/a
Write to RAM	yes; above 0x1000 0300 only	no	n/a
Read Memory	no	no	n/a
Prepare sector(s) for write operation	yes	yes	n/a
Copy RAM to flash	yes; not to sector 0	no	n/a
Go	no	no	n/a
Erase sector(s)	yes; sector 0 can only be erased when all sectors are erased.	yes; all sectors only	n/a
Blank check sector(s)	no	no	n/a
Read Part ID	yes	yes	n/a
Read Boot code version	yes	yes	n/a
Compare	no	no	n/a
ReadUID	yes	yes	n/a

In case a 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.

20.12.1 ISP entry protection

In addition to the three CRP modes, the user can prevent the sampling of pin PIO0_1 for entering ISP mode and thereby release pin PIO0_1 for other uses. This is called the NO_ISP mode. The NO_ISP mode can be entered by programming the pattern 0x4E69 7370 at location 0x0000 02FC.

The NO_ISP mode is identical to the CRP3 mode except for SWD access, which is allowed in NO_ISP mode but disabled in CRP3 mode. The NO_ISP mode does not offer any code protection.

20.13 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 336. ISP command summary

ISP Command	Usage	Described in
Unlock	U <Unlock Code>	Table 337
Set Baud Rate	B <Baud Rate> <stop bit>	Table 338
Echo	A <setting>	Table 339
Write to RAM	W <start address> <number of bytes>	Table 340
Read Memory	R <address> <number of bytes>	Table 341
Prepare sector(s) for write operation	P <start sector number> <end sector number>	Table 342
Copy RAM to flash	C <Flash address> <RAM address> <number of bytes>	Table 343
Go	G <address> <Mode>	Table 344
Erase sector(s)	E <start sector number> <end sector number>	Table 345
Blank check sector(s)	I <start sector number> <end sector number>	Table 346
Read Part ID	J	Table 347
Read Boot code version	K	Table 349
Compare	M <address1> <address2> <number of bytes>	Table 350
ReadUID	N	Table 351

20.13.1 Unlock <Unlock code>

Table 337. ISP Unlock command

Command	U
Input	Unlock code: 23130_{10}
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.

20.13.2 Set Baud Rate <Baud Rate> <stop bit>

Table 338. ISP Set Baud Rate command

Command	B
Input	Baud Rate: 9600 | 19200 | 38400 | 57600 | 115200 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.

20.13.3 Echo <setting>

Table 339. 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.

20.13.4 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 340. ISP Write to RAM command

Command	W
Input	**Start Address:** RAM address where data bytes are to be written. This address should be a word boundary. **Number of Bytes:** 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 is enabled.
Example	"W 268436224 4<CR><LF>" writes 4 bytes of data to address 0x1000 0300.

20.13.5 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 341. ISP Read Memory command

Command	R
Input	**Start Address:** Address from where data bytes are to be read. This address should be a word boundary. **Number of Bytes:** 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 code read protection is enabled.
Example	"R 268435456 4<CR><LF>" reads 4 bytes of data from address 0x1000 0000.

20.13.6 Prepare sector(s) for write operation <start sector number> <end sector number>

This command makes flash write/erase operation a two step process.

Table 342. ISP Prepare sector(s) for write operation command

Command	P
Input	**Start Sector Number** **End Sector Number:** 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. The boot block can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.
Example	"P 0 0<CR><LF>" prepares the flash sector 0.

20.13.7 Copy RAM to flash <Flash address> <RAM address> <no of bytes>

When writing to the flash, the following limitations apply:

1. The smallest amount of data that can be written to flash by the copy RAM to flash command is 256 byte (equal to one page).
2. One page consists of 16 flash words (lines), and the smallest amount that can be modified per flash write is one flash word (one line). This limitation follows from the application of ECC to the flash write operation, see Section 20.6.
3. To avoid write disturbance (a mechanism intrinsic to flash memories), an erase should be performed after following 16 consecutive writes inside the same page. Note that the erase operation then erases the entire sector.

   Remark: Once a page has been written to 16 times, it is still possible to write to other pages within the same sector without performing a sector erase (assuming that those pages have been erased previously).

Table 343. ISP Copy command

Command	C
Input	**Flash Address(DST):** Destination flash address where data bytes are to be written. The destination address should be a 256 byte boundary. **RAM Address(SRC):** Source RAM address from where data bytes are to be read. **Number of Bytes:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096.
Return Code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not on word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR WRITE_OPERATION | BUSY | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	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. The boot block cannot be written by this command. This command is blocked when code read protection is enabled. Also see Section 20.6 for the number of bytes that can be written.
Example	"C 0 268467504 512<CR><LF>" copies 512 bytes from the RAM address 0x1000 0800 to the flash address 0.

20.13.8 Go <address> <mode>

Table 344. ISP Go command

Command	G
Input	**Address:** Flash or RAM address from which the code execution is to be started. This address should be on a word boundary. **Mode:** T (Execute program in Thumb Mode) | A (Execute program in ARM mode).
Return Code	CMD_SUCCESS | ADDR_ERROR | ADDR_NOT_MAPPED | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	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 code read protection is enabled. The command must be used with an address of 0x0000 0200 or greater.
Example	"G 512 T<CR><LF>" branches to address 0x0000 0200 in Thumb mode.

20.13.9 Erase sector(s) <start sector number> <end sector number>

Table 345. ISP Erase sector command

Command	E
Input	**Start Sector Number** **End Sector Number:** 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. The boot block can not be erased using this command. This command only allows erasure of all user sectors when the code read protection is enabled.
Example	"E 2 3<CR><LF>" erases the flash sectors 2 and 3.

20.13.10 Blank check sector(s) <sector number> <end sector number>

Table 346. ISP Blank check sector command

Command	I
Input	**Start Sector Number:** **End Sector Number:** 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. **Blank check on sector 0 always fails as first 64 bytes are re-mapped to flash boot block.** When CRP is enabled, the blank check command returns 0 for the offset and value of sectors which are not blank. Blank sectors are correctly reported irrespective of the CRP setting.
Example	"I 2 3<CR><LF>" blank checks the flash sectors 2 and 3.

20.13.11 Read Part Identification number

Table 347. ISP Read Part Identification command

Command	J
Input	None.
Return Code	CMD_SUCCESS followed by part identification number in ASCII (see Table 348 "LPC11Uxx device identification numbers").
Description	This command is used to read the part identification number.

Table 348. LPC11Uxx device identification numbers

Device	Hex coding
LPC11U12FHN33/201	0x095C 802B/0x295C 802B
LPC11U12FBD48/201	0x095C 802B/0x295C 802B
LPC11U13FBD48/201	0x097A 802B/0x297A 802B
LPC11U14FHN33/201	0x0998 802B/0x2998 802B
LPC11U14FHI33/201	0x2998 802B
LPC11U14FBD48/201	0x0998 802B/0x2998 802B
LPC11U14FET48/201	0x0998 802B/0x2998 802B
LPC11U23FBD48/301	0x2972 402B
LPC11U24FHI33/301	0x2988 402B
LPC11U24FBD48/301	0x2988 402B
LPC11U24FET48/301	0x2988 402B
LPC11U24FHN33/401	0x2980 002B
LPC11U24FBD48/401	0x2980 002B
LPC11U24FBD64/401	0x2980 002B

20.13.12 Read Boot code version number

Table 349. 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.

20.13.13 Compare <address1> <address2> <no of bytes>

Table 350. ISP Compare command

Command	M
Input	**Address1 (DST):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Address2 (SRC):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Number of Bytes:** Number of bytes to be compared; should be a multiple of 4.

Table 350. 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. **Compare result may not be correct when source or destination address contains any of the first 512 bytes starting from address zero. First 512 bytes are re-mapped to boot ROM**
Example	"M 8192 268468224 4<CR><LF>" compares 4 bytes from the RAM address 0x1000 8000 to the 4 bytes from the flash address 0x2000.

20.13.14 ReadUID

Table 351. ReadUID command

Command	N
Input	None
Return Code	CMD_SUCCESS followed by four 32-bit words of a unique serial number in ASCII format. The word sent at the lowest address is sent first.
Description	This command is used to read the unique ID.

20.13.15 ISP Return Codes

Table 352. 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 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 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.

Table 352. ISP Return Codes Summary

Return Code	Mnemonic	Description
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.

20.14 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 of the IAP command is returned in the result 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 the number of results are more than number of parameters. Parameter passing is illustrated in the Figure 68.

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 "ReadUID" command. The command handler sends the status code INVALID_COMMAND when an undefined command is received. The IAP routine resides at 0x1FFF 1FF0 location and it is thumb code.

The IAP function could be called in the following way using C.

Define the IAP location entry point. Since the 0th bit of the IAP location is set there will be a change to Thumb instruction set when the program counter branches to this address.

```
#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 result[4];
```

or

```
unsigned long * command;
unsigned long * result;
```

```
command=(unsigned long *) 0x...
result= (unsigned long *) 0x...
```

Define 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, result);
```

Up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively (see the *ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05)*. 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 353. IAP Command Summary

IAP Command	Command Code	Described in
Prepare sector(s) for write operation	50 (decimal)	Table 354
Copy RAM to flash	51 (decimal)	Table 355
Erase sector(s)	52 (decimal)	Table 356
Blank check sector(s)	53 (decimal)	Table 357
Read Part ID	54 (decimal)	Table 358
Read Boot code version	55 (decimal)	Table 359
Compare	56 (decimal)	Table 360
Reinvoke ISP	57 (decimal)	Table 361
Read UID	58 (decimal)	Table 362
EEPROM Write	61(decimal)	Table 363
EEPROM Read	62(decimal)	Table 364

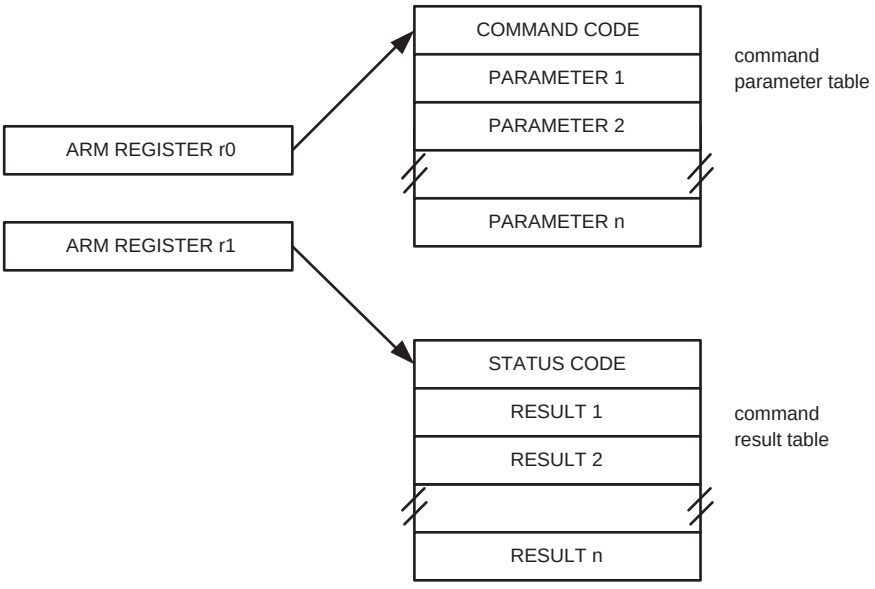


Fig 68. IAP parameter passing

20.14.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two step process.

Table 354. IAP Prepare sector(s) for write operation command

Command	Prepare sector(s) for write operation
Input	**Command code: 50 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR
Result	None
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. The boot sector can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.

20.14.2 Copy RAM to flash

See Section 20.13.7 for limitations on the write-to-flash process.

Table 355. IAP Copy RAM to flash command

Command	Copy RAM to flash
Input	**Command code: 51 (decimal)** **Param0(DST):** Destination flash address where data bytes are to be written. This address should be a 256 byte boundary. **Param1(SRC):** Source RAM address from which data bytes are to be read. This address should be a word boundary. **Param2:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096. **Param3:** System Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not a word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | BUSY
Result	None
Description	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. The boot sector can not be written by this command. Also see Section 20.6 for the number of bytes that can be written.

20.14.3 Erase Sector(s)

Table 356. IAP Erase Sector(s) command

Command	Erase Sector(s)
Input	**Command code: 52 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number). **Param2:** System 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. The boot sector can not be erased by this command. To erase a single sector use the same "Start" and "End" sector numbers.

20.14.4 Blank check sector(s)

Table 357. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	**Command code: 53 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Return Code	CMD_SUCCESS | BUSY | SECTOR_NOT_BLANK | INVALID_SECTOR
Result	**Result0:** Offset of the first non blank word location if the Status Code is SECTOR_NOT_BLANK. **Result1:** 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.

20.14.5 Read Part Identification number

Table 358. IAP Read Part Identification command

Command	Read part identification number
Input	**Command code: 54 (decimal)** **Parameters:** None
Return Code	CMD_SUCCESS
Result	**Result0:** Part Identification Number.
Description	This command is used to read the part identification number.

20.14.6 Read Boot code version number

Table 359. IAP Read Boot Code version number command

Command	Read boot code version number
Input	**Command code: 55 (decimal)** **Parameters:** None
Return Code	CMD_SUCCESS
Result	**Result0:** Boot code version number. Read as <byte1(Major)>.<byte0(Minor)>
Description	This command is used to read the boot code version number.

20.14.7 Compare <address1> <address2> <no of bytes>

Table 360. IAP Compare command

Command	Compare
Input	**Command code: 56 (decimal)** **Param0(DST):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param1(SRC):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param2:** 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	**Result0:** 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. **The result may not be correct when the source or destination includes any of the first 512 bytes starting from address zero. The first 512 bytes can be re-mapped to RAM.**

20.14.8 Reinvoke ISP

Table 361. Reinvoke ISP

Command	Compare
Input	**Command code: 57 (decimal)**
Return Code	None
Result	**None.**
Description	This command is used to invoke the bootloader in ISP mode. It maps boot vectors, sets PCLK = CCLK, configures UART pins RXD and TXD, resets counter/timer CT32B1 and resets the U0FDR (see Table 230). This command may be used when a valid user program is present in the internal flash memory and the PIO0_1 pin is not accessible to force the ISP mode.

20.14.9 ReadUID

Table 362. IAP ReadUID command

Command	Compare
Input	**Command code: 58 (decimal)**
Return Code	CMD_SUCCESS
Result	**Result0:** The first 32-bit word (at the lowest address). **Result1:** The second 32-bit word. **Result2:** The third 32-bit word. **Result3:** The fourth 32-bit word.
Description	This command is used to read the unique ID.

20.14.10 Write EEPROM

Table 363. IAP Write EEPROM command

Command	Compare
Input	**Command code: 61 (decimal)**
Return Code	CMD_SUCCESS | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED
Result	**Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be written. **Param3:** System Clock Frequency (CCLK) in kHz.
Description	Data is copied from the RAM address to the EEPROM address. **Remark:** The top 64 bytes of the EEPROM memory are reserved and cannot be written to.

20.14.11 Read EEPROM

Table 364. IAP Read EEPROM command

Command	Compare
Input	**Command code: 62 (decimal)**
Return Code	CMD_SUCCESS | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED
Result	**Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be read. **Param3:** System Clock Frequency (CCLK) in kHz.
Description	Data is copied from the EEPROM address to the RAM address.

20.14.12 IAP Status Codes

Table 365. 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.

Table 365. IAP Status Codes Summary

Status Code	Mnemonic	Description
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.

20.15 Debug notes

20.15.1 Comparing flash images

Depending on the debugger used and the IDE debug settings, the memory that is visible when the debugger connects might be the boot ROM, the internal SRAM, or the flash. To help determine which memory is present in the current debug environment, check the value contained at flash address 0x0000 0004. This address contains the entry point to the code in the ARM Cortex-M0 vector table, which is the bottom of the boot ROM, the internal SRAM, or the flash memory respectively.

Table 366. Memory mapping in debug mode

Memory mapping mode	Memory start address visible at 0x0000 0004
Bootloader mode	0x1FFF 0000
User flash mode	0x0000 0000
User SRAM mode	0x1000 0000

20.15.2 Serial Wire Debug (SWD) flash programming interface

Debug tools can write parts of the flash image to RAM and then execute the IAP call "Copy RAM to flash" repeatedly with proper offset.

20.16 Register description

Table 367. Register overview: FMC (base address 0x4003 C000)

Name	Access	Address offset	Description	Reset value	Reference
FLASHCFG	R/W	0x010	Flash memory access time configuration register	-	Table 371
FMSSTART	R/W	0x020	Signature start address register	0	Table 372
FMSSTOP	R/W	0x024	Signature stop-address register	0	Table 373
FMSW0	R	0x02C	Word 0 [31:0]	-	Table 374
FMSW1	R	0x030	Word 1 [63:32]	-	Table 375
FMSW2	R	0x034	Word 2 [95:64]	-	Table 376
FMSW3	R	0x038	Word 3 [127:96]	-	Table 377
EEMSSTART	R/W	0x09C	EEPROM BIST start address register	0x0	Table 368
EEMSSTOP	R/W	0x0A0	EEPROM BIST stop address register	0x0	Table 369

Table 367. Register overview: FMC (base address 0x4003 C000) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
EEMSSIG	R	0x0A4	EEPROM 24-bit BIST signature register	0x0	Table 370
FMSTAT	R	0xFE0	Signature generation status register	0	Section 20.16.4.5
FMSTATCLR	W	0xFE8	Signature generation status clear register	-	Section 20.16.4.6

20.16.1 EEPROM BIST start address register

The EEPROM BIST start address register is used to program the start address for the BIST. During BIST the EEPROM devices are accessed with 16-bit read operations so the LSB of the address is fixed zero.

Table 368. EEPROM BIST start address register (EEMSSTART - address 0x4003 C09C) bit description

Bit	Symbol	Description	Reset value	Access
13:0	STARTA	BIST start address: Bit 0 is fixed zero since only even addresses are allowed.	0x0	R/W
31:14	-	Reserved	0x0	-

20.16.2 EEPROM BIST stop address register

The EEPROM BIST stop address register is used to program the stop address for the BIST and also to start the BIST. During BIST the EEPROM devices are accessed with 16-bit read operations so the LSB of the address is fixed zero.

Table 369. EEPROM BIST stop address register (EEMSSTOP - address 0x4003 C0A0) bit description

Bit	Symbol	Description	Reset value	Access
13:0	STOPA	BIST stop address: Bit 0 is fixed zero since only even addresses are allowed.	0x0	R/W
29:14	-	Reserved	0x0	-
30	DEVSEL	BIST device select bit 0: the BIST signature is generated over the total memory space. Singe pages are interleaved over the EEPROM devices when multiple devices are used, the signature is generated over memory of multiple devices. 1: the BIST signature is generated only over a memory range located on a single EEPROM device. Therefore the internal address generation is done such that the address' CS bits are kept stable to select only the same device. The address' MSB and LSB bits are used to step through the memory range specified by the start and stop address fields. Note: if this bit is set the start and stop address fields must be programmed such that they both address the same EEPROM device. Therefore the address' CS bits in both the start and stop address must be the same.	0x0	R/W
31	STRTBIST	BIST start bit Setting this bit will start the BIST. This bit is self-clearing.	0x0	R/W

20.16.3 EEPROM signature register

The EEPROM BIST signature register returns the signatures as produced by the embedded signature generators.

Table 370. EEPROM BIST signature register (EEMSSIG - address 0x4003 C0A4) bit description

Bits	Field name	Description	Reset value	Access
15:0	DATA_SIG	BIST 16-bit signature calculated from only the data bytes	0x0	R
31:16	PARITY_SIG	BIST 16-bit signature calculated from only the parity bits of the data bytes	0x0	R

20.16.4 Flash controller registers

20.16.4.1 Flash memory access register

Depending on the system clock frequency, access to the flash memory can be configured with various access times by writing to the FLASHCFG register.

Remark: Improper setting of this register may result in incorrect operation of the LPC11Uxx flash memory.

Table 371. Flash configuration register (FLASHCFG, address 0x4003 C010) bit description

Bit	Symbol	Value	Description	Reset value
1:0	FLASHTIM		Flash memory access time. FLASHTIM +1 is equal to the number of system clocks used for flash access.	0x2
		0x0	1 system clock flash access time (for system clock frequencies of up to 20 MHz).	
		0x1	2 system clocks flash access time (for system clock frequencies of up to 40 MHz).	
		0x2	3 system clocks flash access time (for system clock frequencies of up to 50 MHz).	
		0x3	Reserved	
31:2	-	-	Reserved. **User software must not change the value of these bits. Bits 31:2 must be written back exactly as read**.	-

20.16.4.2 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.

20.16.4.3 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 372 and Table 373 show the bit assignments in the FMSSTART and FMSSTOP registers respectively.

Table 372. Flash module signature start register (FMSSTART - 0x4003 C020) 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 373. Flash module signature stop register (FMSSTOP - 0x4003 C024) 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

20.16.4.4 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 20.16.4.7.

Table 377 show bit assignment of the FMSW0 and FMSW1, FMSW2, FMSW3 registers respectively.

Table 374. FMSW0 register (FMSW0, address: 0x4003 C02C) bit description

Bit	Symbol	Description	Reset value
31:0	SW0[31:0]	Word 0 of 128-bit signature (bits 31 to 0).	-

Table 375. FMSW1 register (FMSW1, address: 0x4003 C030) bit description

Bit	Symbol	Description	Reset value
31:0	SW1[63:32]	Word 1 of 128-bit signature (bits 63 to 32).	-

Table 376. FMSW2 register (FMSW2, address: 0x4003 C034) bit description

Bit	Symbol	Description	Reset value
31:0	SW2[95:64]	Word 2 of 128-bit signature (bits 95 to 64).	-

Table 377. FMSW3 register (FMSW3, address: 0x4003 40C8) bit description

Bit	Symbol	Description	Reset value
31:0	SW3[127:96]	Word 3 of 128-bit signature (bits 127 to 96).	-

20.16.4.5 Flash module status register

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 register. 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 378. Flash module status register (FMSTAT - 0x4003 CFE0) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved, user software should not write ones to reserved bits. 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
31:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

20.16.4.6 Flash module status clear register

The FMSTATCLR register is used to clear the signature generation completion flag.

Table 379. Flash module status clear register (FMSTATCLR - 0x0x4003 CFE8) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	SIG_DONE_CLR	Writing a 1 to this bits clears the signature generation completion flag (SIG_DONE) in the FMSTAT register.	0
31:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

20.16.4.7 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 the SIG_START bit in the FMSSTOP register. Starting the signature generation is typically combined with defining the stop address, which is done in the STOP bits 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:

Duration = int((60 / tcy) + 3) x (FMSSTOP - FMSSTART + 1)

When signature generation is triggered via software, the duration is in AHB clock cycles, and tcy 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.

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 69.

```
int128 signature = 0
int128 nextSignature
FOR address = flashpage 0 TO address = flashpage max
{
        FOR i = 0 TO 126 {
                nextSignature[i] = flashword[i] XOR signature[i+1] }
        nextSignature[127] = flashword[127] XOR signature[0] XOR signature[2]
                XOR signature[27] XOR signature[29]
        signature = nextSignature
}
return signature
```

Fig 69. Algorithm for generating a 128-bit signature

21.1 How to read this chapter

The debug functionality is identical for all LPC11Uxx parts.

21.2 Features

- Supports ARM Serial Wire Debug mode.
- Direct debug access to all memories, registers, and peripherals.
- No target resources are required for the debugging session.
- Four breakpoints. Four 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.
- Two data watchpoints that can also be used as triggers.
- Supports JTAG boundary scan.

21.3 Introduction

Debug functions are integrated into the ARM Cortex-M0. Serial wire debug functions are supported. The ARM Cortex-M0 is configured to support up to four breakpoints and two watchpoints.

21.4 Description

Debugging with the LPC11Uxx uses the Serial Wire Debug mode. Support for boundary scan is available.

21.5 Pin description

The tables below indicate the various pin functions related to debug. Some of these functions share pins with other functions which therefore may not be used at the same time.

Table 380. Serial Wire Debug pin description

Pin Name	Type	Description
SWCLK	Input	**Serial Wire Clock.** This pin is the clock for SWD debug logic when in the Serial Wire Debug mode (SWD). This pin is pulled up internally.
SWDIO	Input / Output	**Serial wire debug data input/output.** The SWDIO pin is used by an external debug tool to communicate with and control the LPC11Uxx. This pin is pulled up internally.

Table 381. JTAG boundary scan pin description

Pin Name	Type	Description
TCK	Input	**JTAG Test Clock.** This pin is the clock for JTAG boundary scan when the RESET pin is LOW.
TMS	Input	**JTAG Test Mode Select.** The TMS pin selects the next state in the TAP state machine. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.
TDI	Input	**JTAG Test Data In.** This is the serial data input for the shift register. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.
TDO	Output	**JTAG Test Data Output.** 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. This pin is used for JTAG boundary scan when the RESET pin is LOW.
TRST	Input	**JTAG Test Reset.** The TRST pin can be used to reset the test logic within the debug logic. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.

21.6 Functional description

21.6.1 Debug limitations

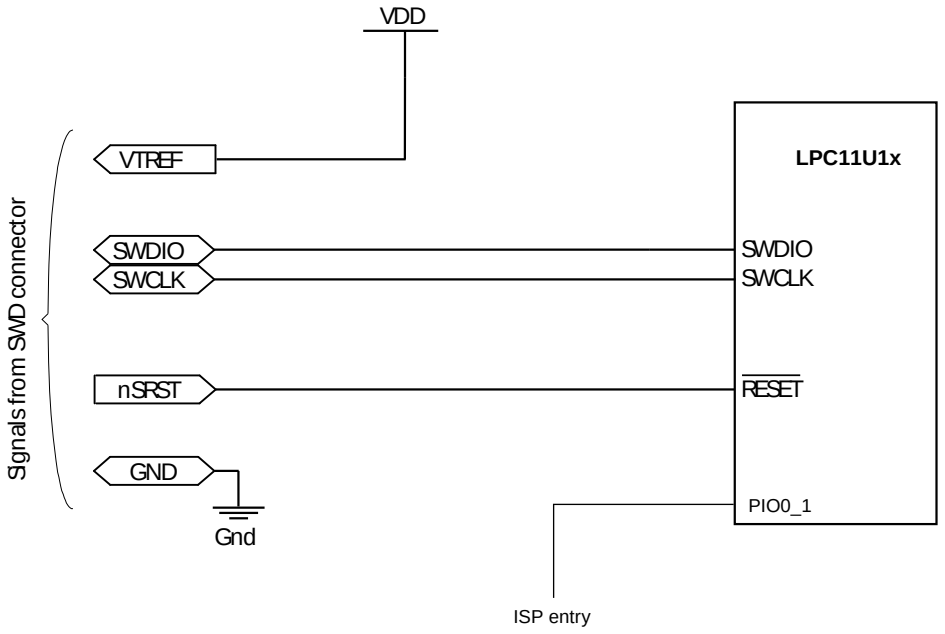
Important: Due to limitations of the ARM Cortex-M0 integration, the LPC11Uxx cannot wake up in the usual manner from Deep-sleep mode. It is recommended not to use this mode during debug.

Another issue is that debug mode changes the way in which reduced power modes work internal to the ARM Cortex-M0 CPU, and this ripples through the entire system. 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.

21.6.2 Debug connections for SWD

For debugging purposes, it is useful to provide access to the ISP entry pin PIO0_1. This pin can be used to recover the part from configurations which would disable the SWD port such as improper PLL configuration, reconfiguration of SWD pins as ADC inputs, entry into Deep power-down mode out of reset, etc. This pin can be used for other functions such as GPIO, but it should not be held LOW on power-up or reset.



The VTREF pin on the SWD connector enables the debug connector to match the target voltage.

Fig 70. Connecting the SWD pins to a standard SWD connector

21.6.3 Boundary scan

Debug functions are integrated into the ARM Cortex-M0. Serial wire debug functions are supported in addition to a standard JTAG boundary scan. The ARM Cortex-M0 is configured to support up to four breakpoints and two watch points.

The $\overline{\text{RESET}}$ pin selects between the JTAG boundary scan ($\overline{\text{RESET}}$ = LOW) and the ARM SWD debug ($\overline{\text{RESET}}$ = HIGH). The ARM SWD debug port is disabled while the UM10462 is in reset.

To perform boundary scan testing, follow these steps:

1. Erase any user code residing in flash.
2. Power up the part with the $\overline{\text{RESET}}$ pin pulled HIGH externally.
3. Wait for at least 250 μs.
4. Pull the $\overline{\text{RESET}}$ pin LOW externally.
5. Perform boundary scan operations.
6. Once the boundary scan operations are completed, assert the TRST pin to enable the SWD debug mode and release the $\overline{\text{RESET}}$ pin (pull HIGH).

Remark: The JTAG interface cannot be used for debug purposes.

Remark: POR, BOD reset, or a LOW on the TRST pin puts the test TAP controller in the Test-Logic Reset state. The first TCK clock while $\overline{\text{RESET}}$ = HIGH places the test TAP in Run-Test Idle mode.

22.1 How to read this chapter

The ROM-based 32-bit integer division routines are available for all LPC11Uxx.

22.2 Features

- Performance-optimized signed/unsigned integer division.
- Performance-optimized signed/unsigned integer division with remainder.
- ROM-based routines to reduce code size.
- Support for integers up to 32 bit.
- ROM calls can easily be added to EABI-compliant functions to overload “/” and “%” operators in C.

22.3 Description

The API calls to the ROM are performed by executing functions which are pointed by a pointer within the ROM Driver Table. Figure 71 shows the pointer structure used to call the Integer divider API.

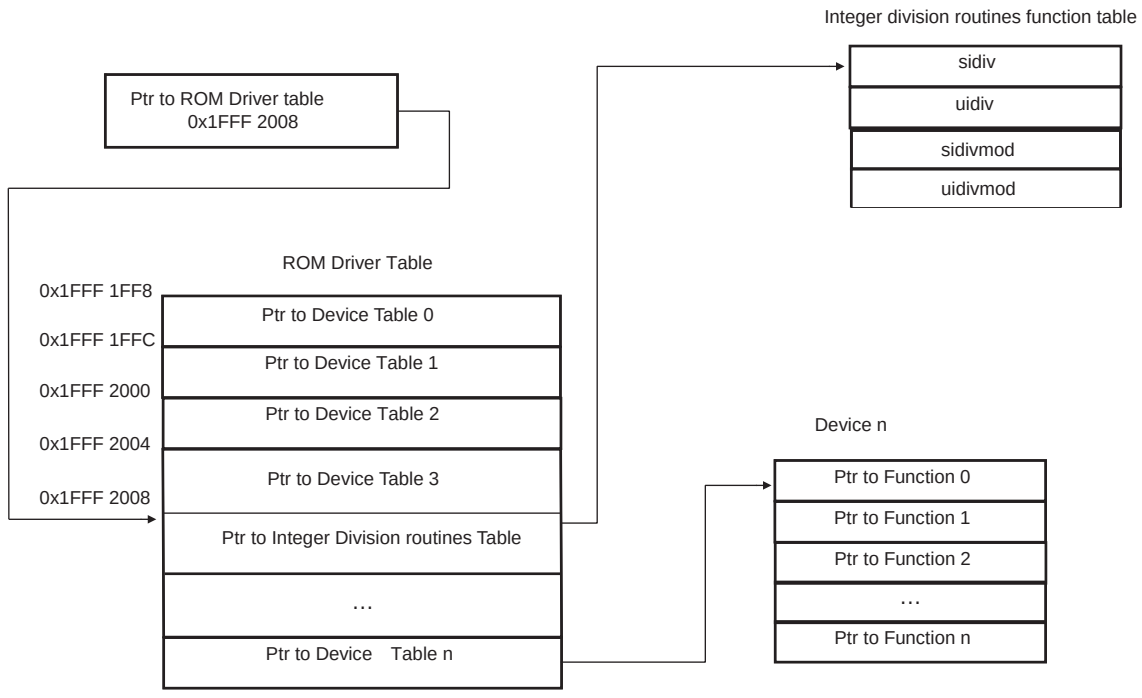


Fig 71. Power profiles pointer structure

The integer division routines perform arithmetic integer division operations and can be called in the application code through simple API calls.

The following function prototypes are used:

```
typedef struct { int quot; int rem; } idiv_return;

typedef struct { unsigned quot; unsigned rem; } uidiv_return;

typedef struct {
     /* Signed integer division */
     int(*sidiv)(int numerator, int denominator);
     /* Unsigned integer division */
     unsigned(*uidiv) (unsigned numerator, unsigned denominator);
     /* Signed integer division with remainder */
     idiv_return(*sidivmod)(int numerator, int denominator);
     /* Unsigned integer division with remainder */
     uidiv_return (*uidivmod)(unsigned numerator, unsigned denominator);
} LPC_ROM_DIV_STRUCT;
```

22.4 Examples

22.4.1 Initialization

The example C-code listing below shows how to initialize the API's ROM table pointer.

```
typedef struct _ROM {
     const unsigned p_dev1;
     const unsigned p_dev2;
     const unsigned p_dev3;
     const PWRD *pPWRD;
     const LPC_ROM_DIV_STUCT * pROMDiv;
     const unsigned p_dev4;
     const unsigned p_dev5;
     const unsigned p_dev6;

} ROM;

ROM ** rom = (ROM **) 0x1FFF1FF8;
     pDivROM = (*rom)->pROMDiv;
```

22.4.2 Signed division

The example C-code listing below shows how to perform a signed integer division via the ROM API.

```
/* Divide (-99) by (+6) */
int32_t result;
```

```
result = pDivROM->sidiv(-99, 6);
/* result now contains (-16) */
```

22.4.3 Unsigned division with remainder

The example C-code listing below shows how to perform an unsigned integer division with remainder via the ROM API.

```
/* Modulus Divide (+99) by (+4) */
uidiv_return result;
result = pDivROM-> uidivmod (+99, 4);
/* result.quot contains (+24) */
/* result.rem contains (+3) */
```

23.1 Introduction

The following material is using the ARM *Cortex-M0 User Guide*. Minor changes have been made regarding the specific implementation of the Cortex-M0 for the LPC11Uxx.

23.2 About the Cortex-M0 processor and core peripherals

The Cortex-M0 processor is an entry-level 32-bit ARM Cortex processor designed for a broad range of embedded applications. It offers significant benefits to developers, including:

- a simple architecture that is easy to learn and program
- ultra-low power, energy efficient operation
- excellent code density
- deterministic, high-performance interrupt handling
- upward compatibility with Cortex-M processor family.

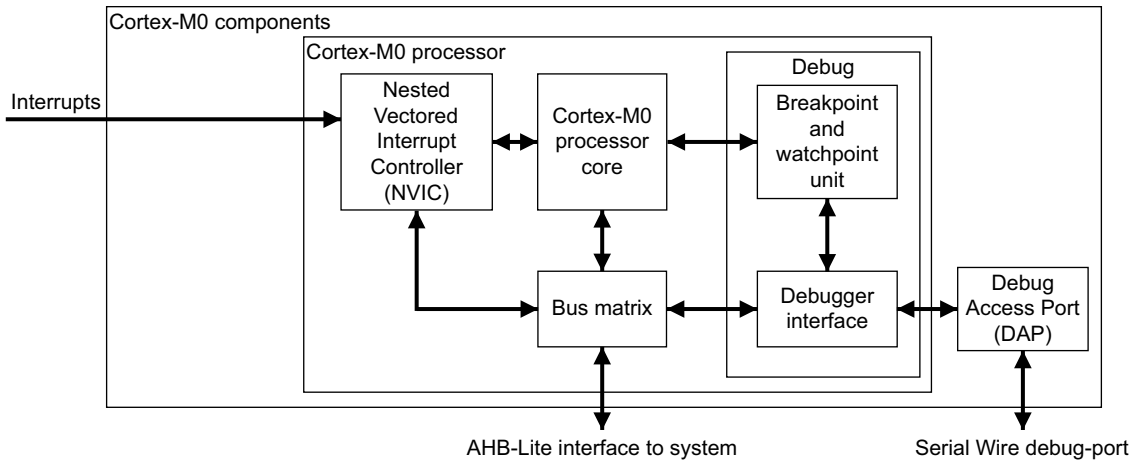


Fig 72. Cortex-M0 implementation

The Cortex-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single-cycle multiplier.

The Cortex-M0 processor implements the ARMv6-M architecture, which is based on the 16-bit Thumb instruction set and includes Thumb-2 technology. This provides the exceptional performance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

The Cortex-M0 processor closely integrates a configurable **Nested Vectored Interrupt Controlle**r (NVIC), to deliver industry-leading interrupt performance. The NVIC:

- includes a **non-maskable interrupt** (NMI).
- provides zero jitter interrupt option.
- provides four 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 abandon and restart load-multiple and store-multiple operations. Interrupt handlers do not require any assembler wrapper code, 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.

23.2.1 System-level interface

The Cortex-M0 processor provides a single system-level interface using AMBA technology to provide high speed, low latency memory accesses.

23.2.2 Integrated configurable debug

The Cortex-M0 processor implements a complete hardware debug solution, with extensive hardware breakpoint and watchpoint options. This provides high system visibility of the processor, memory and peripherals through a 2-pin **Serial Wire Debug** (SWD) port that is ideal for microcontrollers and other small package devices.

23.2.3 Cortex-M0 processor features summary

- high code density with 32-bit performance
- tools and binary upwards compatible with Cortex-M processor family
- integrated ultra low-power sleep modes
- efficient code execution permits slower processor clock or increases sleep mode time
- single-cycle 32-bit hardware multiplier
- zero jitter interrupt handling
- extensive debug capabilities.

23.2.4 Cortex-M0 core peripherals

These are:

NVIC — The 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.

23.3 Processor

23.3.1 Programmers model

This section describes the Cortex-M0 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and stacks.

23.3.1.1 Processor modes

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 all exception processing.

23.3.1.2 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 23.3.1.3.2.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see Section 23–23.3.1.3.7. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 382. Summary of processor mode and stack use options

Processor mode	Used to execute	Stack used
Thread	Applications	Main stack or process stack See Section 23–23.3.1.3.7
Handler	Exception handlers	Main stack

23.3.1.3 Core registers

The processor core registers are:

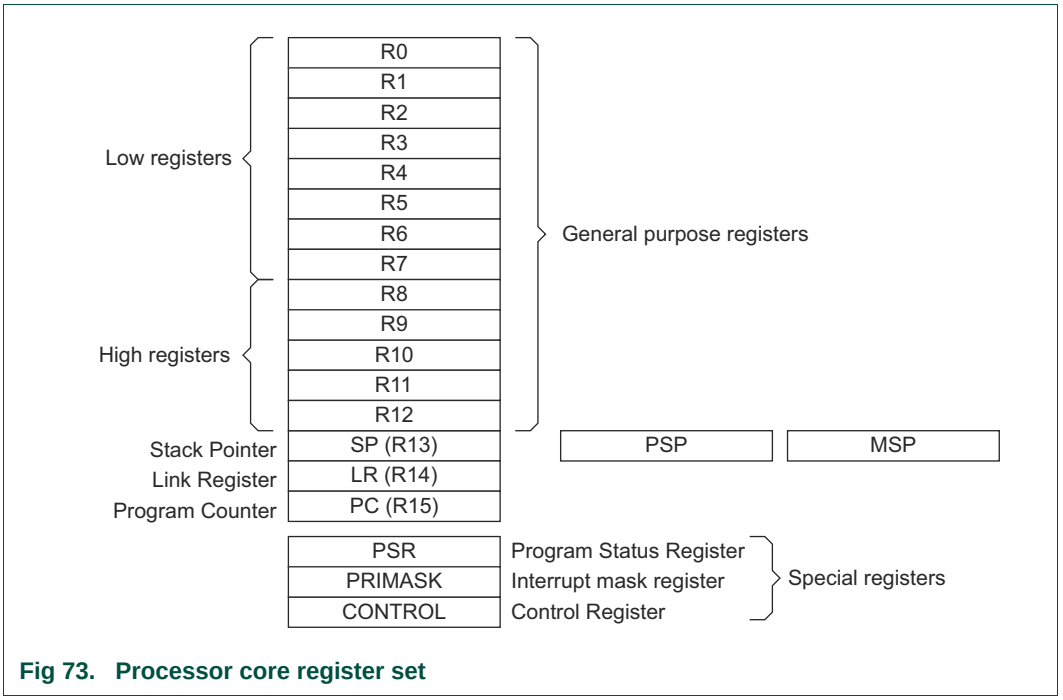


Fig 73. Processor core register set

Table 383. Core register set summary

Name	Type[1]	Reset value	Description
R0-R12	RW	Unknown	Section 23–23.3.1.3.1
MSP	RW	See description	Section 23–23.3.1.3.2
PSP	RW	Unknown	Section 23–23.3.1.3.2
LR	RW	Unknown	Section 23–23.3.1.3.3
PC	RW	See description	Section 23–23.3.1.3.4
PSR	RW	Unknown[2]	Table 23–384
APSR	RW	Unknown	Table 23–385
IPSR	RO	0x00000000	Table 386
EPSR	RO	Unknown[2]	Table 23–387
PRIMASK	RW	0x00000000	Table 23–388
CONTROL	RW	0x00000000	Table 23–389

[1] Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

[2] Bit[24] is the T-bit and is loaded from bit[0] of the reset vector.

23.3.1.3.1 General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

23.3.1.3.2 Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = **Main Stack Pointer** (MSP). This is the reset value.
- 1 = **Process Stack Pointer** (PSP).

On reset, the processor loads the MSP with the value from address 0x00000000.

23.3.1.3.3 Link Register

The **Link Register** (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the LR value is Unknown.

23.3.1.3.4 Program Counter

The **Program Counter** (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

23.3.1.3.5 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 PSR bit assignments are:

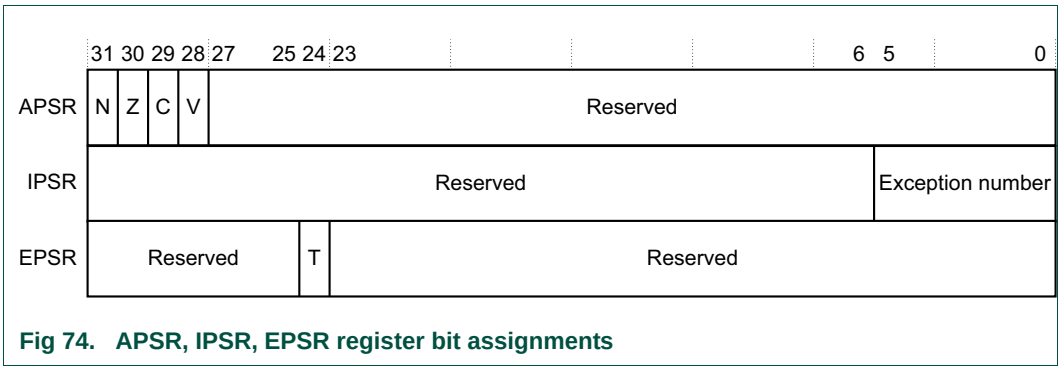


Fig 74. APSR, IPSR, EPSR register bit assignments

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 384. PSR register combinations

Register	Type	Combination
PSR	RW[1][2]	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	RW[1]	APSR and IPSR
EAPSR	RW[2]	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 23–23.4.7.6 and Section 23–23.4.7.7 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 23–383 for its attributes. The bit assignments are:

Table 385. APSR bit assignments

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27:0]	-	Reserved

See Section 23.4.4.1.4 for more information about the APSR negative, zero, carry or borrow, and overflow flags.

Interrupt Program Status Register: The IPSR contains the exception number of the current **Interrupt Service Routine** (ISR). See the register summary in Table 23–383 for its attributes. The bit assignments are:

Table 386. IPSR bit assignments

Bits	Name	Function
[31:6]	-	Reserved
[5:0]	Exception number	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4-10 = Reserved 11 = SVCall 12, 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 . . . 47 = IRQ31 48-63 = Reserved. see Section 23–23.3.3.2 for more information.

Execution Program Status Register: The EPSR contains the Thumb state bit.

See the register summary in Table 23–383 for the EPSR attributes. The bit assignments are:

Table 387. EPSR bit assignments

Bits	Name	Function
[31:25]	-	Reserved
[24]	T	Thumb state bit
[23:0]	-	Reserved

Attempts by application software to read the EPSR directly using the `MRS` instruction always return zero. Attempts to write the EPSR using the `MSR` instruction are ignored. Fault handlers can examine the EPSR value in the stacked PSR to determine the cause of the fault. See Section 23–23.3.3.6. The following can clear the T bit to 0:

- instructions `BLX`, `BX` and `POP{PC}`
- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry.

Attempting to execute instructions when the T bit is 0 results in a HardFault or lockup. See Section 23–23.3.4.1 for more information.

Interruptible-restartable instructions: The interruptible-restartable instructions are `LDM` and `STM`. When an interrupt occurs during the execution of one of these instructions, the processor abandons execution of the instruction.

After servicing the interrupt, the processor restarts execution of the instruction from the beginning.

23.3.1.3.6 Exception mask register

The exception mask register disables the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks or code sequences requiring atomicity.

To disable or re-enable exceptions, use the `MSR` and `MRS` instructions, or the `CPS` instruction, to change the value of PRIMASK. See Section 23–23.4.7.6, Section 23–23.4.7.7, and Section 23–23.4.7.2 for more information.

Priority Mask Register: The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in Table 23–383 for its attributes. The bit assignments are:

Table 388. 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.

23.3.1.3.7 CONTROL register

The CONTROL register controls the stack used when the processor is in Thread mode. See the register summary in Table 23–383 for its attributes. The bit assignments are:

Table 389. 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]	-	Reserved.

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, it is recommended 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 23–23.4.7.6.

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 23–23.4.7.5.

23.3.1.4 Exceptions and interrupts

The Cortex-M0 processor supports interrupts and system exceptions. The processor and the **Nested Vectored Interrupt Controller** (NVIC) prioritize and handle all exceptions. An interrupt or exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See Section 23–23.3.3.6.1 and Section 23–23.3.3.6.2 for more information.

The NVIC registers control interrupt handling. See Section 23–23.5.2 for more information.

23.3.1.5 Data types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- manages all data memory accesses as little-endian. Instruction memory and **Private Peripheral Bus** (PPB) accesses are always little-endian. See Section 23–23.3.2.1 for more information.

23.3.1.6 The Cortex Microcontroller Software Interface Standard

ARM provides the **Cortex Microcontroller Software Interface Standard** (CMSIS) for programming Cortex-M0 microcontrollers. The CMSIS is an integrated part of the device driver library.

For a Cortex-M0 microcontroller system, 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.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M0 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

The 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.

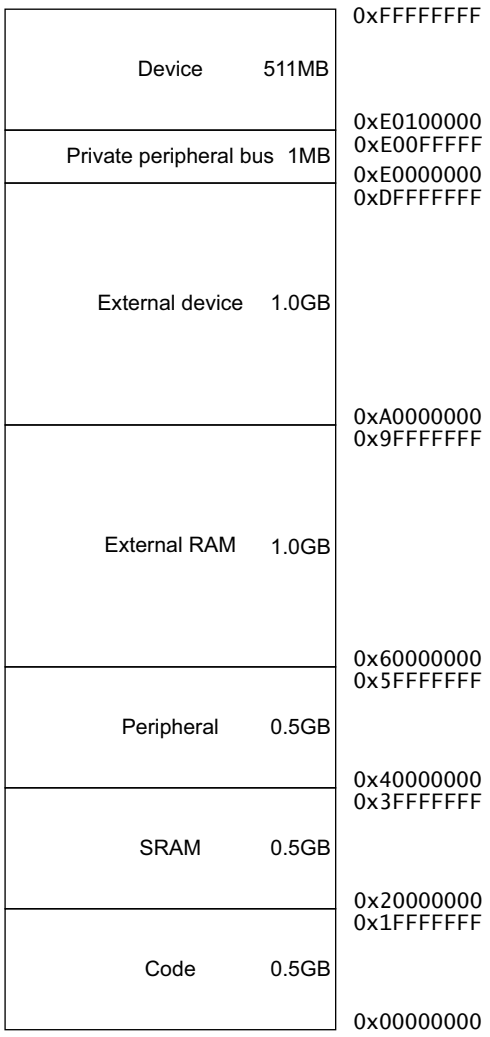
Remark: This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- Section 23.3.5.3 “Power management programming hints”
- Section 23.4.2 “Intrinsic functions”
- Section 23.5.2.1 “Accessing the Cortex-M0 NVIC registers using CMSIS”
- Section 23.5.2.8.1 “NVIC programming hints”.

23.3.2 Memory model

This section describes the processor memory map and the behavior of memory accesses. The processor has a fixed memory map that provides up to 4GB of addressable memory. The memory map is:



For the LPC11Uxx specific implementation of the memory map, see Figure 3.

Fig 75. Cortex-M0 memory map

The processor reserves regions of the **Private peripheral bus** (PPB) address range for core peripheral registers, see Section 23–23.2.

23.3.2.1 Memory regions, types and attributes

The memory map is split 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.

Execute Never (XN) — Means the processor prevents instruction accesses. A HardFault exception is generated on executing an instruction fetched from an XN region of memory.

23.3.2.2 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 any re-ordering 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 23–23.3.2.4.

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	-	<	<	<

Fig 76. Memory ordering restrictions

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.

23.3.2.3 Behavior of memory accesses

The behavior of accesses to each region in the memory map is:

Table 390. Memory access behavior

Address range	Memory region	Memory type[1]	XN[1]	Description
0x00000000-0x1FFFFFFF	Code	Normal	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal	-	Executable region for data. You can also put code here.
0x40000000-0x5FFFFFFF	Peripheral	Device	XN	External device memory.
0x60000000-0x9FFFFFFF	External RAM	Normal	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device	XN	External device memory.
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	XN	This region includes the NVIC, System timer, and System Control Block. Only word accesses can be used in this region.
0xE0100000-0xFFFFFFFF	Device	Device	XN	Vendor specific.

[1] See Section 23–23.3.2.1 for more information.

The Code, SRAM, and external RAM regions can hold programs.

23.3.2.4 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
- memory or devices in the memory map might have different wait states
- some memory accesses are buffered or speculative.

Section 23–23.3.2.2 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 23–23.4.7.3.

DSB — The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See Section 23–23.4.7.4.

ISB — The Instruction Synchronization Barrier (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See Section 23–23.4.7.5.

The following are examples of using memory barrier instructions:

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. This ensures subsequent instruction execution uses the updated memory map.

Memory accesses to Strongly-ordered memory, such as the System Control Block, do not require the use of `DMB` instructions.

The processor preserves transaction order relative to all other transactions.

23.3.2.5 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 23–23.3.2.5.1 describes how words of data are stored in memory.

23.3.2.5.1 Little-endian format

In little-endian format, the processor stores the **least significant byte** (lsbyte) of a word at the lowest-numbered byte, and the **most significant byte** (msbyte) at the highest-numbered byte. For example:

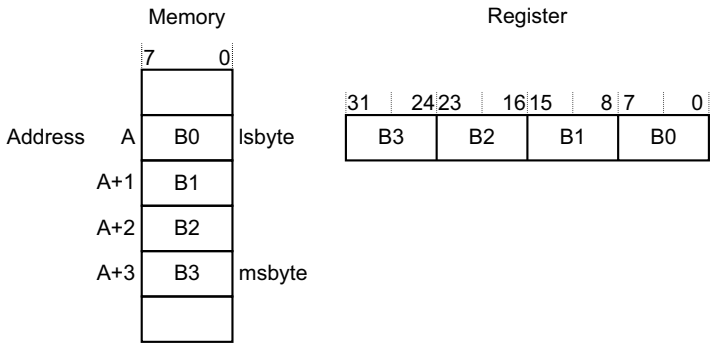


Fig 77. Little-endian format

23.3.3 Exception model

This section describes the exception model.

23.3.3.1 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.

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.

23.3.3.2 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 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.

HardFault — A HardFault is an exception that occurs because of an error during normal or exception processing. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

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 391. Properties of different exception types

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address[2]
1	-	Reset	-3, the highest	`0x00000004`
2	-14	NMI	-2	`0x00000008`
3	-13	HardFault	-1	`0x0000000C`
4-10	-	Reserved	-	-
11	-5	SVCall	Configurable[3]	`0x0000002C`
12-13	-	Reserved	-	-

Table 391. Properties of different exception types

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address[2]
14	-2	PendSV	Configurable[3]	0x00000038
15	-1	SysTick	Configurable[3]	0x0000003C
16 and above	0 and above	Interrupt (IRQ)	Configurable[3]	0x00000040 and above[4]

[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 23–386.

[2] See Section 23.3.3.4 for more information.

[3] See Section 23–23.5.2.6.

[4] Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute additional instructions between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 23–391 shows as having configurable priority, see Section 23–23.5.2.3.

For more information about HardFaults, see Section 23–23.3.4.

23.3.3.3 Exception handlers

The processor handles exceptions using:

Interrupt Service Routines (ISRs) — Interrupts IRQ0 to IRQ31 are the exceptions handled by ISRs.

Fault handler — HardFault is the only exception handled by the fault handler.

System handlers — NMI, PendSV, SVCall SysTick, and HardFault are all system exceptions handled by system handlers.

23.3.3.4 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 23–78 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 written in Thumb code.

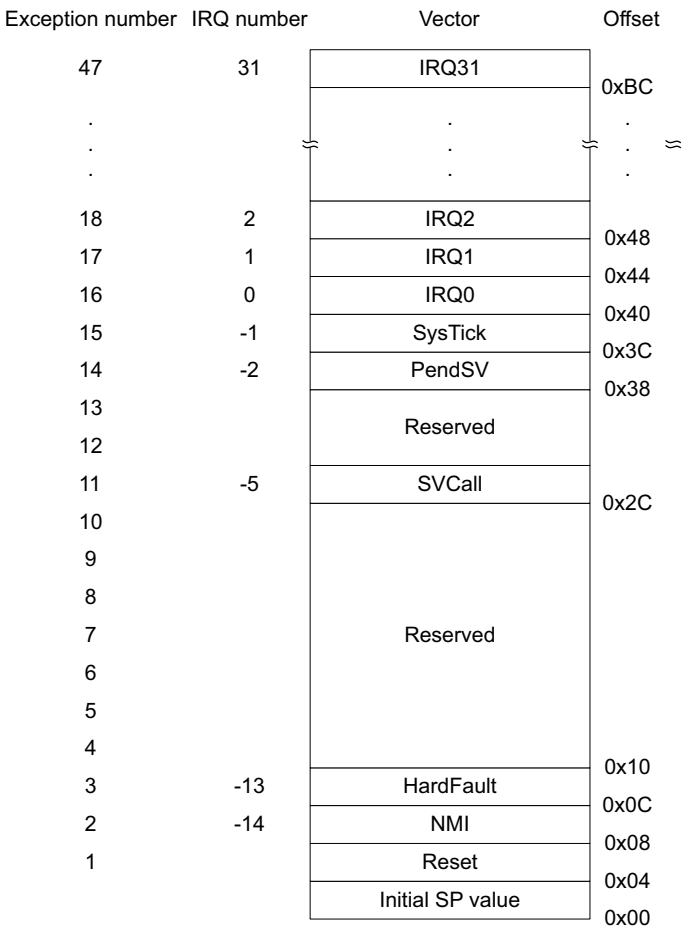


Fig 78. Vector table

The vector table is fixed at address `0x00000000`.

23.3.3.5 Exception priorities

As Table 23–391 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, 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 23–23.5.3.7
- Section 23–23.5.2.6.

Configurable priority values are in the range 0-192, in steps of 64. The Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

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.

23.3.3.6 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.

When one exception preempts another, the exceptions are called nested exceptions. See Section 23–23.3.3.6.1 for 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 23–23.3.3.6.2 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 would be the same for both exceptions. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

23.3.3.6.1 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 exception being handled.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has greater priority than any limit set by the mask register, see Section 23–23.3.1.3.6. 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 to as **stacking** and the structure of eight data words is referred to as a **stack frame**. The stack frame contains the following information:

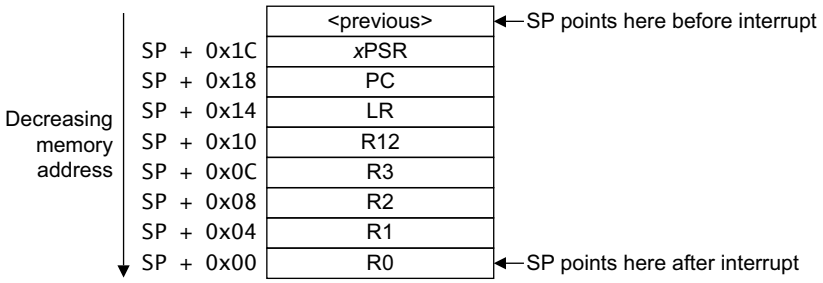


Fig 79. Exception entry stack contents

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The stack frame is aligned to a double-word address.

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.

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 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.

23.3.3.6.2 Exception return

Exception return occurs when the processor is in Handler mode and execution of one of the following instructions attempts to set the PC to an EXC_RETURN value:

- a `POP` instruction that loads the PC
- a `BX` instruction using any register.

The processor saves an EXC_RETURN value to the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. Bits[31:4] of an EXC_RETURN value are `0xFFFFFFF`. When the processor loads a value matching this pattern to the PC it detects that the operation is a

not a normal branch operation and, instead, that the exception is complete. Therefore, it starts the exception return sequence. Bits[3:0] of the EXC_RETURN value indicate the required return stack and processor mode, as Table 23–392 shows.

Table 392. Exception return behavior

EXC_RETURN	Description
0xFFFFFFF1	Return to Handler mode. Exception return gets state from the main stack. Execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode. Exception return gets state from MSP. Execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode. Exception return gets state from PSP. Execution uses PSP after return.
All other values	Reserved.

23.3.4 Fault handling

Faults are a subset of exceptions, see Section 23–23.3.3. All faults result in the HardFault exception being taken or cause lockup if they occur in the NMI or HardFault handler. The faults are:

- execution of an SVC instruction at a priority equal or higher than SVCall
- execution of a BKPT instruction without a debugger attached
- a system-generated bus error on a load or store
- execution of an instruction from an XN memory address
- execution of an instruction from a location for which the system generates a bus fault
- a system-generated bus error on a vector fetch
- execution of an Undefined instruction
- execution of an instruction when not in Thumb-State as a result of the T-bit being previously cleared to 0
- an attempted load or store to an unaligned address.

Only Reset and NMI can preempt the fixed priority HardFault handler. A HardFault can preempt any exception other than Reset, NMI, or another hard fault.

23.3.4.1 Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers, or if the system generates a bus error when unstacking the PSR on an exception return using the MSP. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until one of the following occurs:

- it is reset
- a debugger halts it
- an NMI occurs and the current lockup is in the HardFault handler.

If lockup state occurs in the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

23.3.5 Power management

The Cortex-M0 processor sleep modes reduce power consumption:

- a sleep mode, that stops the processor clock
- a Deep-sleep and Power-down modes, for details see Section 3.9.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see Section 23–23.5.3.5.

This section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

23.3.5.1 Entering sleep mode

This section describes the mechanisms software can use to put the processor into sleep mode.

The system can generate spurious wake-up 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 in to sleep mode.

23.3.5.1.1 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 23–23.4.7.12 for more information.

23.3.5.1.2 Wait for event

Remark: The WFE instruction is not implemented on the LPC11Uxx.

The Wait For Event instruction, `WFE`, causes entry to sleep mode conditional on the value of a one-bit event register. When the processor executes a `WFE` instruction, it checks the value of the event register:

0 — The processor stops executing instructions and enters sleep mode

1 — The processor sets the register to zero and continues executing instructions without entering sleep mode.

See Section 23–23.4.7.11 for more information.

If the event register is 1, this indicates that the processor must not enter sleep mode on execution of a `WFE` instruction. Typically, this is because of the assertion of an external event, or because another processor in the system has executed a `SEV` instruction, see Section 23–23.4.7.9. Software cannot access this register directly.

23.3.5.1.3 Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of an exception handler and returns to Thread mode it immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an interrupt occurs.

23.3.5.2 Wake-up from sleep mode

The conditions for the processor to wake-up depend on the mechanism that caused it to enter sleep mode.

23.3.5.2.1 Wake-up 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. 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, see Section 23–23.3.1.3.6.

23.3.5.2.2 Wake-up from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry
- in a multiprocessor system, another processor in the system executes a `SEV` instruction.

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 23–23.5.3.5.

23.3.5.3 Power management programming hints

ISO/IEC C cannot directly generate the `WFI`, `WFE`, and `SEV` instructions. The CMSIS provides the following intrinsic functions for these instructions:

```
void __WFE(void) // Wait for Event

void __WFI(void) // Wait for Interrupt

void __SEV(void) // Send Event
```

23.4 Instruction set

23.4.1 Instruction set summary

The processor implements a version of the Thumb instruction set. Table 393 lists the supported instructions.

Remark: In Table 393

- angle brackets, <>, enclose alternative forms of the operand
- braces, {}, enclose optional operands and mnemonic parts
- the Operands column is not exhaustive.

For more information on the instructions and operands, see the instruction descriptions.

Table 393. Cortex-M0 instructions

Mnemonic	Operands	Brief description	Flags	Reference
ADCS	*{Rd,} Rn, Rm*	Add with Carry	N,Z,C,V	Section 23–23.4.5.1
ADD{S}	*{Rd,} Rn, <Rm|#imm>*	Add	N,Z,C,V	Section 23–23.4.5.1
ADR	*Rd, label*	PC-relative Address to Register	-	Section 23–23.4.4.1
ANDS	*{Rd,} Rn, Rm*	Bitwise AND	N,Z	Section 23–23.4.5.1
ASRS	*{Rd,} Rm, <Rs|#imm>*	Arithmetic Shift Right	N,Z,C	Section 23–23.4.5.3
B{cc}	*label*	Branch {conditionally}	-	Section 23–23.4.6.1
BICS	*{Rd,} Rn, Rm*	Bit Clear	N,Z	Section 23–23.4.5.2
BKPT	*#imm*	Breakpoint	-	Section 23–23.4.7.1
BL	*label*	Branch with Link	-	Section 23–23.4.6.1
BLX	*Rm*	Branch indirect with Link	-	Section 23–23.4.6.1
BX	*Rm*	Branch indirect	-	Section 23–23.4.6.1
CMN	*Rn, Rm*	Compare Negative	N,Z,C,V	Section 23–23.4.5.4
CMP	*Rn, <Rm|#imm>*	Compare	N,Z,C,V	Section 23–23.4.5.4
CPSID	*i*	Change Processor State, Disable Interrupts	-	Section 23–23.4.7.2
CPSIE	*i*	Change Processor State, Enable Interrupts	-	Section 23–23.4.7.2
DMB	-	Data Memory Barrier	-	Section 23–23.4.7.3
DSB	-	Data Synchronization Barrier	-	Section 23–23.4.7.4
EORS	*{Rd,} Rn, Rm*	Exclusive OR	N,Z	Section 23–23.4.5.2
ISB	-	Instruction Synchronization Barrier	-	Section 23–23.4.7.5
LDM	*Rn{!}, reglist*	Load Multiple registers, increment after	-	Section 23–23.4.4.5

Table 393. Cortex-M0 instructions

Mnemonic	Operands	Brief description	Flags	Reference
LDR	*Rt, label*	Load Register from PC-relative address	-	Section 23–23.4.4
LDR	*Rt, [Rn, <Rm|#imm>]*	Load Register with word	-	Section 23–23.4.4
LDRB	*Rt, [Rn, <Rm|#imm>]*	Load Register with byte	-	Section 23–23.4.4
LDRH	*Rt, [Rn, <Rm|#imm>]*	Load Register with halfword	-	Section 23–23.4.4
LDRSB	*Rt, [Rn, <Rm|#imm>]*	Load Register with signed byte	-	Section 23–23.4.4
LDRSH	*Rt, [Rn, <Rm|#imm>]*	Load Register with signed halfword	-	Section 23–23.4.4
LSLS	*{Rd,} Rn, <Rs|#imm>*	Logical Shift Left	N,Z,C	Section 23–23.4.5.3
U	*{Rd,} Rn, <Rs|#imm>*	Logical Shift Right	N,Z,C	Section 23–23.4.5.3
MOV{S}	*Rd, Rm*	Move	N,Z	Section 23–23.4.5.5
MRS	*Rd, spec_reg*	Move to general register from special register	-	Section 23–23.4.7.6
MSR	*spec_reg, Rm*	Move to special register from general register	N,Z,C,V	Section 23–23.4.7.7
MULS	*Rd, Rn, Rm*	Multiply, 32-bit result	N,Z	Section 23–23.4.5.6
MVNS	*Rd, Rm*	Bitwise NOT	N,Z	Section 23–23.4.5.5
NOP	-	No Operation	-	Section 23–23.4.7.8
ORRS	*{Rd,} Rn, Rm*	Logical OR	N,Z	Section 23–23.4.5.2
POP	*reglist*	Pop registers from stack	-	Section 23–23.4.4.6
PUSH	*reglist*	Push registers onto stack	-	Section 23–23.4.4.6
REV	*Rd, Rm*	Byte-Reverse word	-	Section 23–23.4.5.7
REV16	*Rd, Rm*	Byte-Reverse packed halfwords	-	Section 23–23.4.5.7
REVSH	*Rd, Rm*	Byte-Reverse signed halfword	-	Section 23–23.4.5.7
RORS	*{Rd,} Rn, Rs*	Rotate Right	N,Z,C	Section 23–23.4.5.3
RSBS	*{Rd,} Rn, #0*	Reverse Subtract	N,Z,C,V	Section 23–23.4.5.1
SBCS	*{Rd,} Rn, Rm*	Subtract with Carry	N,Z,C,V	Section 23–23.4.5.1
SEV	-	Send Event	-	Section 23–23.4.7.9
STM	*Rn!, reglist*	Store Multiple registers, increment after	-	Section 23–23.4.4.5

Table 393. Cortex-M0 instructions

Mnemonic	Operands	Brief description	Flags	Reference
STR	*Rt, [Rn, <Rm|#imm>]*	Store Register as word	-	Section 23–23.4.4
STRB	*Rt, [Rn, <Rm|#imm>]*	Store Register as byte	-	Section 23–23.4.4
STRH	*Rt, [Rn, <Rm|#imm>]*	Store Register as halfword	-	Section 23–23.4.4
SUB{S}	*{Rd,} Rn, <Rm|#imm>*	Subtract	N,Z,C,V	Section 23–23.4.5.1
SVC	*#imm*	Supervisor Call	-	Section 23–23.4.7.10
SXTB	*Rd, Rm*	Sign extend byte	-	Section 23–23.4.5.8
SXTH	*Rd, Rm*	Sign extend halfword	-	Section 23–23.4.5.8
TST	*Rn, Rm*	Logical AND based test	N,Z	Section 23–23.4.5.9
UXTB	*Rd, Rm*	Zero extend a byte	-	Section 23–23.4.5.8
UXTH	*Rd, Rm*	Zero extend a halfword	-	Section 23–23.4.5.8
WFE	-	Wait For Event	-	Section 23–23.4.7.11
WFI	-	Wait For Interrupt	-	Section 23–23.4.7.12

23.4.2 Intrinsic functions

ISO/IEC C code cannot directly access some Cortex-M0 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 the relevant instruction.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

Table 394. CMSIS intrinsic functions to generate some Cortex-M0 instructions

Instruction	CMSIS intrinsic function
CPSIE i	void __enable_irq(void)
CPSID i	void __disable_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
NOP	void __NOP(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)

Table 394. CMSIS intrinsic functions to generate some Cortex-M0 instructions

Instruction	CMSIS intrinsic function
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 395. insic 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)
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)

23.4.3 About the instruction descriptions

The following sections give more information about using the instructions:

- Section 23.4.3.1 “Operands”
- Section 23.4.3.2 “Restrictions when using PC or SP”
- Section 23.4.3.3 “Shift Operations”
- Section 23.4.3.4 “Address alignment”
- Section 23.4.3.5 “PC-relative expressions”
- Section 23.4.3.6 “Conditional execution”.

23.4.3.1 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 other operands.

23.4.3.2 Restrictions when using PC or SP

Many instructions are unable to use, or 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: When you update the PC with a BX, BLX, or POP instruction, bit[0] of any address must be 1 for correct execution. This is because this bit indicates the destination instruction set, and the Cortex-M0 processor only supports Thumb instructions. When a BL or BLX instruction writes the value of bit[0] into the LR it is automatically assigned the value 1.

23.4.3.3 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, and ROR and the result is written to a destination register.The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description. 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.

23.4.3.3.1 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 23–80.

You can use the ASR operation to divide the signed value in the register *Rm* by 2^n, with the result being rounded towards negative-infinity.

When the instruction is ASRS the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

Remark:

- 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 80. ASR #3

23.4.3.3.2 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 81.

You can use the LSR 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, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

Remark:

- 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 81. LSR #3

23.4.3.3.3 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 82.

You can use the LSL 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 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.

Remark:

- 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 82. LSL #3

23.4.3.3.4 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 23–83.

When the instruction is RORS the carry flag is updated to the last bit rotation, bit[*n*-1], of the register *Rm*.

Remark:

- 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*, greater than 32 is the same as

  ROR

  with shift length *n*-32.

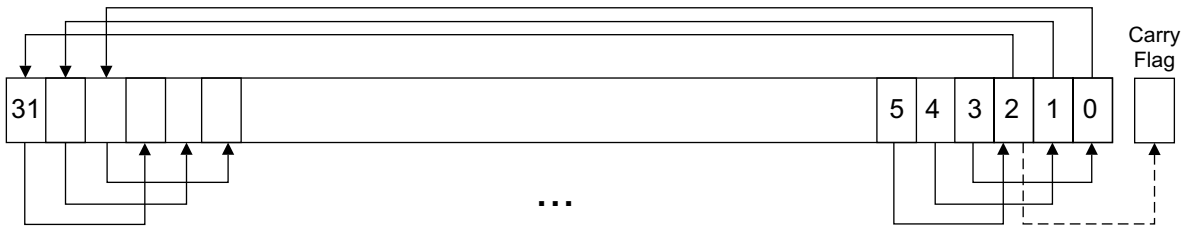


Fig 83. ROR #3

23.4.3.4 Address alignment

An aligned access is an operation where a word-aligned address is used for a word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

There is no support for unaligned accesses on the Cortex-M0 processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

23.4.3.5 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.

Remark:

- For most instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- 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, #*imm*].

23.4.3.6 Conditional execution

Most data processing instructions update the condition flags in the **Application Program Status Register** (APSR) according to the result of the operation, see Section . 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 a conditional branch instruction, 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.

On the Cortex-M0 processor, conditional execution is available by using conditional branches.

This section describes:

- Section 23.4.3.6.1 “The condition flags”
- Section 23.4.3.6.2 “Condition code suffixes”.

23.4.3.6.1 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 23–23.3.1.3.5.

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 a shift or rotate instruction.

Overflow occurs when the sign of the result, in bit[31], does not match the sign of the result had the operation been performed at infinite precision, for example:

- if adding two negative values results in a positive value
- if adding two positive values results in a negative value
- if subtracting a positive value from a negative value generates a positive value
- if subtracting a negative value from a positive value generates a negative value.

The Compare operations are identical to subtracting, for CMP, or adding, for CMN, except that the result is discarded. See the instruction descriptions for more information.

23.4.3.6.2 Condition code suffixes

Conditional branch is shown in syntax descriptions as B{*cond*}. A branch instruction with a condition code is only taken if the condition code flags in the APSR meet the specified condition, otherwise the branch instruction is ignored. shows the condition codes to use.

Table 396 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 396. Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal, last flag setting result was zero
NE	Z = 0	Not equal, last flag setting result was non-zero
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative

Table 396. 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.

23.4.4 Memory access instructions

Table 397 shows the memory access instructions:

Table 397. Access instructions

Mnemonic	Brief description	See
LDR{type}	Load Register using register offset	Section 23–23.4.4.3
LDR	Load Register from PC-relative address	Section 23–23.4.4.4
POP	Pop registers from stack	Section 23–23.4.4.6
PUSH	Push registers onto stack	Section 23–23.4.4.6
STM	Store Multiple registers	Section 23–23.4.4.5
STR{type}	Store Register using immediate offset	Section 23–23.4.4.2
STR{type}	Store Register using register offset	Section 23–23.4.4.3

23.4.4.1 ADR

Generates a PC-relative address.

23.4.4.1.1 Syntax

ADR *Rd*, *label*

where:

Rd is the destination register.

label is a PC-relative expression. See Section 23–23.4.3.5.

23.4.4.1.2 Operation

ADR generates an address by adding an immediate value to the PC, and writes the result to the destination register.

ADR facilitates the generation of 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.

23.4.4.1.3 Restrictions

In this instruction *Rd* must specify R0-R7. The data-value addressed must be word aligned and within 1020 bytes of the current PC.

23.4.4.1.4 Condition flags

This instruction does not change the flags.

23.4.4.1.5 Examples

```
ADR    R1, TextMessage   ; Write address value of a location labelled as
                               ; TextMessage to R1

ADR    R3, [PC,#996]     ; Set R3 to value of PC + 996.
```

23.4.4.2 LDR and STR, immediate offset

Load and Store with immediate offset.

23.4.4.2.1 Syntax

LDR *Rt*, [<*Rn* | SP> {, #*imm*}]

LDR<B|H> *Rt*, [*Rn* {, #*imm*}]

STR *Rt*, [<*Rn* | SP>, {,#*imm*}]

STR<B|H> *Rt*, [*Rn* {,#*imm*}]

where:

Rt is the register to load or store.

Rn is the register on which the memory address is based.

imm is an offset from *Rn*. If *imm* is omitted, it is assumed to be zero.

23.4.4.2.2 Operation

LDR, LDRB and LDRH instructions load the register specified by *Rt* with either a word, byte or halfword data value from memory. Sizes less than word are zero extended to 32-bits before being written to the register specified by *Rt*.

STR, STRB and STRH instructions store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* in to memory. The memory address to load from or store to is the sum of the value in the register specified by either *Rn* or SP and the immediate value *imm*.

23.4.4.2.3 Restrictions

In these instructions:

- *Rt* and *Rn* must only specify R0-R7.

- *imm* must be between:
  - 0 and 1020 and an integer multiple of four for LDR and STR using SP as the base register
  - 0 and 124 and an integer multiple of four for LDR and STR using R0-R7 as the base register
  - 0 and 62 and an integer multiple of two for LDRH and STRH
  - 0 and 31 for LDRB and STRB.
- The computed address must be divisible by the number of bytes in the transaction, see Section 23–23.4.3.4.

23.4.4.2.4 Condition flags

These instructions do not change the flags.

23.4.4.2.5 Examples

```
LDR     R4, [R7                ; Loads R4 from the address in R7.
STR     R2, [R0,#const-struc]  ; const-struc is an expression evaluating
                               ; to a constant in the range 0-1020.
```

23.4.4.3 LDR and STR, register offset

Load and Store with register offset.

23.4.4.3.1 Syntax

LDR *Rt*, [*Rn*, *Rm*]

LDR<B|H> *Rt*, [*Rn*, *Rm*]

LDR<SB|SH> *Rt*, [*Rn*, *Rm*]

STR *Rt*, [*Rn*, *Rm*]

STR<B|H> *Rt*, [*Rn*, *Rm*]

where:

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.

23.4.4.3.2 Operation

LDR, LDRB, U, LDRSB and LDRSH load the register specified by *Rt* with either a word, zero extended byte, zero extended halfword, sign extended byte or sign extended halfword value from memory.

STR, STRB and STRH store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* into memory.

The memory address to load from or store to is the sum of the values in the registers specified by *Rn* and *Rm*.

23.4.4.3.3 Restrictions

In these instructions:

- *Rt*, *Rn*, and *Rm* must only specify R0-R7.
- the computed memory address must be divisible by the number of bytes in the load or store, see Section 23–23.4.3.4.

23.4.4.3.4 Condition flags

These instructions do not change the flags.

23.4.4.3.5 Examples

```
STR    R0, [R5, R1]         ; Store value of R0 into an address equal to
                                ; sum of R5 and R1

LDRSH  R1, [R2, R3]         ; Load a halfword from the memory address
                                ; specified by (R2 + R3), sign extend to 32-bits

                            ; and write to R1.
```

23.4.4.4 LDR, PC-relative

Load register (literal) from memory.

23.4.4.4.1 Syntax

LDR *Rt*, *label*

where:

Rt is the register to load.

label is a PC-relative expression. See Section 23–23.4.3.5.

23.4.4.4.2 Operation

Loads the register specified by *Rt* from the word in memory specified by *label*.

23.4.4.4.3 Restrictions

In these instructions, *label* must be within 1020 bytes of the current PC and word aligned.

23.4.4.4.4 Condition flags

These instructions do not change the flags.

23.4.4.4.5 Examples

```
LDR     R0, LookUpTable   ; Load R0 with a word of data from an address
                              ; labelled as LookUpTable.

LDR     R3, [PC, #100]    ; Load R3 with memory word at (PC + 100).
```

23.4.4.5 LDM and STM

Load and Store Multiple registers.

23.4.4.5.1 Syntax

LDM *Rn*{!}, reglist

STM *Rn*!, reglist

where:

Rn is the register on which the memory addresses are based.

! writeback suffix.

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 23–23.4.4.5.5.

LDMIA and LDMFD are synonyms for LDM. LDMIA refers to the base register being Incremented After each access. LDMFD refers to its use for popping data from Full Descending stacks.

STMIA and STMEA are synonyms for STM. STMIA refers to the base register being Incremented After each access. STMEA refers to its use for pushing data onto Empty Ascending stacks.

23.4.4.5.2 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*.

The memory addresses used for the accesses are at 4-byte intervals ranging from the value in the register specified by *Rn* to the value in the register specified by *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 in the register specified by *Rn* + 4 **n* is written back to the register specified by *Rn*.

23.4.4.5.3 Restrictions

In these instructions:

- *reglist* and *Rn* are limited to R0-R7.
- the writeback suffix must always be used unless the instruction is an LDM where reglist also contains *Rn*, in which case the writeback suffix must not be used.
- the value in the register specified by *Rn* must be word aligned. See Section 23–23.4.3.4 for more information.
- for STM, if *Rn* appears in *reglist*, then it must be the first register in the list.

23.4.4.5.4 Condition flags

These instructions do not change the flags.

23.4.4.5.5 Examples

```
LDM     R0,{R0,R3,R4}      ; LDMIA is a synonym for LDM
      STMIA   R1!,{R2-R4,R6}
```

23.4.4.5.6 Incorrect examples

```
STM     R5!,{R4,R5,R6} ; Value stored for R5 is unpredictable
      LDM     R2,{}          ; There must be at least one register in the list
```

23.4.4.6 PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

23.4.4.6.1 Syntax

PUSH *reglist*

POP *reglist*

where:

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.

23.4.4.6.2 Operation

PUSH stores registers on the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

POP loads registers from the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

PUSH uses the value in the SP register minus four as the highest memory address,

POP uses the value in the SP register as the lowest memory address, implementing a full-descending stack. On completion,

PUSH updates the SP register to point to the location of the lowest store value,

POP updates the SP register to point to the location above the highest location loaded.

If a POP instruction includes PC in its *reglist*, a branch to this location is performed when the POP instruction has completed. Bit[0] of the value read for the PC is used to update the APSR T-bit. This bit must be 1 to ensure correct operation.

23.4.4.6.3 Restrictions

In these instructions:

- *reglist* must use only R0-R7.
- The exception is LR for a PUSH and PC for a POP.

23.4.4.6.4 Condition flags

These instructions do not change the flags.

23.4.4.6.5 Examples

```
PUSH    {R0,R4-R7}      ; Push R0,R4,R5,R6,R7 onto the stack
        PUSH    {R2,LR}         ; Push R2 and the link-register onto the stack
        POP     {R0,R6,PC}      ; Pop r0,r6 and PC from the stack, then branch to
                                ; the new PC.
```

23.4.5 General data processing instructions

Table 398 shows the data processing instructions:

Table 398. Data processing instructions

Mnemonic	Brief description	See
ADCS	Add with Carry	Section 23–23.4.5.1
ADD{S}	Add	Section 23–23.4.5.1
ANDS	Logical AND	Section 23–23.4.5.2
ASRS	Arithmetic Shift Right	Section 23–23.4.5.3
BICS	Bit Clear	Section 23–23.4.5.2
CMN	Compare Negative	Section 23–23.4.5.4
CMP	Compare	Section 23–23.4.5.4
EORS	Exclusive OR	Section 23–23.4.5.2
LSLS	Logical Shift Left	Section 23–23.4.5.3
LSRS	Logical Shift Right	Section 23–23.4.5.3
MOV{S}	Move	Section 23–23.4.5.5
MULS	Multiply	Section 23–23.4.5.6
MVNS	Move NOT	Section 23–23.4.5.5
ORRS	Logical OR	Section 23–23.4.5.2
REV	Reverse byte order in a word	Section 23–23.4.5.7
REV16	Reverse byte order in each halfword	Section 23–23.4.5.7
REVSH	Reverse byte order in bottom halfword and sign extend	Section 23–23.4.5.7
RORS	Rotate Right	Section 23–23.4.5.3
RSBS	Reverse Subtract	Section 23–23.4.5.1
SBCS	Subtract with Carry	Section 23–23.4.5.1
SUBS	Subtract	Section 23–23.4.5.1
SXTB	Sign extend a byte	Section 23–23.4.5.8
SXTH	Sign extend a halfword	Section 23–23.4.5.8
UXTB	Zero extend a byte	Section 23–23.4.5.8
UXTH	Zero extend a halfword	Section 23–23.4.5.8
TST	Test	Section 23–23.4.5.9

23.4.5.1 ADC, ADD, RSB, SBC, and SUB

Add with carry, Add, Reverse Subtract, Subtract with carry, and Subtract.

23.4.5.1.1 Syntax

ADCS {*Rd*,} *Rn*, *Rm*

ADD{S} {*Rd*,} *Rn*, <*Rm*|*#imm*>

RSBS {*Rd*,} *Rn*, *Rm*, *#0*

SBCS {*Rd*,} *Rn*, *Rm*

SUB{S} {*Rd*,} *Rn*,

<*Rm*|*#imm*>

Where:

S causes an ADD or SUB instruction to update flags

Rd specifies the result register

Rn specifies the first source register

Rm specifies the second source register

imm specifies a constant immediate value.

When the optional *Rd* register specifier is omitted, it is assumed to take the same value as *Rn*, for example ADDS R1,R2 is identical to ADDS R1,R1,R2.

23.4.5.1.2 Operation

The ADCS instruction adds the value in *Rn* to the value in *Rm*, adding a further one if the carry flag is set, places the result in the register specified by *Rd* and updates the N, Z, C, and V flags.

The ADD instruction adds the value in *Rn* to the value in *Rm* or an immediate value specified by *imm* and places the result in the register specified by *Rd*.

The ADDS instruction performs the same operation as ADD and also updates the N, Z, C and V flags.

The RSBS instruction subtracts the value in *Rn* from zero, producing the arithmetic negative of the value, and places the result in the register specified by Rd and updates the N, Z, C and V flags.

The SBCS instruction subtracts the value of *Rm* from the value in *Rn*, deducts a further one if the carry flag is set. It places the result in the register specified by Rd and updates the N, Z, C and V flags.

The SUB instruction subtracts the value in *Rm* or the immediate specified by *imm*. It places the result in the register specified by *Rd*.

The SUBS instruction performs the same operation as SUB and also updates the N, Z, C and V flags.

Use ADC and SBC to synthesize multiword arithmetic, see Section 23.4.5.1.4.

See also Section 23–23.4.4.1.

23.4.5.1.3 Restrictions

Table 399 lists the legal combinations of register specifiers and immediate values that can be used with each instruction.

Table 399. ADC, ADD, RSB, SBC and SUB operand restrictions

Instruction	Rd	Rn	Rm	imm	Restrictions
ADCS	R0-R7	R0-R7	R0-R7	-	*Rd* and *Rn* must specify the same register.
ADD	R0-R15	R0-R15	R0-PC	-	*Rd* and *Rn* must specify the same register. *Rn* and *Rm* must not both specify PC.
	R0-R7	SP or PC	-	0-1020	Immediate value must be an integer multiple of four.
	SP	SP	-	0-508	Immediate value must be an integer multiple of four.
ADDS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	*Rd* and *Rn* must specify the same register.
	R0-R7	R0-R7	R0-R7	-	-
RSBS	R0-R7	R0-R7	-	-	-
SBCS	R0-R7	R0-R7	R0-R7	-	*Rd* and *Rn* must specify the same register.
SUB	SP	SP	-	0-508	Immediate value must be an integer multiple of four.
SUBS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	Rd and *Rn* must specify the same register.
	R0-R7	R0-R7	R0-R7	-	-

23.4.5.1.4 Examples

The following shows two instructions that add a 64-bit integer contained in R0 and R1 to another 64-bit integer contained in R2 and R3, and place the result in R0 and R1.

64-bit addition:

```
ADDS    R0, R0, R2    ; add the least significant words
ADCS    R1, R1, R3    ; add the most significant words with carry
```

Multiword values do not have to use consecutive registers. The following shows instructions that subtract a 96-bit integer contained in R1, R2, and R3 from another contained in R4, R5, and R6. The example stores the result in R4, R5, and R6.

96-bit subtraction:

```
SUBS    R4, R4, R1    ; subtract the least significant words
SBCS    R5, R5, R2    ; subtract the middle words with carry
SBCS    R6, R6, R3    ; subtract the most significant words with carry
```

The following shows the RSBS instruction used to perform a 1's complement of a single register.

Arithmetic negation:

```
RSBS    R7, R7, #0    ; subtract R7 from zero
```

23.4.5.2 AND, ORR, EOR, and BIC

Logical AND, OR, Exclusive OR, and Bit Clear.

23.4.5.2.1 Syntax

ANDS {*Rd,*} *Rn*, *Rm*

ORRS {*Rd,*} *Rn*, *Rm*

EORS {*Rd,*} *Rn*, *Rm*

BICS {*Rd,*} *Rn*, *Rm*

where:

Rd is the destination register.

Rn is the register holding the first operand and is the same as the destination register.

Rm second register.

23.4.5.2.2 Operation

The AND, EOR, and ORR instructions perform bitwise AND, exclusive OR, and inclusive OR operations on the values in *Rn* and *Rm*.

The BIC instruction performs an AND operation on the bits in *Rn* with the logical negation of the corresponding bits in the value of *Rm*.

The condition code flags are updated on the result of the operation, see Section 23.4.3.6.1.

23.4.5.2.3 Restrictions

In these instructions, *Rd*, *Rn*, and *Rm* must only specify R0-R7.

23.4.5.2.4 Condition flags

These instructions:

- update the N and Z flags according to the result
- do not affect the C or V flag.

23.4.5.2.5 Examples

```
ANDS    R2, R2, R1
     ORRS    R2, R2, R5
     ANDS    R5, R5, R8
     EORS    R7, R7, R6
     BICS    R0, R0, R1
```

23.4.5.3 ASR, LSL, LSR, and ROR

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, and Rotate Right.

23.4.5.3.1 Syntax

ASRS {Rd,} *Rm*, *Rs*

ASRS {Rd,} *Rm*, *#imm*

LSLS {Rd,} *Rm*, *Rs*

LSLS {Rd,} *Rm*, *#imm*

LSRS {Rd,} *Rm*, *Rs*

LSRS {Rd,} *Rm*, *#imm*

RORS {Rd,} *Rm*, *Rs*

where:

Rd is the destination register. If *Rd* is omitted, it is assumed to take the same value as *Rm*.

Rm is the register holding the value to be shifted.

Rs is the register holding the shift length to apply to the value in *Rm*.

imm 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.

Remark: MOVS Rd, *Rm* is a pseudonym for LSLS Rd, *Rm*, #0.

23.4.5.3.2 Operation

ASR, LSL, LSR, and ROR perform an arithmetic-shift-left, logical-shift-left, logical-shift-right or a right-rotation of the bits in the register *Rm* by the number of places specified by the immediate *imm* or the value in the least-significant byte of the register specified by *Rs*.

For details on what result is generated by the different instructions, see Section 23–23.4.3.3.

23.4.5.3.3 Restrictions

In these instructions, *Rd*, *Rm*, and *Rs* must only specify R0-R7. For non-immediate instructions, *Rd* and *Rm* must specify the same register.

23.4.5.3.4 Condition flags

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 23–23.4.3.3. The V flag is left unmodified.

23.4.5.3.5 Examples

```
ASRS    R7, R5, #9  ; Arithmetic shift right by 9 bits
     LSLS    R1, R2, #3  ; Logical shift left by 3 bits with flag update
     LSRS    R4, R5, #6  ; Logical shift right by 6 bits
     RORS    R4, R4, R6  ; Rotate right by the value in the bottom byte of R6.
```

23.4.5.4 CMP and CMN

Compare and Compare Negative.

23.4.5.4.1 Syntax

CMN *Rn*, *Rm*

CMP *Rn*, *#imm*

CMP *Rn*, *Rm*

where:

Rn is the register holding the first operand.

Rm is the register to compare with.

imm is the immediate value to compare with.

23.4.5.4.2 Operation

These instructions compare the value in a register with either the value in another register or an immediate value. They update the condition flags on the result, but do not write the result to a register.

The CMP instruction subtracts either the value in the register specified by *Rm*, or the immediate *imm* from the value in *Rn* and updates the flags. This is the same as a SUBS instruction, except that the result is discarded.

The CMN instruction adds the value of *Rm* to the value in *Rn* and updates the flags. This is the same as an ADDS instruction, except that the result is discarded.

23.4.5.4.3 Restrictions

For the:

- CMN

  instruction *Rn*, and *Rm* must only specify R0-R7.
- CMP instruction:
  - *Rn* and *Rm* can specify R0-R14
  - immediate must be in the range 0-255.

23.4.5.4.4 Condition flags

These instructions update the N, Z, C and V flags according to the result.

23.4.5.4.5 Examples

```
CMP     R2, R9
     CMN     R0, R2
```

23.4.5.5 MOV and MVN

Move and Move NOT.

23.4.5.5.1 Syntax

MOV{S} *Rd*, *Rm*

MOVS *Rd*, #*imm*

MVNS *Rd*, *Rm*

where:

S is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see Section 23–23.4.3.6.

Rd is the destination register.

Rm is a register.

imm is any value in the range 0-255.

23.4.5.5.2 Operation

The MOV instruction copies the value of *Rm* into *Rd*.

The MOVS instruction performs the same operation as the MOV instruction, but also updates the N and Z flags.

The MVNS instruction takes the value of *Rm*, performs a bitwise logical negate operation on the value, and places the result into *Rd*.

23.4.5.5.3 Restrictions

In these instructions, *Rd*, and *Rm* must only specify R0-R7.

When *Rd* is the PC in a MOV instruction:

- Bit[0] of the result is discarded.
- A branch occurs to the address created by forcing bit[0] of the result to 0. The T-bit remains unmodified.

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.

23.4.5.5.4 Condition flags

If *S* is specified, these instructions:

- update the N and Z flags according to the result
- do not affect the C or V flags.

23.4.5.5.5 Example

```
MOVS  R0, #0x000B     ; Write value of 0x000B to R0, flags get updated
      MOVS  R1, #0x0        ; Write value of zero to R1, flags are updated
      MOV   R10, R12        ; Write value in R12 to R10, flags are not updated
      MOVS  R3, #23         ; Write value of 23 to R3
      MOV   R8, SP          ; Write value of stack pointer to R8
      MVNS  R2, R0          ; Write inverse of R0 to the R2 and update flags
```

23.4.5.6 MULS

Multiply using 32-bit operands, and producing a 32-bit result.

23.4.5.6.1 Syntax

MULS Rd, *Rn*, *Rm*

where:

Rd is the destination register.

Rn, Rm are registers holding the values to be multiplied.

23.4.5.6.2 Operation

The MUL instruction multiplies the values in the registers specified by *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*. The condition code flags are updated on the result of the operation, see Section 23–23.4.3.6.

The results of this instruction does not depend on whether the operands are signed or unsigned.

23.4.5.6.3 Restrictions

In this instruction:

- *Rd*, *Rn*, and *Rm* must only specify R0-R7
- *Rd* must be the same as *Rm*.

23.4.5.6.4 Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

23.4.5.6.5 Examples

```
MULS    R0, R2, R0      ; Multiply with flag update, R0 = R0 x R2
```

23.4.5.7 REV, REV16, and REVSH

Reverse bytes.

23.4.5.7.1 Syntax

REV Rd, *Rn*

REV16 Rd, *Rn*

REVSH Rd, *Rn*

where:

Rd is the destination register.

Rn is the source register.

23.4.5.7.2 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 two packed 16-bit big-endian data into little-endian data or two packed 16-bit little-endian data into big-endian data.

REVSH — converts 16-bit signed big-endian data into 32-bit signed little-endian data or 16-bit signed little-endian data into 32-bit signed big-endian data.

23.4.5.7.3 Restrictions

In these instructions, *Rd*, and *Rn* must only specify R0-R7.

23.4.5.7.4 Condition flags

These instructions do not change the flags.

23.4.5.7.5 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
```

23.4.5.8 SXT and UXT

Sign extend and Zero extend.

23.4.5.8.1 Syntax

SXTB Rd, *Rm*

SXTH Rd, *Rm*

UXTB Rd, *Rm*

UXTH Rd, *Rm*

where:

Rd is the destination register.

Rm is the register holding the value to be extended.

23.4.5.8.2 Operation

These instructions 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.

23.4.5.8.3 Restrictions

In these instructions, *Rd* and *Rm* must only specify R0-R7.

23.4.5.8.4 Condition flags

These instructions do not affect the flags.

23.4.5.8.5 Examples

```
SXTH  R4, R6          ; Obtain the lower halfword of the
                           ; value in R6 and then sign extend to
                           ; 32 bits and write the result to R4.
     UXTB  R3, R1          ; Extract lowest byte of the value in R10 and zero
                           ; extend it, and write the result to R3
```

23.4.5.9 TST

Test bits.

23.4.5.9.1 Syntax

TST *Rn*, *Rm*

where:

Rn is the register holding the first operand.

Rm the register to test against.

23.4.5.9.2 Operation

This instruction tests the value in a register against another register. It updates the condition flags based on the result, but does not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value in *Rm*. 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 a register that has that bit set to 1 and all other bits cleared to 0.

23.4.5.9.3 Restrictions

In these instructions, *Rn* and *Rm* must only specify R0-R7.

23.4.5.9.4 Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

23.4.5.9.5 Examples

```
TST     R0, R1  ; Perform bitwise AND of R0 value and R1 value,
                ; condition code flags are updated but result is discarded
```

23.4.6 Branch and control instructions

Table 400 shows the branch and control instructions:

Table 400. Branch and control instructions

Mnemonic	Brief description	See
B{cc}	Branch {conditionally}	Section 23–23.4.6.1
BL	Branch with Link	Section 23–23.4.6.1
BLX	Branch indirect with Link	Section 23–23.4.6.1
BX	Branch indirect	Section 23–23.4.6.1

23.4.6.1 B, BL, BX, and BLX

Branch instructions.

23.4.6.1.1 Syntax

B{*cond*} *label*

BL *label*

BX *Rm*

BLX *Rm*

where:

cond is an optional condition code, see Section 23–23.4.3.6.

label is a PC-relative expression. See Section 23–23.4.3.5.

Rm is a register providing the address to branch to.

23.4.6.1.2 Operation

All these instructions cause a branch to the address indicated by *label* or contained in the register specified by *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 result in a HardFault exception if bit[0] of *Rm* is 0.

BL and BLX instructions also set bit[0] of the LR to 1. This ensures that the value is suitable for use by a subsequent POP {PC} or BX instruction to perform a successful return branch.

Table 401 shows the ranges for the various branch instructions.

Table 401. Branch ranges

Instruction	Branch range
B *label*	–2 KB to +2 KB
B*cond label*	–256 bytes to +254 bytes
BL *label*	–16 MB to +16 MB
BX *Rm*	Any value in register
BLX *Rm*	Any value in register

23.4.6.1.3 Restrictions

In these instructions:

- Do not use SP or PC in the BX or BLX instruction.
- For BX and BLX, bit[0] of *Rm* must be 1 for correct execution. Bit[0] is used to update the EPSR T-bit and is discarded from the target address.

Remark: B*cond* is the only conditional instruction on the Cortex-M0 processor.

23.4.6.1.4 Condition flags

These instructions do not change the flags.

23.4.6.1.5 Examples

```
B      loopA  ; Branch to loopA
      BL     funC   ; Branch with link (Call) to function funC, return address
                    ; stored in LR
```

```
        BX      LR      ; Return from function call
        BLX     R0      ; Branch with link and exchange (Call) to a address stored
                        ; in R0

    BEQ     labelD ; Conditionally branch to labelD if last flag setting

                    ; instruction set the Z flag, else do not branch.
```

23.4.7 Miscellaneous instructions

Table 402 shows the remaining Cortex-M0 instructions:

Table 402. Miscellaneous instructions

Mnemonic	Brief description	See
BKPT	Breakpoint	Section 23–23.4.7.1
CPSID	Change Processor State, Disable Interrupts	Section 23–23.4.7.2
CPSIE	Change Processor State, Enable Interrupts	Section 23–23.4.7.2
DMB	Data Memory Barrier	Section 23–23.4.7.3
DSB	Data Synchronization Barrier	Section 23–23.4.7.4
ISB	Instruction Synchronization Barrier	Section 23–23.4.7.5
MRS	Move from special register to register	Section 23–23.4.7.6
MSR	Move from register to special register	Section 23–23.4.7.7
NOP	No Operation	Section 23–23.4.7.8
SEV	Send Event	Section 23–23.4.7.9
SVC	Supervisor Call	Section 23–23.4.7.10
WFE	Wait For Event	Section 23–23.4.7.11
WFI	Wait For Interrupt	Section 23–23.4.7.12

23.4.7.1 BKPT

Breakpoint.

23.4.7.1.1 Syntax

BKPT *#imm*

where:

imm is an integer in the range 0-255.

23.4.7.1.2 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 processor might also produce a HardFault or go in to lockup if a debugger is not attached when a BKPT instruction is executed. See Section 23–23.3.4.1 for more information.

23.4.7.1.3 Restrictions

There are no restrictions.

23.4.7.1.4 Condition flags

This instruction does not change the flags.

23.4.7.1.5 Examples

```
BKPT #0     ; Breakpoint with immediate value set to 0x0.
```

23.4.7.2 CPS

Change Processor State.

23.4.7.2.1 Syntax

CPSID i

CPSIE i

23.4.7.2.2 Operation

CPS changes the PRIMASK special register values. CPSID causes interrupts to be disabled by setting PRIMASK. CPSIE cause interrupts to be enabled by clearing PRIMASK.See Section 23–23.3.1.3.6 for more information about these registers.

23.4.7.2.3 Restrictions

There are no restrictions.

23.4.7.2.4 Condition flags

This instruction does not change the condition flags.

23.4.7.2.5 Examples

```
CPSID i ; Disable all interrupts except NMI (set PRIMASK)

CPSIE i ; Enable interrupts (clear PRIMASK)
```

23.4.7.3 DMB

Data Memory Barrier.

23.4.7.3.1 Syntax

DMB

23.4.7.3.2 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 observed before any explicit memory accesses that appear in program order after the DMB instruction. DMB does not affect the ordering of instructions that do not access memory.

23.4.7.3.3 Restrictions

There are no restrictions.

23.4.7.3.4 Condition flags

This instruction does not change the flags.

23.4.7.3.5 Examples

```
DMB  ; Data Memory Barrier
```

23.4.7.4 DSB

Data Synchronization Barrier.

23.4.7.4.1 Syntax

DSB

23.4.7.4.2 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.

23.4.7.4.3 Restrictions

There are no restrictions.

23.4.7.4.4 Condition flags

This instruction does not change the flags.

23.4.7.4.5 Examples

```
DSB ; Data Synchronisation Barrier
```

23.4.7.5 ISB

Instruction Synchronization Barrier.

23.4.7.5.1 Syntax

ISB

23.4.7.5.2 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.

23.4.7.5.3 Restrictions

There are no restrictions.

23.4.7.5.4 Condition flags

This instruction does not change the flags.

23.4.7.5.5 Examples

```
ISB  ; Instruction Synchronisation Barrier
```

23.4.7.6 MRS

Move the contents of a special register to a general-purpose register.

23.4.7.6.1 Syntax

MRS *Rd*, *spec_reg*

where:

Rd is the general-purpose destination register.

spec_reg is one of the special-purpose registers: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

23.4.7.6.2 Operation

MRS stores the contents of a special-purpose register to a general-purpose register. The MRS instruction can be combined with the MR instruction to produce read-modify-write sequences, which are suitable for modifying a specific flag in the PSR.

See Section 23–23.4.7.7.

23.4.7.6.3 Restrictions

In this instruction, *Rd* must not be SP or PC.

23.4.7.6.4 Condition flags

This instruction does not change the flags.

23.4.7.6.5 Examples

```
MRS  R0, PRIMASK ; Read PRIMASK value and write it to R0
```

23.4.7.7 MSR

Move the contents of a general-purpose register into the specified special register.

23.4.7.7.1 Syntax

MSR *spec_reg*, *Rn*

where:

Rn is the general-purpose source register.

spec_reg is the special-purpose destination register: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

23.4.7.7.2 Operation

MSR updates one of the special registers with the value from the register specified by *Rn*.

See Section 23–23.4.7.6.

23.4.7.7.3 Restrictions

In this instruction, *Rn* must not be SP and must not be PC.

23.4.7.7.4 Condition flags

This instruction updates the flags explicitly based on the value in *Rn*.

23.4.7.7.5 Examples

```
MSR  CONTROL, R1 ; Read R1 value and write it to the CONTROL register
```

23.4.7.8 NOP

No Operation.

23.4.7.8.1 Syntax

NOP

23.4.7.8.2 Operation

NOP performs no operation and is not guaranteed to be time consuming. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the subsequent instructions on a 64-bit boundary.

23.4.7.8.3 Restrictions

There are no restrictions.

23.4.7.8.4 Condition flags

This instruction does not change the flags.

23.4.7.8.5 Examples

```
NOP  ; No operation
```

23.4.7.9 SEV

Send Event.

23.4.7.9.1 Syntax

SEV

23.4.7.9.2 Operation

SEV causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register, see Section 23–23.3.5.

See also Section 23–23.4.7.11.

23.4.7.9.3 Restrictions

There are no restrictions.

23.4.7.9.4 Condition flags

This instruction does not change the flags.

23.4.7.9.5 Examples

```
SEV ; Send Event
```

23.4.7.10 SVC

Supervisor Call.

23.4.7.10.1 Syntax

SVC #*imm*

where:

imm is an integer in the range 0-255.

23.4.7.10.2 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.

23.4.7.10.3 Restrictions

There are no restrictions.

23.4.7.10.4 Condition flags

This instruction does not change the flags.

23.4.7.10.5 Examples

```
SVC  #0x32 ; Supervisor Call (SVC handler can extract the immediate value
               ; by locating it via the stacked PC)
```

23.4.7.11 WFE

Wait For Event.

Remark: The WFE instruction is not implemented on the LPC11Uxx.

23.4.7.11.1 Syntax

WFE

23.4.7.11.2 Operation

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 completes immediately.

For more information see Section 23–23.3.5.

Remark: WFE is intended for power saving only. When writing software assume that WFE might behave as NOP.

23.4.7.11.3 Restrictions

There are no restrictions.

23.4.7.11.4 Condition flags

This instruction does not change the flags.

23.4.7.11.5 Examples

```
WFE  ; Wait for event
```

23.4.7.12 WFI

Wait for Interrupt.

23.4.7.12.1 Syntax

WFI

23.4.7.12.2 Operation

WFI

suspends execution until one of the following events occurs:

- an exception
- an interrupt becomes pending which would preempt if PRIMASK was clear
- a Debug Entry request, regardless of whether debug is enabled.

Remark: WFI is intended for power saving only. When writing software assume that WFI might behave as a NOP operation.

23.4.7.12.3 Restrictions

There are no restrictions.

23.4.7.12.4 Condition flags

This instruction does not change the flags.

23.4.7.12.5 Examples

```
WFI ; Wait for interrupt
```

23.5 Peripherals

23.5.1 About the ARM Cortex-M0

The address map of the **Private peripheral bus** (PPB) is:

Table 403. Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System Control Block	Table 23–412
0xE000E010-0xE000E01F	System timer	Table 23–421
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 23–404
0xE000ED00-0xE000ED3F	System Control Block	Table 23–412
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 23–404

In register descriptions, the register **type** is described as follows:

RW — Read and write.

RO — Read-only.

WO — Write-only.

23.5.2 Nested Vectored Interrupt Controller

This section describes the **Nested Vectored Interrupt Controller** (NVIC) and the registers it uses. The NVIC supports:

- 32 interrupts.
- A programmable priority level of 0-3 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.
- 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 404. NVIC register summary

Address	Name	Type	Reset value	Description
0xE000E100	ISER	RW	0x00000000	Section 23–23.5.2.2
0xE000E180	ICER	RW	0x00000000	Section 23–23.5.2.3
0xE000E200	ISPR	RW	0x00000000	Section 23–23.5.2.4
0xE000E280	ICPR	RW	0x00000000	Section 23–23.5.2.5
0xE000E400-0xE000E41C	IPR0-7	RW	0x00000000	Section 23–23.5.2.6

23.5.2.1 Accessing the Cortex-M0 NVIC registers using CMSIS

CMSIS functions enable software portability between different Cortex-M profile processors.

To access the NVIC registers when using CMSIS, use the following functions:

Table 405. CMISIS acess NVIC functions

CMSIS function	Description
void NVIC_EnableIRQ(IRQn_Type IRQn)[1]	Enables an interrupt or exception.
void NVIC_DisableIRQ(IRQn_Type IRQn)[1]	Disables an interrupt or exception.
void NVIC_SetPendingIRQ(IRQn_Type IRQn)[1]	Sets the pending status of interrupt or exception to 1.
void NVIC_ClearPendingIRQ(IRQn_Type IRQn)[1]	Clears the pending status of interrupt or exception to 0.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)[1]	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)[1]	Sets the priority of an interrupt or exception with configurable priority level to 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn)[1]	Reads the priority of an interrupt or exception with configurable priority level. This function returns the current priority level.

[1] The input parameter IRQn is the IRQ number, see Table 391 for more information.

23.5.2.2 Interrupt Set-enable Register

The ISER enables interrupts, and shows which interrupts are enabled. See the register summary in Table 404 for the register attributes.

The bit assignments are:

Table 406. 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.

23.5.2.3 Interrupt Clear-enable Register

The ICER disables interrupts, and show which interrupts are enabled. See the register summary in Table 23–404 for the register attributes.

The bit assignments are:

Table 407. 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.

23.5.2.4 Interrupt Set-pending Register

The ISPR forces interrupts into the pending state, and shows which interrupts are pending. See the register summary in Table 23–404 for the register attributes.

The bit assignments are:

Table 408. 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.

23.5.2.5 Interrupt Clear-pending Register

The ICPR removes the pending state from interrupts, and shows which interrupts are pending. See the register summary in Table 23–404 for the register attributes.

The bit assignments are:

Table 409. 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.

23.5.2.6 Interrupt Priority Registers

The IPR0-IPR7 registers provide an 2-bit priority field for each interrupt. These registers are only word-accessible. See the register summary in Table 23–404 for their attributes. Each register holds four priority fields as shown:

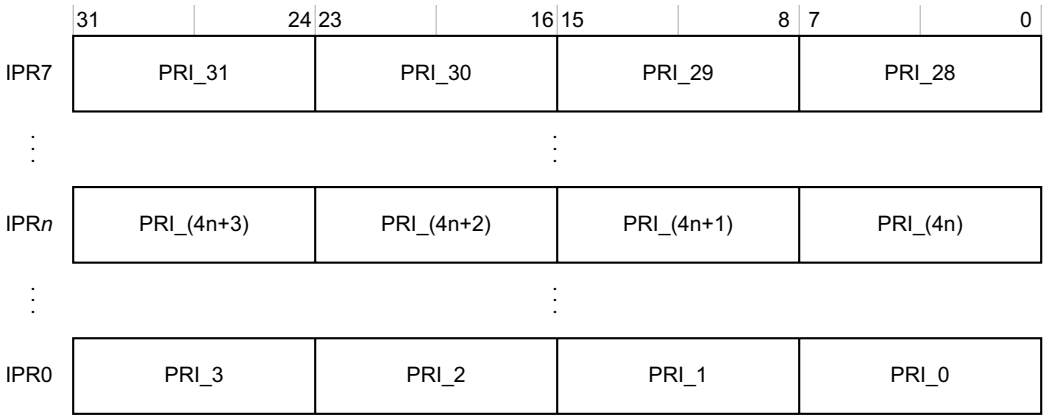


Fig 84. IPR register

Table 410. IPR bit assignments

Bits	Name	Function
[31:24]	Priority, byte offset 3	Each priority field holds a priority value, 0-3. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:6] of each field, bits [5:0] 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 Section 23–23.5.2.1 for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt **M** as follows:

- the corresponding IPR number, **N**, is given by **N** = **N** DIV 4
- the byte offset of the required Priority field in this register is **M** 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].

23.5.2.7 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 23.5.2.7.1. 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.

23.5.2.7.1 Hardware and software control of interrupts

The Cortex-M0 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is active and the corresponding 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 Section 23–23.5.2.4.

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.

23.5.2.8 NVIC usage hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

23.5.2.8.1 NVIC programming hints

Software uses the `CPSIE i` and 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 411. CMSIS functions for NVIC control

CMSIS interrupt control function	Description
`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 (1) if IRQn is pending
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	Clear IRQn pending status
`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

The input parameter IRQn is the IRQ number, see Table 23–391 for more information. For more information about these functions, see the CMSIS documentation.

23.5.3 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 SCB registers are:

Table 412. Summary of the SCB registers

Address	Name	Type	Reset value	Description
`0xE000ED00`	CPUID	RO	`0x410CC200`	Section 23.5.3.2
`0xE000ED04`	ICSR	RW[1]	`0x00000000`	Section 23–23.5.3.3
`0xE000ED0C`	AIRCR	RW[1]	`0xFA050000`	Section 23–23.5.3.4
`0xE000ED10`	SCR	RW	`0x00000000`	Section 23–23.5.3.5
`0xE000ED14`	CCR	RO	`0x00000204`	Section 23–23.5.3.6
`0xE000ED1C`	SHPR2	RW	`0x00000000`	Section 23–23.5.3.7.1
`0xE000ED20`	SHPR3	RW	`0x00000000`	Section 23–23.5.3.7.2

[1] See the register description for more information.

23.5.3.1 The CMSIS mapping of the Cortex-M0 SCB registers

To improve software efficiency, the CMSIS simplifies the SCB register presentation. In the CMSIS, the array `SHP[1]` corresponds to the registers SHPR2-SHPR3.

23.5.3.2 CPUID Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in for its attributes. The bit assignments are:

Table 413. CPUID register bit assignments

Bits	Name	Function
[31:24]	Implementer	Implementer code: 0x41 = ARM
[23:20]	Variant	Variant number, the r value in the r**n**p**n** product revision identifier
[19:16]	Constant	Constant that defines the architecture of the processor:, reads as 0xC = ARMv6-M architecture
[15:4]	Partno	Part number of the processor: 0xC20 = Cortex-M0
[3:0]	Revision	Revision number, the p value in the r**n**p**n** product revision identifier

23.5.3.3 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 23–412 for the ICSR attributes. The bit assignments are:

Table 414. ICSR bit assignments

Bits	Name	Type	Function
[31]	NMIPENDSET	RW	NMI set-pending bit. Write: 0 = no effect 1 = changes NMI exception state to pending. Read: 0 = NMI exception is not pending 1 = NMI exception is pending. Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it detects a write of 1 to this bit. Entering the handler then clears this bit to 0. This means 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.
[30:29]	-	-	Reserved.
[28]	PENDSVSET	RW	PendSV set-pending bit. Write: 0 = no effect 1 = changes PendSV exception state to pending. Read: 0 = PendSV exception is not pending 1 = PendSV exception is pending. Writing 1 to this bit is the only way to set the PendSV exception state to pending.
[27]	PENDSVCLR	WO	PendSV clear-pending bit. Write: 0 = no effect 1 = removes the pending state from the PendSV exception.
[26]	PENDSTSET	RW	SysTick exception set-pending bit. Write: 0 = no effect 1 = changes SysTick exception state to pending. Read: 0 = SysTick exception is not pending 1 = SysTick exception is pending.
[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.
[24:23]	-	-	Reserved.

Table 414. ICSR bit assignments

Bits	Name	Type	Function
[22]	ISRPENDING	RO	Interrupt pending flag, excluding NMI and Faults: 0 = interrupt not pending 1 = interrupt pending.
[21:18]	-	-	Reserved.
[17:12]	VECTPENDING	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.
[11:6]	-	-	Reserved.
[5:0]	VECTACTIVE[1]	RO	Contains the active exception number: 0 = Thread mode Nonzero = The exception number[1] of the currently active exception. **Remark:** Subtract 16 from this value to obtain the CMSIS IRQ number that identifies the corresponding bit in the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-pending, and Priority Register, see Table 23–386.

[1] This is the same value as IPSR bits[5:0], see Table 23–386.

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.

23.5.3.4 Application Interrupt and Reset Control Register

The AIRCR provides endian status for data accesses and reset control of the system. See the register summary in Table 23–412 and Table 23–415 for its attributes.

To write to this register, you must write `0x05FA` to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

Table 415. AIRCR bit assignments

Bits	Name	Type	Function
[31:16]	Read: Reserved Write: VECTKEY	RW	Register key: Reads as Unknown On writes, write `0x05FA` to VECTKEY, otherwise the write is ignored.
[15]	ENDIANESS	RO	Data endianness implemented: 0 = Little-endian 1 = Big-endian.
[14:3]	-	-	Reserved

Table 415. AIRCR bit assignments

Bits	Name	Type	Function
[2]	SYSRESETREQ	WO	System reset request: 0 = no effect 1 = requests a system level reset. 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]	-	-	Reserved.

23.5.3.5 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in Table 23–412 for its attributes. The bit assignments are:

Table 416. 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.
[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 to Thread mode. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
[0]	-	Reserved.

23.5.3.6 Configuration and Control Register

The CCR is a read-only register and indicates some aspects of the behavior of the Cortex-M0 processor. See the register summary in Table 23–412 for the CCR attributes.

The bit assignments are:

Table 417. CCR bit assignments

Bits	Name	Function
[31:10]	-	Reserved.
[9]	STKALIGN	Always reads as one, indicates 8-byte stack alignment on exception entry. 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.
[8:4]	-	Reserved.
[3]	UNALIGN_TRP	Always reads as one, indicates that all unaligned accesses generate a HardFault.
[2:0]	-	Reserved.

23.5.3.7 System Handler Priority Registers

The SHPR2-SHPR3 registers set the priority level, 0 to 3, of the exception handlers that have configurable priority.

SHPR2-SHPR3 are word accessible. See the register summary in Table 23–412 for their attributes.

To access to the system exception priority level using CMSIS, use the following CMSIS functions:

- `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
- `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`

The input parameter `IRQn` is the IRQ number, see Table 23–391 for more information.

The system fault handlers, and the priority field and register for each handler are:

Table 418. System fault handler priority fields

Handler	Field	Register description
SVCall	PRI_11	Section 23–23.5.3.7.1
PendSV	PRI_14	Section 23–23.5.3.7.2
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the processor implements only bits[7:6] of each field, and bits[5:0] read as zero and ignore writes.

23.5.3.7.1 System Handler Priority Register 2

The bit assignments are:

Table 419. SHPR2 register bit assignments

Bits	Name	Function
[31:24]	PRI_11	Priority of system handler 11, SVCall
[23:0]	-	Reserved

23.5.3.7.2 System Handler Priority Register 3

The bit assignments are:

Table 420. 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

23.5.3.8 SCB usage hints and tips

Ensure software uses aligned 32-bit word size transactions to access all the SCB registers.

23.5.4 System timer, SysTick

When enabled, the timer counts down from the reload value to zero, reloads (wraps to) the value in the SYST_RVR on the next clock cycle, then decrements on subsequent clock cycles. Writing a value of zero to the SYST_RVR disables the counter on the next wrap. When the counter transitions to zero, the COUNTFLAG status bit is set to 1. Reading SYST_CSR clears the COUNTFLAG bit to 0.

Writing to the SYST_CVR clears the register and the COUNTFLAG status bit to 0. The write does not trigger the SysTick exception logic. Reading the register returns its value at the time it is accessed.

Remark: When the processor is halted for debugging the counter does not decrement.

The system timer registers are:

Table 421. System timer registers summary

Address	Name	Type	Reset value	Description
0xE000E010	SYST_CSR	RW	0x00000000	Section 23.5.4.1
0xE000E014	SYST_RVR	RW	Unknown	Section 23–23.5.4.2
0xE000E018	SYST_CVR	RW	Unknown	Section 23–23.5.4.3
0xE000E01C	SYST_CALIB	RO	0xC0000000 [1]	Section 23–23.5.4.4

[1] SysTick calibration value.

23.5.4.1 SysTick Control and Status Register

The SYST_CSR enables the SysTick features. See the register summary in for its attributes. The bit assignments are:

Table 422. SYST_CSR bit assignments

Bits	Name	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since the last read of this register.
[15:3]	-	Reserved.

Table 422. SYST_CSR bit assignments

Bits	Name	Function
[2]	CLKSOURCE	Selects the SysTick timer clock source: 0 = external reference 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.
[0]	ENABLE	Enables the counter: 0 = counter disabled 1 = counter enabled.

23.5.4.2 SysTick Reload Value Register

The SYST_RVR specifies the start value to load into the SYST_CVR. See the register summary in Table 23–421 for its attributes. The bit assignments are:

Table 423. SYST_RVR bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	RELOAD	Value to load into the SYST_CVR when the counter is enabled and when it reaches 0, see Section 23.5.4.2.1.

23.5.4.2.1 Calculating the RELOAD value

The RELOAD value can be any value in the range `0x00000001-0x00FFFFFF`. You can program a value of 0, but this has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

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.

23.5.4.3 SysTick Current Value Register

The SYST_CVR contains the current value of the SysTick counter. See the register summary in Table 23–421 for its attributes. The bit assignments are:

Table 424. SYST_CVR 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 SYST_CSR.COUNTFLAG bit to 0.

23.5.4.4 SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties. See the register summary in Table 23–421 for its attributes. The bit assignments are:

Table 425. SYST_CALIB register bit assignments

Bits	Name	Function
[31]	NOREF	Reads as one. Indicates that no separate reference clock is provided.
[30]	SKEW	Reads as one. Calibration value for the 10ms inexact timing is not known because TENMS is not known. This can affect the suitability of SysTick as a software real time clock.
[29:24]	-	Reserved.
[23:0]	TENMS	Reads as zero. Indicates calibration value is not known.

If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

23.5.4.5 SysTick usage hints and tips

The interrupt controller clock updates the SysTick counter.

Ensure software uses word accesses to access the SysTick registers.

If the SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

23.6 Cortex-M0 instruction summary

Table 426. Cortex M0- instruction summary

Operation	Description	Assembler	Cycles
Move	8-bit immediate	MOVS Rd, #<imm>	1
	Lo to Lo	MOVS Rd, Rm	1
	Any to Any	MOV Rd, Rm	1
	Any to PC	MOV PC, Rm	3
Add	3-bit immediate	ADDS Rd, Rn, #<imm>	1
	All registers Lo	ADDS Rd, Rn, Rm	1
	Any to Any	ADD Rd, Rd, Rm	1
	Any to PC	ADD PC, PC, Rm	3
Add	8-bit immediate	ADDS Rd, Rd, #<imm>	1
	With carry	ADCS Rd, Rd, Rm	1
	Immediate to SP	ADD SP, SP, #<imm>	1
	Form address from SP	ADD Rd, SP, #<imm>	1
	Form address from PC	ADR Rd, <label>	1

Table 426. Cortex M0- instruction summary

Operation	Description	Assembler	Cycles
Subtract	Lo and Lo	SUBS Rd, Rn, Rm	1
	3-bit immediate	SUBS Rd, Rn, #<imm>	1
	8-bit immediate	SUBS Rd, Rd, #<imm>	1
	With carry	SBCS Rd, Rd, Rm	1
	Immediate from SP	SUB SP, SP, #<imm>	1
	Negate	RSBS Rd, Rn, #0	1
Multiply	Multiply	MULS Rd, Rm, Rd	1
Compare	Compare	CMP Rn, Rm	1
	Negative	CMN Rn, Rm	1
	Immediate	CMP Rn, #<imm>	1
Logical	AND	ANDS Rd, Rd, Rm	1
	Exclusive OR	EORS Rd, Rd, Rm	1
	OR	ORRS Rd, Rd, Rm	1
	Bit clear	BICS Rd, Rd, Rm	1
	Move NOT	MVNS Rd, Rm	1
	AND test	TST Rn, Rm	1
Shift	Logical shift left by immediate	LSLS Rd, Rm, #<shift>	1
	Logical shift left by register	LSLS Rd, Rd, Rs	1
	Logical shift right by immediate	LSRS Rd, Rm, #<shift>	1
	Logical shift right by register	LSRS Rd, Rd, Rs	1
	Arithmetic shift right	ASRS Rd, Rm, #<shift>	1
	Arithmetic shift right by regist	ASRS Rd, Rd, Rs	1
Rotate	Rotate right by register	RORS Rd, Rd, Rs	1
Load	Word, immediate offset	LDR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	LDRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	LDRB Rd, [Rn, #<imm>]	2
	Word, register offset	LDR Rd, [Rn, Rm]	2
	Halfword, register offset	LDRH Rd, [Rn, Rm]	2
	Signed halfword, register offset	LDRSH Rd, [Rn, Rm]	2
	Byte, register offset	LDRB Rd, [Rn, Rm]	2
	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2
	PC-relative	LDR Rd, <label>	2
	SP-relative	LDR Rd, [SP, #<imm>]	2
	Multiple, excluding base	LDM Rn!, {<loreglist>}	1 + N[1]
	Multiple, including base	LDM Rn, {<loreglist>}	1 + N[1]
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2

Table 426. Cortex M0- instruction summary

Operation	Description	Assembler	Cycles
Store	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2
	Word, register offset	STR Rd, [Rn, Rm]	2
	Halfword, register offset	STRH Rd, [Rn, Rm]	2
	Byte, register offset	STRB Rd, [Rn, Rm]	2
	SP-relative	STR Rd, [SP, #<imm>]	2
	Multiple	STM Rn!, {<loreglist>}	1 + N[1]
Push	Push	PUSH {<loreglist>}	1 + N[1]
	Push with link register	PUSH {<loreglist>, LR}	1 + N[1]
Pop	Pop	POP {<loreglist>}	1 + N[1]
	Pop and return	POP {<loreglist>, PC}	4 + N[2]
Branch	Conditional	B<cc> <label>	1 or 3[3]
	Unconditional	B <label>	3
	With link	BL <label>	4
	With exchange	BX Rm	3
	With link and exchange	BLX Rm	3
Extend	Signed halfword to word	SXTH Rd, Rm	1
	Signed byte to word	SXTB Rd, Rm	1
	Unsigned halfword	UXTH Rd, Rm	1
	Unsigned byte	UXTB Rd, Rm	1
Reverse	Bytes in word	REV Rd, Rm	1
	Bytes in both halfwords	REV16 Rd, Rm	1
	Signed bottom half word	REVSH Rd, Rm	1
State change	Supervisor Call	SVC <imm>	-[4]
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS Rd, <specreg>	4
	Write special register	MSR <specreg>, Rn	4
Hint	Send event	SEV	1
	Wait for interrupt	WFI	2[5]
	Yield	YIELD[6]	1
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	4
	Data memory	DMB	4
	Data synchronization	DSB	4

[1] N is the number of elements.

[2] N is the number of elements in the stack-pop list including PC and assumes load or store does not generate a HardFault exception.

[3] 3 if taken, 1 if not taken.

[4] Cycle count depends on core and debug configuration.

[5] Excludes time spend waiting for an interrupt or event.

[6] Executes as NOP.

24.1 Abbreviations

Table 427. Abbreviations

Acronym	Description
A/D	Analog-to-Digital
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
BOD	BrownOut Detection
GPIO	General Purpose Input/Output
JTAG	Joint Action Test Group
PLL	Phase-Locked Loop
RC	Resistor-Capacitor
SPI	Serial Peripheral Interface
SSI	Serial Synchronous Interface
SSP	Synchronous Serial Port
TAP	Test Access Port
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous Asynchronous Receiver/Transmitter

24.2 Legal information

24.2.1 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.

24.2.2 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. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

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 and its suppliers accept 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 competent authorities.

24.2.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I^2C-bus — logo is a trademark of NXP B.V.

24.3 Tables

24.4 Figures

24.5 Contents

Chapter 1: LPC11Uxx Introductory information

Chapter 2: LPC11Uxx Memory mapping

Chapter 3: LPC11Uxx System control block

Chapter 4: LPC11Uxx Power Management Unit (PMU)

Chapter 5: LPC11Uxx Power profiles

Chapter 6: LPC11Uxx NVIC

Chapter 7: LPC11Uxx I/O configuration

Chapter 8: LPC11Uxx Pin configuration

Chapter 9: LPC11Uxx GPIO

Chapter 10: LPC11Uxx USB on-chip drivers

Chapter 11: LPC11Uxx USB2.0 device controller

Chapter 12: LPC11Uxx USART

Chapter 13: LPC11Uxx SSP/SPI

Chapter 14: LPC11Uxx I2C-bus controller

Chapter 15: LPC11Uxx 16-bit counter/timers CT16B0/1

Chapter 16: LPC11Uxx 32-bit counter/timers CT32B0/1

Chapter 17: LPC11Uxx Windowed Watchdog Timer (WWDT)

Chapter 18: LPC11Uxx System tick timer

Chapter 19: LPC11Uxx ADC

Chapter 20: LPC11Uxx Flash programming firmware

Chapter 21: LPC11Uxx Serial Wire Debugger (SWD)

Chapter 22: LPC11Uxx Integer division routines

Chapter 23: LPC11Uxx Appendix ARM Cortex-M0

continued >>

Chapter 24: Supplementary information

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 13 January 2012
Document identifier: UM10462