UM10586

EM783 User manual

Rev. 1 — 14 November 2013 User manual

Document information

Info	Content
Keywords	EM783
Abstract	EM783 User manual

Revision history

Rev	Date	Description
1.0	20131114	Initial version of EM783 user manual

Contact information

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

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

UM10586

Chapter 1: Introductory information

Rev. 1 — 14 November 2013 User manual

1.1 Introduction

The EM783-SC/SP/TP/MC3/MC6 is an ARM Cortex-M0 based, low-cost 32-bit family of application processors, designed for energy measurement and monitoring applications. The EM783 offers programmability and on-chip metrology functionality combined with a low power, simple instruction set. It also has memory addressing together with reduced code size compared to existing 8/16-bit architectures.

The EM783 operate at CPU frequencies of up to 48 MHz.

The digital peripherals on the EM783 include:

- 32 kB of flash memory
- 4 kB of EEPROM data memory
- 8 kB of SRAM data memory
- Fast-mode Plus I^2C-bus interface
- RS-485/EIA-485 USART
- one SSP controller
- two general-purpose counter/timers
- up to 22 general-purpose I/O pins

A metrology engine with built-in temperature sensor is used for energy measurements. A 10-bit DAC and an internal voltage reference are also available.

1.2 Features and benefits

- System:
  - ARM Cortex-M0 processor, running at frequencies of up to 48 MHz.
  - ARM Cortex-M0 built-in Nested Vectored Interrupt Controller (NVIC).
  - Serial Wire Debug (SWD)
  - System tick timer.
- Memory:
  - 32 kB on-chip flash program memory.
  - 4 kB on-chip EEPROM data memory for energy registers and calibration parameters; byte erasable and byte programmable.
  - 8 kB SRAM data memory.
  - 16 kB boot ROM.
  - In-System Programming (ISP) for flash and In-Application Programming (IAP) for flash and EEPROM via on-chip bootloader software.
  - Includes ROM-based 32-bit integer division routines.
- Digital peripherals:

- Up to 22 General Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors, repeater mode, and open-drain mode.
- Up to 9 pins are configurable with a digital input glitch filter for removing glitches with widths of 10 ns, two pins are configurable for 20 ns glitch filter and another two pins are configurable for 50 ns glitch filters.
- GPIO pins can be used as edge and level sensitive interrupt sources.
- High-current source output driver (20 mA) on one pin (P0_21).
- High-current sink driver (20 mA) on true open-drain pins (P0_2 and P0_3).
- Two general-purpose counter/timers with a total of up to four capture inputs and five match outputs.
- Programmable Windowed WatchDog Timer (WWDT) with a dedicated, internal low-power WatchDog Oscillator (WDOsc).

- Analog peripherals:
  - Metrology engine for smart metering with one voltage input, one bias input, from two up to six current inputs and a temperature sensor.
  - Internal voltage reference.
  - 10-bit DAC with flexible conversion triggering.
- Serial interfaces:
  - USART with fractional baud rate generation, internal FIFO, support for RS-485/9-bit mode and synchronous mode.
  - One SSP controller 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 1 Mbit/s with multiple address recognition and monitor mode.
- Clock generation:
  - Crystal Oscillator (SysOsc) with an operating range of 1 MHz to 25 MHz.
  - 12 MHz internal RC Oscillator (IRC) trimmed to 1% accuracy that can optionally be used as a system clock.
  - Internal low-power, Low-Frequency Oscillator (LFOsc) with programmable frequency output.
  - Clock input for external system clock (25 MHz typical).
  - PLL allows CPU operation up to the maximum CPU rate with the IRC, the external clock, or the SysOsc as clock sources.
  - Clock output function with divider that can reflect the SysOsc, the IRC, the main clock, or the LFOsc.
- Power control:
  - Supports ARM Cortex-M0 Sleep mode as reduced power mode.
  - Power profiles residing in boot ROM allowed to optimize performance and minimize power consumption for any given application through one simple function call.
  - Processor wake-up from reduced power mode using any interrupt.
  - Power-On Reset (POR).

  - BrownOut Detect (BOD) with two separate programmable thresholds for interrupt and one hardware controlled reset trip point.
  - POR and BOD are always enabled for rapid UVLO protection against power supply voltage drops below 2.4 V
- Unique device serial number for identification.
- Single 3.3 V power supply (2.6 V to 3.6 V).
- Temperature range −40 °C to +85 °C.
- Available as 33-pin HVQFN 7 mm × 7 mm × 0.85 mm package.

1.3 Applications

- Smart plugs and plug meters
- Single phase residential meters
- Industrial submeters
- Server power monitoring
- Smart appliances

1.4 Ordering information

Table 1. Ordering information

Type number	Package		
	Name	Description	Version
EM783-SC	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
EM783-SP	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
EM783-TP	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
EM783-MC3	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a
EM783-MC6	HVQFN33	HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm	n/a

Table 2. Ordering options

Type number	Flash	SRAM	EEPROM	Metrology engine inputs	10-bit DAC	USART	SSP/ SPI	I^2C	Accuracy (%)	Dynamic range	Package
EM783-SC	32 kB	8 kB	4 kB	1x I, 1x V	1	1	1	1	1	1000	HVQFN33
EM783-SP	32 kB	8 kB	4 kB	2x I, 1x V	1	1	1	1	1	1000	HVQFN33
EM783-TP	32 kB	8 kB	4 kB	3x I, 3x V	1	1	1	1	2	50	HVQFN33
EM783-MC3	32 kB	8 kB	4 kB	3x I, 1x V	1	1	1	1	1	1000	HVQFN33
EM783-MC6	32 kB	8 kB	4 kB	6x I, 1x V	1	1	1	1	2	50	HVQFN33

1.5 Block diagram

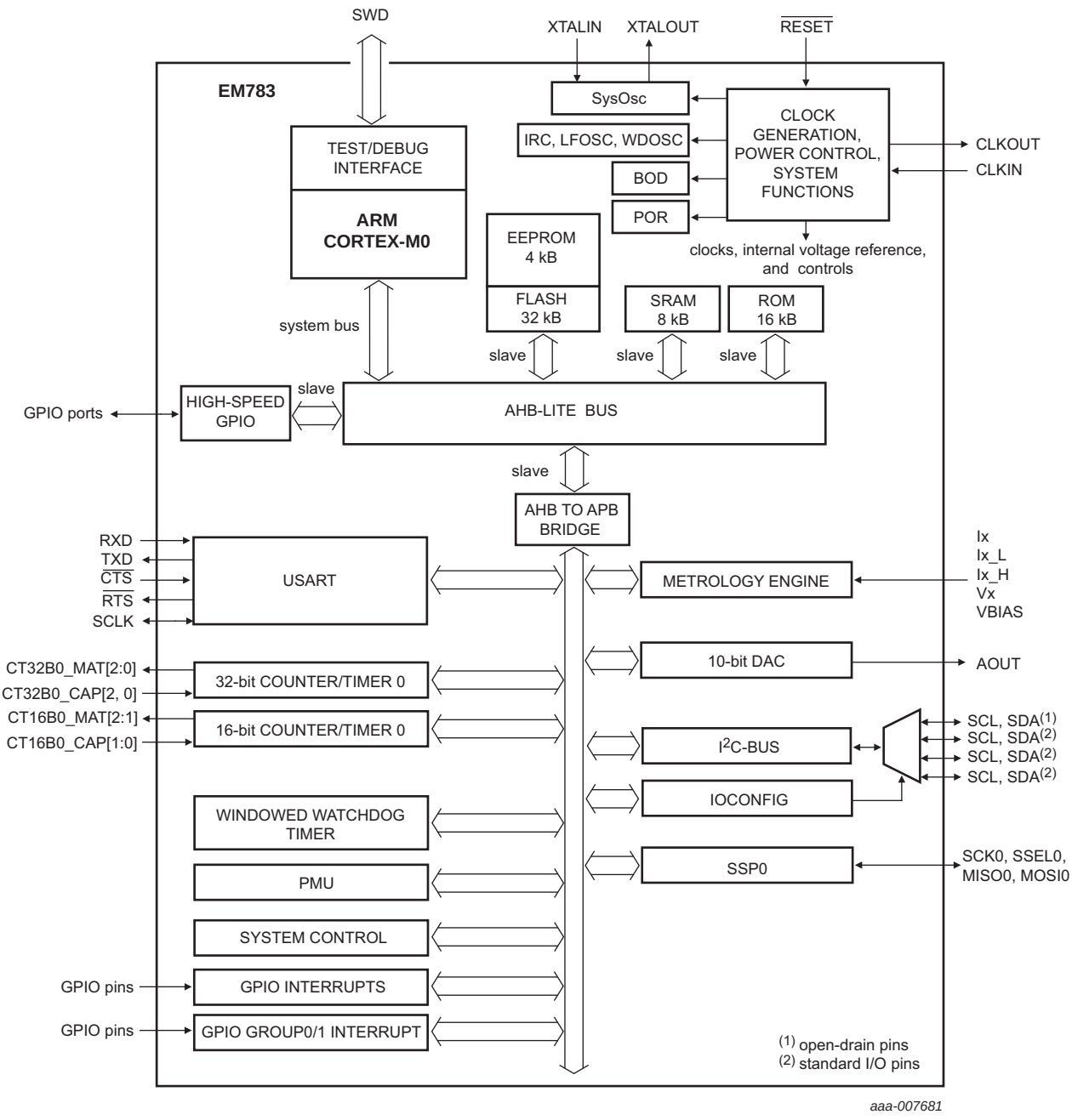


Fig 1. EM783 block diagram

1.6 ARM Cortex-M0 processor

The ARM Cortex-M0 processor is described in detail in Section 23.2 “About the Cortex-M0 processor and core peripherals”. For the EM783, the ARM Cortex-M0 processor core is configured as follows:

- System options
  - Nested Vectored Interrupt Controller (NVIC) supports up to 32 interrupts.
  - System tick timer.
- Debug options
  - Serial Wire Debug with two watchpoints and four breakpoints.

2.1 Memory map

The UM10586 incorporate several distinct memory regions, shown in the following figures. Figure 2 shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address remapping.

The AHB peripheral area is 2 MB in size, and is divided to allow for up to 128 peripherals. The GPIO ports are the only AHB 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.

All APB register addresses are 32-bit word aligned regardless of their size. All accesses to APB registers are treated as word accesses by the peripheral. It is not possible to write individual bytes or halfwords of an APB register separately.

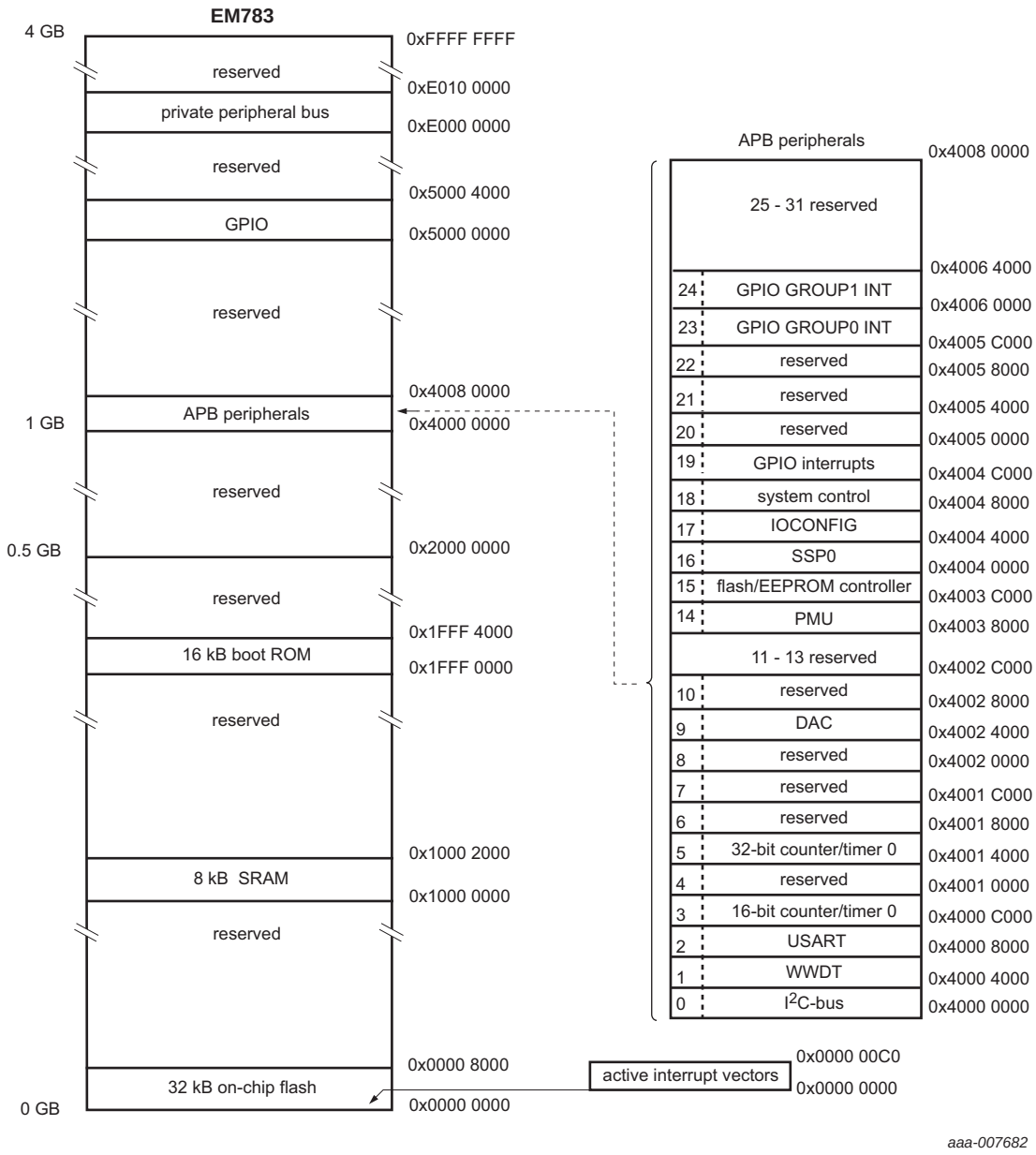


Fig 2. EM783 memory map

3.1 Introduction

The system configuration block controls oscillators and clock generation of the EM783. Also included in this block are registers for setting priority for AHB access and for remapping flash, SRAM, and ROM memory areas.

3.2 Pin description

Table 3 shows pins that are associated with system control block functions.

Table 3. Pin summary

Pin name	Pin direction	Pin description
CLKIN	I	Clock in pin
CLKOUT	O	Clock out pin

3.3 Clocking and Power Control

3.3.1 Clocking

See Figure 3 for an overview of the EM783 Clock Generation Unit (CGU).

The EM783 includes four oscillators: a crystal oscillator (XTAL), a 12 MHz High Frequency oscillator (IRC) and the programmable Low Frequency and Watchdog oscillators (LFOSC, WDOSC). The XTAL, IRC, and LFOSC can be used for multiple purposes in a particular application.

The clock source for the PLL can be the crystal oscillator, the IRC or an external clock on the CLKIN pin, as selected by the SYSPLLCLKSEL register.

After reset, the EM783 will operate from the IRC unless switched by software.

The SYSAHBCLKCTRL register gates the system clock to the various peripherals and memories. The UART and SSP interfaces and the Watchdog and SysTick timers have individual clock dividers to derive peripheral clocks from the main clock.

Any clock can be output directly, or divided by factor between 2 and 255, on the CLKOUT pin.

The term “PCLK” is an abbreviation for “Peripheral Clock” and is used in subsequent chapters to indicate the clock provided to an APB peripheral. Figure 3 shows several signals that may be called PCLK from the peripheral point of view.

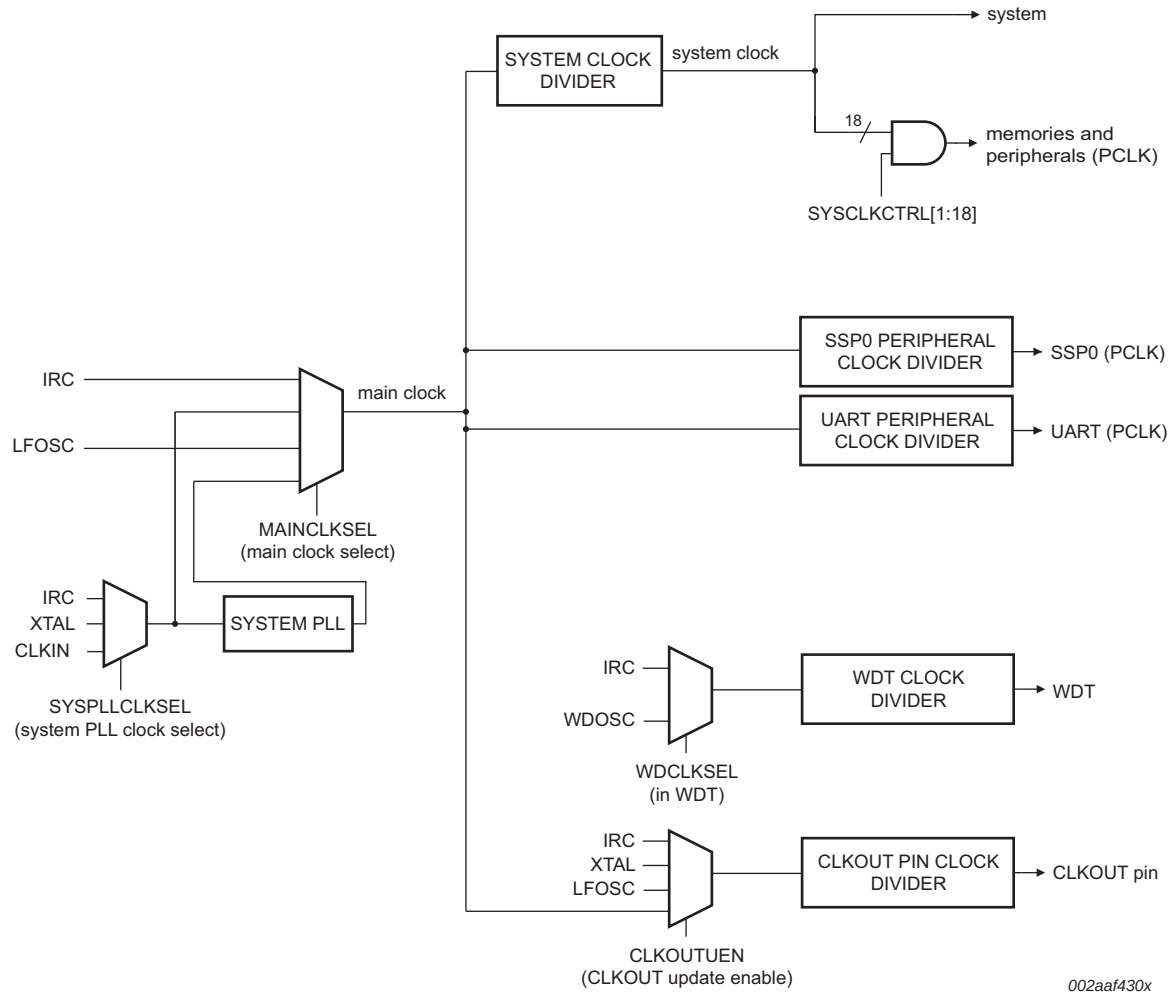


Fig 3. EM783 CGU block diagram

3.3.2 Power Control

The EM783 supports various power control features. Power and clocks to selected blocks of the EM783 can be optimized to minimize power consumption when the chip is running.

3.4 Register description

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

See Section 3.10 for the flash access timing register, which can be reconfigured as part the system setup. This register is not part of the system configuration block.

All address offsets not shown in Table 4 are reserved and should not be written.

Table 4. Register overview: system control block (base address 0x4004 8000)

Name	Access	Offset	Description	Reset value	Reference
SYSMEMREMAP	R/W	0x0	System memory remap	0x02	Table 5
PRESETCTRL	R/W	0x004	Peripheral reset control	0	Table 6
SYSPLLCTRL	R/W	0x008	System PLL control	0	Table 7
SYSPLLSTAT	R	0x00C	System PLL status	0	Table 8
SYSOSCCTRL	R/W	0x020	System oscillator control	0x000	Table 9
WDTOSCCTRL	R/W	0x024	Watchdog oscillator control	0x0A0	Table 10
IRCCTRL	R/W	0x028	IRC oscillator control	0x0xx	Table 11
LFOSCCTRL	R/W	0x02C	LF oscillator control	0x0A0	Table 12
SYSRSTSTAT	R/W	0x030	System reset status register	0	Table 13
SYSPLLCLKSEL	R/W	0x040	System PLL clock source select	0	Table 14
SYSPLLCLKUEN	R/W	0x044	System PLL clock source update enable	0	Table 15
MAINCLKSEL	R/W	0x070	Main clock source select	0	Table 16
MAINCLKUEN	R/W	0x074	Main clock source update enable	0	Table 17
SYSAHBCLKDIV	R/W	0x078	System clock divider	0x001	Table 18
SYSAHBCLKCTRL	R/W	0x080	System clock control	0x01F	Table 19
SSP0CLKDIV	R/W	0x094	SSP0 clock divider	0	Table 20
UARTCLKDIV	R/W	0x098	UART clock divider	0	Table 21
-	-	0x09C to 0x0DC	Reserved	-	-
CLKOUTSEL	R/W	0x0E0	CLKOUT clock source select	0	Table 22
CLKOUTUEN	R/W	0x0E4	CLKOUT clock source update enable	0	Table 23
CLKOUTDIV	R/W	0x0E8	CLKOUT clock divider	0	Table 24
PIOPORCAP0	R	0x100	POR captured PIO status 0	user dependent	Table 25
-	-	0x104	Reserved	-	-
BODR	R/W	0x150	BrownOut Detect	0	Table 26
SYSTCKCAL	R/W	0x158	System tick counter calibration	0x004	Table 27
NMISRC	R/W	0x174	NMI Source Control	0	Table 28
PINTSEL0	R/W	0x178	GPIO Pin Interrupt Select register 0	0	Table 29
PINTSEL1	R/W	0x17C	GPIO Pin Interrupt Select register 1	0	Table 29
PINTSEL2	R/W	0x180	GPIO Pin Interrupt Select register 2	0	Table 29
PINTSEL3	R/W	0x184	GPIO Pin Interrupt Select register 3	0	Table 29
PINTSEL4	R/W	0x188	GPIO Pin Interrupt Select register 4	0	Table 29
PINTSEL5	R/W	0x18C	GPIO Pin Interrupt Select register 5	0	Table 29
PINTSEL6	R/W	0x190	GPIO Pin Interrupt Select register 6	0	Table 29
PINTSEL7	R/W	0x194	GPIO Pin Interrupt Select register 7	0	Table 29
-	R/W	0x198 - 0x234	Reserved	-	-
PDRUNCFG	R/W	0x238	Power configuration register	0x0000 EDF0	Table 30
DEVICE_ID	R	0x3F4	Device ID	0x45A0 002B	Table 31
DEVICE_CFG	R	0x3FC	Device Variant	part dependent	Table 32

3.4.1 System memory remap register

The system memory remap register selects whether the exception vectors are read from the boot ROM, the flash, or the 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.

Table 5. System memory remap register (SYSMEMREMAP, address 0x4004 8000) bit description

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

3.4.2 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.

Table 6. Peripheral reset control register (PRESETCTRL, address 0x4004 8004) bit description

Bit	Symbol	Value	Description	Reset value
0	SSP0_RST_N		SSP0 reset control	1
		0	Resets the SSP0 peripheral.	
		1	SSP0 reset de-asserted.	
1	I2C_RST_N		I2C reset control	1
		0	Resets the I2C peripheral.	
		1	I2C reset de-asserted.	
3:2	-		Reserved	-
4	UART_RST_N		UART reset control	1
		0	Resets the UART peripheral.	
		1	UART reset de-asserted.	
5	CT16B0_RST_N		CT16B0 reset control	1
		0	Resets the CT16B0 peripheral.	
		1	CT16B0 reset de-asserted.	
6	-		Reserved	-

Table 6. Peripheral reset control register (PRESETCTRL, address 0x4004 8004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
7	CT32B0_RST_N		CT32B0 reset control	1
		0	Resets the CT32B0 peripheral.	
		1	CT32B0 reset de-asserted.	
9:8	-		Reserved	-
10	DAC_RST_N		DAC reset control	1
		0	Resets the DAC peripheral.	
		1	DAC reset de-asserted.	
31:11	-	-	Reserved	-

3.4.3 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 up to a high frequency, 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.

The PLL operating mode is set by the DIRECT and BYPASS bits (see Table 34).

Table 7. 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.	-

3.4.4 System PLL status register

This register is a Read-only register and supplies the PLL lock status (see Section 3.9.1).

Table 8. 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	-

3.4.5 System oscillator control register

This register configures the frequency range for the system oscillator.

Table 9. 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 pin bypassing the oscillator. Use this mode when using an external clock source instead of the crystal oscillator.	
1	FREQRANGE		Determines frequency range for Low-power oscillator	0x0
		0	1 to 20 MHz frequency range.	
		1	15 to 25 MHz frequency range.	
31:2	-	-	Reserved	0x00

3.4.6 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 0.6 MHz and 4.6 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
xxx_osc_clk = Fclkana/(2 × (1 + DIVSEL)).

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

Table 10. 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

Table 10. Watchdog oscillator control register (WDTOSCCTRL, address 0x4004 8024) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
8:5	FREQSEL		Select watchdog oscillator analog output frequency (Fclkana).	0
		0	Operation is undefined for this value. Startup code should program a non-zero value in this field as soon after reset as possible.	
		0x1	0.6 MHz	
		0x2	1.05 MHz	
		0x3	1.4 MHz	
		0x4	1.75 MHz	
		0x5	2.1 MHz	
		0x6	2.4 MHz	
		0x7	2.7 MHz	
		0x8	3.0 MHz	
		0x9	3.25 MHz	
		0xA	3.5 MHz	
		0xB	3.75 MHz	
		0xC	4.0 MHz	
		0xD	4.2 MHz	
		0xE	4.4 MHz	
		0xF	4.6 MHz	
31:9	-	-	Reserved	-

3.4.7 High-frequency oscillator control register

This register is used to trim the on-chip 12 MHz oscillator. The trim value is factory-preset and written by the boot code on start-up.

Table 11. High-frequency oscillator control register (IRCCTRL, address 0x4004 8028) bit description

Bit	Symbol	Description	Reset value
7:0	TRIM	Trim value	programmed during manufacturing
31:9	-	Reserved	-

3.4.8 LF oscillator control register

This register configures the LF oscillator. The LF 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 0.6 MHz and 4.6 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 of the LF oscillator can be calculated as xxx_osc_clk = Fclkana/(2 × (1 + DIVSEL).

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

Table 12. LF oscillator control register (LFOSCCTRL, address 0x4004 802C)) 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).	0
		0	Operation is undefined for this value. Startup code should program a non-zero value in this field as soon after reset as possible.	
		0x1	0.6 MHz	
		0x2	1.05 MHz	
		0x3	1.4 MHz	
		0x4	1.75 MHz	
		0x5	2.1 MHz	
		0x6	2.4 MHz	
		0x7	2.7 MHz	
		0x8	3.0 MHz	
		0x9	3.25 MHz	
		0xA	3.5 MHz	
		0xB	3.75 MHz	
		0xC	4.0 MHz	
		0xD	4.2 MHz	
		0xE	4.4 MHz	
		0xF	4.6 MHz	
31:9	-	-	Reserved	-

3.4.9 System reset status register

If another reset signal - for example the external 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 13 applies to the POR reset.

Table 13. 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	

Table 13. System reset status register (SYSRSTSTAT, address 0x4004 8030) bit description
…continued

Bit	Symbol	Value	Description	Reset value
1	EXTRST		External reset status	0
		0	No $\overline{\text{RESET}}$ event detected	
		1	$\overline{\text{RESET}}$ detected	
2	WDT		Status of the Watchdog reset	0
		0	No WDT reset detected	
		1	WDT reset detected	
3	BOD		Status of the Brownout detect reset	0
		0	No BOD reset detected	
		1	BOD reset detected	
4	SYSRST		Status of the software system reset	0
		0	No System reset detected	
		1	System reset detected	
31:5	-	-	Reserved	-

3.4.10 System PLL clock source select register

This register selects the clock source for the system PLL. Toggle the SYSPLLCLKUEN register (see Section 3.4.11) from LOW to HIGH for the update to take effect.

Table 14. 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 (XTAL)	
		0x2	CLKIN pin	
		0x3	Reserved	
31:2	-	-	Reserved	-

3.4.11 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 15. 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	-

3.4.12 Main clock source select register

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

Bit 0 of the MAINCLKUEN register (see Section 3.4.13) must be toggled from 0 to 1 for the update to take effect.

Table 16. 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	LF Oscillator	
		0x3	PLL output	
31:2	-	-	Reserved	-

3.4.13 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 17. 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	-

3.4.14 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 18. 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	-

3.4.15 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 19. 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/EEPROM register interface.	1
		0	Disabled	
		1	Enabled	
4	FLASHARRAY		Enables clock for flash/EEPROM array access.	1
		0	Disabled	
		1	Enabled	
5	I2C		Enables clock for I2C.	1
		0	Disable	
		1	Enable	
6	GPIO		Enables clock for GPIO.	0
		0	Disable	
		1	Enable	
7	CT16B0		Enables clock for 16-bit counter/timer 0.	0
		0	Disable	
		1	Enable	
8	-	-	Reserved	-
9	CT32B0		Enables clock for 32-bit counter/timer 0.	0
		0	Disable	
		1	Enable	
10	-	-	Reserved	-
11	SSP0		Enables clock for SSP0.	0
		0	Disable	
		1	Enable	

Table 19. System clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
12	UART		Enables clock for UART. The UART pins must be configured in the IOCON block before the UART clock can be enabled.	0
		0	Disable	
		1	Enable	
14:13	-		Reserved	-
15	WDT		Enables clock for WDT.	0
		0	Disable	
		1	Enable	
16	IOCON		Enables clock for I/O configuration block.	0
		0	Disable	
		1	Enable	
18:17	-	-	Reserved	-
19	PINT		GPIO Pin interrupts	0
		0	Disable	
		1	Enable	
20	-	-	Reserved	-
21	DAC		Enables clock for DAC.	0
		0	Disable	
		1	Enable	
22	-		Reserved	-
23	P0INT		GPIO Port 0 interrupt	0
		0	Disable	
		1	Enable	
31:24	-	-	Reserved	-

3.4.16 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 20. SSP0 clock divider register (SSP0CLKDIV, address 0x4004 8094) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	SSP0_PCLK clock divider values 0: Disable SSP0_PCLK. 1: Divide by 1. to 255: Divide by 255.	0
31:8	-	Reserved	-

3.4.17 UART clock divider register

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

Remark: The UART pins must be configured in the IOCON block before the UART clock can be enabled.

Table 21. UART 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	-

3.4.18 CLKOUT clock source select register

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

Toggle bit 0 of the CLKOUTUEN register (see Section 3.4.19) from 0 to 1 for the update to take effect.

Table 22. 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 (XTAL)	
		0x2	LF oscillator	
		0x3	Main clock	
31:2	-	-	Reserved	0

3.4.19 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 23. 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	-

3.4.20 CLKOUT clock divider register

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

Table 24. 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	-

3.4.21 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 25. 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	NC by other resets

3.4.22 BrownOut Detect register

The BrownOut Detect circuit monitors the voltage on the VDD supply. If it is below a certain threshold, the circuit requests an interrupt. The BOD register selects the thresholds for interrupt. The thresholds shown in Table 26 are typical values (see *EM783 data sheet*).

The BOD interrupt can be enabled in the Section 23.5.2.2 “Interrupt Set-enable Register” and disabled in the Section 23.5.2.3 “Interrupt Clear-enable Register”, using the bit number corresponding to the IRQ number for the BOD shown in Table 36.

Table 26. BrownOut Detect register (BODR, address 0x4004 8150) bit description

Bit	Symbol	Value	Description	Reset value
1:0	-		Reserved	0
3:2	BODINTVAL		BOD interrupt threshold	0x2
		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	-		Reserved	0
5	-		Reserved	0
6	BODINT		This bit is 1 if the BOD is requesting an interrupt.	0
31:7	-	-	Reserved	-

3.4.23 System tick counter calibration register

Table 27. System tick timer calibration register (SYSTCKCAL, address 0x4004 8158) bit description

Bit	Symbol	Description	Reset value
25:0	CAL	System tick timer calibration value	0x4
31:26	-	Reserved	-

3.4.24 NMI source selection register

Table 28. 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 is to act as the Non-Maskable interrupt (NMI) if bit 31 is 1. See Section 4.3 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.

3.4.25 Pin interrupt select registers

Each of these 8 registers selects one GPIO pin from all GPIO pins as the source of a pin interrupt.

Each of the 8-pin interrupts must be enabled in the NVIC using interrupt slots # 0 to 7 (see Table 36).

To enable each pin interrupt and configure its edge or level sensitivity, use the GPIO pin interrupt registers (Table 37).

Table 29. Pin interrupt select registers (PINTSEL0 to 7, address 0x4004 8178 to 0x4004 8194) bit description

Bit	Symbol	Description	Reset value
4:0	INTPIN	The pin number in Port 0.	0
31:5	-	Reserved	-

3.4.26 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 takes effect immediately except for 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 30. Power configuration register (PDRUNCFG, address 0x4004 8238) bit description

Bit	Symbol	Value	Description	Reset value
0	IRCOUT_PD		IRC oscillator output power-down	0
		1	Powered down	
		0	Powered	
1	IRC_PD		IRC oscillator power-down	0
		1	Powered down	
		0	Powered	
2	FLASH_PD		Flash power-down	0
		1	Powered down	
		0	Powered	
3	-		Reserved	1
4	-		Reserved. **This bit must be set to zero during normal operation in Run mode.**	1
5	XTAL_PD		Crystal oscillator power-down	1
		1	Powered down	
		0	Powered	
6	WDTOSC_PD		Watchdog oscillator power-down	1
		1	Powered down	
		0	Powered	
7	SYSPLL_PD		System PLL power-down	1
		1	Powered down	
		0	Powered	
8	-		Reserved	1
9	-		Reserved. **This bit must be set to zero during normal operation in Run mode.**	0
10	-		Reserved	1
11	-		Reserved. This bit must be set to one in Run mode.	1
12	-		Reserved.	0
13	LFOSC_PD		Low frequency oscillator power-down	1
		1	Powered down	
		0	Powered	
14	DAC_PD		DAC power-down	1
		1	Powered down	
		0	Powered	

Table 30. Power configuration register (PDRUNCFG, address 0x4004 8238) bit description

Bit	Symbol	Value	Description	Reset value
15	TS_PD		Temperature Sensor power-down	1
		1	Powered down	
		0	Powered	
16	-	-	Reserved. **This bit must be set to zero during normal operation in Run mode.**	1
31:17	-	-	Reserved	-

3.4.27 Device ID register

This device ID register is a read-only register and contains the device ID for EM783 series. This register is also read by the ISP/IAP commands (see Section 18.6 and Section 18.7).

Table 31. Device ID register (DEVICE_ID, address 0x4004 83F4) bit description

Bit	Symbol	Description	Reset value
31:0	DEVICEID	Part ID number for EM783 series	0x45A0 002B

3.4.28 Device Configuration register

This device configuration register is a read-only register and contains the part ID for each EM783 variant.

Table 32. Device configuration register (DEVICECFG, address 0x4004 83FC) bit description

Bit	Symbol	Description	Reset value
31:0	DEVICECFG	Part ID for each EM783 variant	
		EM783-SC	0x9000 5A00
		EM783-SP	0xA000 5A00
		EM783-MC3	0xB000 5A00
		EM783-MC6	0xC000 5A00
		EM783-TP	0xE000 5A00

3.5 Reset

Reset has several sources on the EM783: the $\overline{\text{RESET}}$ pin, Watchdog Reset, Power-On Reset (POR), BrownOut Detect (BOD) and an internal reset timer. In addition, there is a software reset. The software reset is triggered by writing a 1 to the ARM Cortex-M0 SYSRESETREQ bit (see Table 290).

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.

3.5.1 Under-Voltage Lockout (UVLO) reset behavior

The UVLO in EM783 uses an always-on reset circuit to keep the part in a safe reset state during brownout, black-out, and cold start-up situations. The UVLO circuit comprises of the BOD and POR circuits which hold the part in reset for voltage levels

below operating voltage. The POR holds the part in reset below the POR trip point. The always-on BOD reset trigger holds the part in reset for supply voltages between the BOD trip point and the POR trip point. The UVLO circuit is active on both ends of the power cycle: when powering down and powering back up.

Once the BOD has released the BOD reset signal, an internal reset timer releases the internal reset after an additional 100 μs, and the code begins to execute.

To allow the analog BOD and POR circuits to settle in a shallow or deep brownout situation, power drop fluctuations should not be shorter than 5 μs for shallow brownouts and 12 μs for deep brownouts. Refer also to *NXP Semiconductors data sheet EM783*.

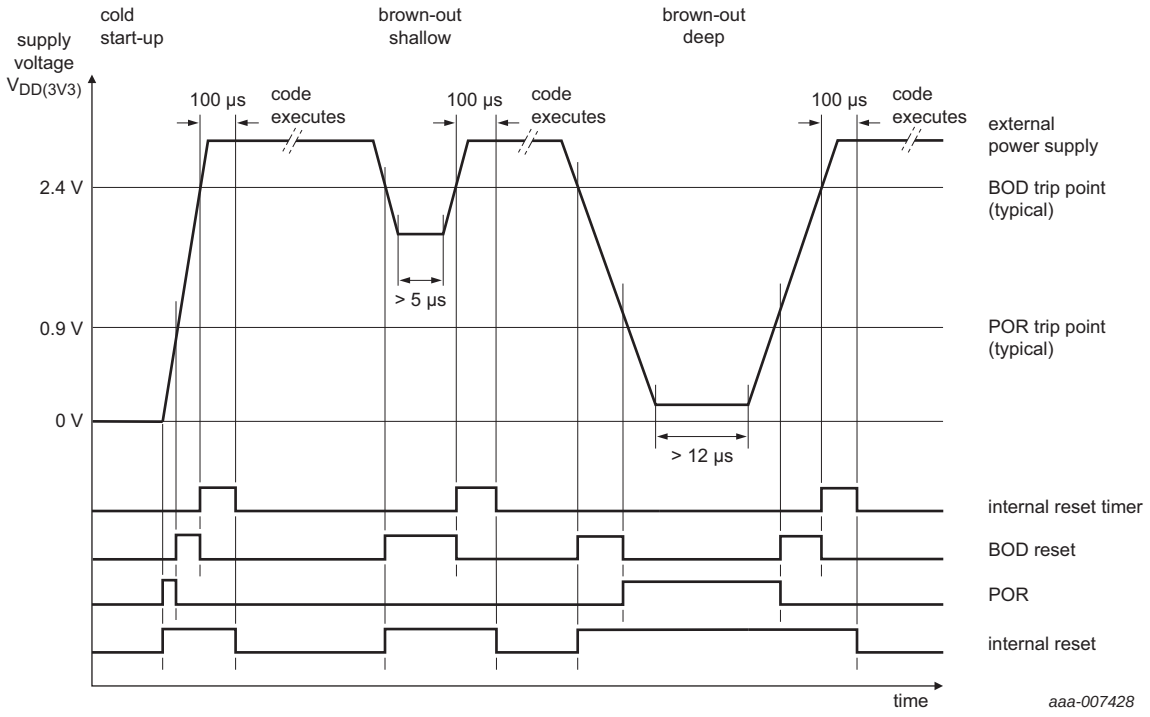


Fig 4. UVLO reset behaviour

3.6 Brownout detection

The EM783 includes a brownout detect circuit that monitors the voltage on the V_{DD} pin. If this voltage falls below the selected level, the BOD asserts an interrupt signal to the NVIC. This signal can be enabled for interrupt in the Interrupt Enable Register in the NVIC; if not, software can monitor the signal by reading a dedicated status register. A fixed hardware-enabled threshold assures that the chip is reset if V_{DD} falls below the threshold; see Table 26.

3.7 Start-up timing

This 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. The internal reset timer starts counting when the supply voltage reaches the BOD threshold voltage. Once the internal

reset timer has finished counting, the internal reset is released.

If the external reset is not used, then the boot loader will execute immediately after the internal reset timer has expired. This is shown in Figure 5.

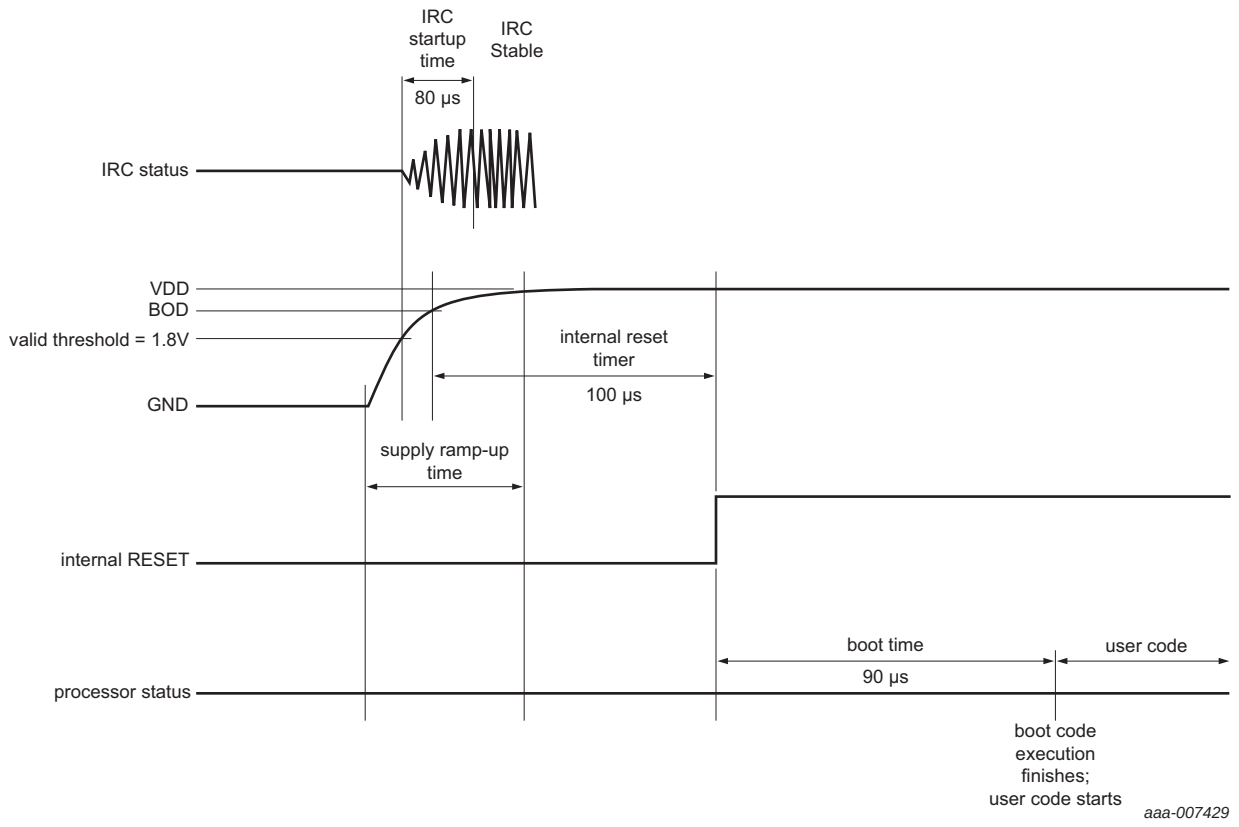


Fig 5. Typical start-up time with respect to the internal reset

If the external reset is used, the boot loader will execute only after the internal reset timer has expired and the external reset is deasserted. This is shown in Figure 6.

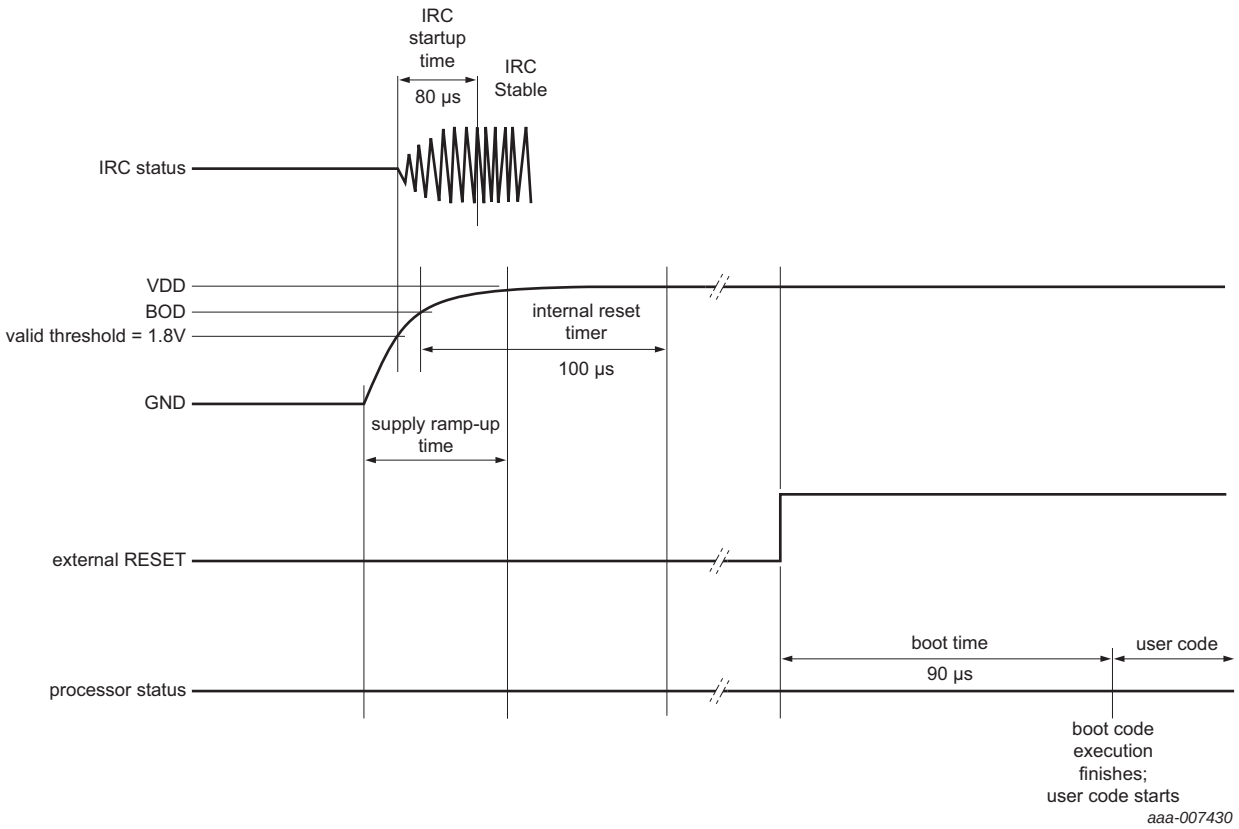


Fig 6. Typical start-up time with respect to the external reset

3.8 Power management

Section 3.3.2 describes the reduced-power modes of the EM783.

The CPU clock rate can be controlled as needed by changing clock sources, reconfiguring PLL values, and/or altering the system clock divider value. This allows a trade-off of power versus processing speed based on application requirements.

Run-time power control allows shutting down the clocks to individual on-chip peripherals, allowing fine-tuning of power consumption by eliminating all dynamic power use in any peripherals not required for the application. Selected peripherals (UART, SysTick timer, Watchdog timer) have their own clock divider for power control.

Remark: The debug mode is not supported in any of the reduced power modes.

Table 33. EM783 power and clock control registers

Register		Power/clock control function	Applies to modes
Power control			
PDRUNCFG	Table 30	Controls power to the analog blocks (oscillators, PLLs, DAC, flash, BOD and metrology engine). **Remark:** Bits 9 and 12 of this register must be zero for proper operation during run mode.	Run
Clock control			
SYSAHBCLKCTRL	Table 19	Controls clocks to the ARM Cortex-M0 CPU, memories, and individual APB peripherals.	Run
SYSAHBCLKDIV	Table 18	Disables or configures the system clock.	Run
UARTCLKDIV	Table 21	Disables or configures the UART peripheral clock.	Run
CLKOUTDIV	Table 24	Disables or configures the clock on the CLKOUT pin.	Run

3.8.1 Run mode

In Run mode, the ARM Cortex-M0 core, memories, and peripherals are clocked by the system clock. The SYSAHBCLKCTRL register controls which memories and peripherals are running. The system clock frequency can be selected by the SYSAHBCLKDIV register.

Selected peripherals (UART, WDT, and the Systick timer) have individual peripheral clocks with their own clock dividers in addition to the system clock. The peripheral clocks can be shut down through the respective clock divider registers.

The power to various blocks (PLLs, oscillators, the ADC, DAC, analog comparator, BOD circuit, and the flash block) can be controlled individually through the PDRUNCFG register.

Remark: Ensure that bit 9 of the PDRUNCFG register is zero in Run mode.

3.8.2 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.

Sleep mode is entered when software performs the following steps:

1. Ensure that the SLEEPDEEP bit is 0 in the Cortex-M0 SCR register (Table 291).
2. Execute the Wait-For-Interrupt (WFI) instruction.

Sleep mode is exited automatically when an interrupt arrives at the processor.

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 System PLL functional description

The EM783 uses the system PLL to create the clocks for the core and peripherals.

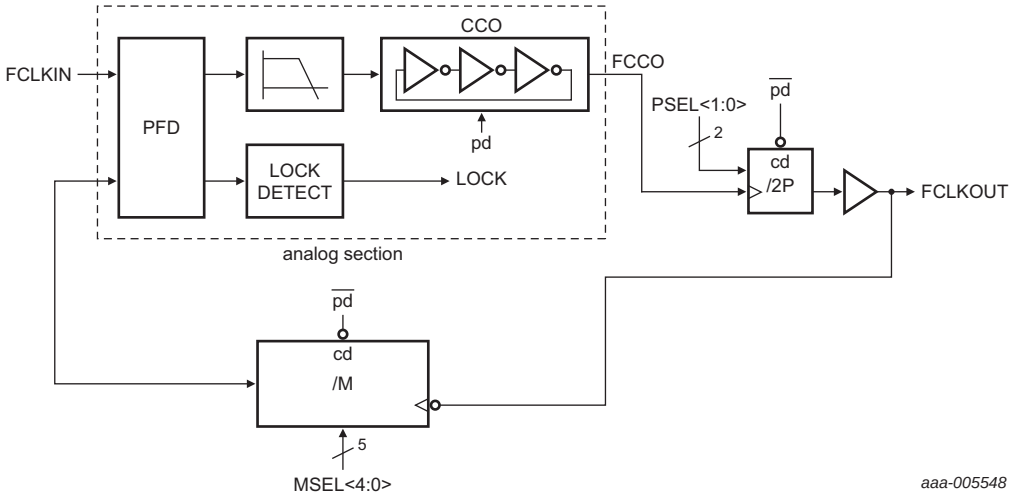


Fig 7. System PLL block diagram

The block diagram of the 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.The CCO output is divided by 2×P by the programmable post divider to create the output clock(s). 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.

3.9.1 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 aligned. This effectively prevents false lock indications, and thus ensures a glitch free lock signal.

3.9.2 Direct output mode

In normal operating mode, the CCO clock is divided by 2, 4, 8 or 16 depending on the PSEL field, giving an output clock with a 50 % duty cycle.

3.9.3 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 SYS_PLL_PD bits to one in the Power-down configuration register (Table 30). In this mode, the internal current reference is turned off, the oscillator and the phase-frequency detector are stopped and the dividers enter a reset state. While in Power-down mode, the lock output is low to indicate that the PLL is not in lock. When the Power-down mode is terminated by setting the SYS_PLL_PD bits to zero, the PLL resumes normal operation and makes the lock signal high once it has regained lock on the input clock.

3.9.4 Operating modes

Table 34. PLL operating modes

Mode	Description	PD bit	BYPASS bit	DIRECT bit
1	Normal mode	0	0	0
3	Power-down mode	1	0	x

3.9.5 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 7. This guarantees an output clock with a 50 % duty cycle.

Feedback divider

The MSEL bits control the division ratio of the feedback divider. The division ratio between the PLLs output clock and the input clock is the decimal value on MSEL bits plus one, as specified in Table 7.

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 reads 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.

3.9.6 Frequency selection

The PLL frequency equations use the following parameters (also see Figure 3):

Table 35. PLL frequency parameters

Parameter	System PLL
FCLKIN	Frequency of sys_pllclkin (input clock to the PLL), from XTAL, IRC, or CLKIN
FCCO	Frequency of the Current Controlled Oscillator (CCO); 156 MHz to 320 MHz.
FCLKOUT	Frequency of sys_pllclkout
P	System PLL post divider ratio; PSEL bits in SYSPLLCTRL (see Section 3.4.3).
M	System PLL feedback divider register; MSEL bits in SYSPLLCTRL (see Section 3.4.3).

3.9.6.1 Mode 1 (Normal mode)

This mode gives a 50 % duty cycle clock with the following frequency relations:

$$Fclkout = M \times Fclkin = \frac{FCCO}{2 \times P}$$

Select integers M and P as follows:

1. Given that F_{clkin} is 12 MHz from the IRC, select M in the range 1 to 32 to produce a desired output frequency

   $$Fclkout = M \times 12MHz$$

2. Verify that for one of the values P = 1, 2, 4, or 8,

   $$FCCO = Fclkout \times 2 \times P$$

   is in the range 156 MHz to 320 MHz.

3. If the PLL output is selected in the Main clock source select register (Table 16), verify that F_{clkout} divided by the value in the System clock divider register (Table 18) is less than or equal to the maximum processor clock frequency (Section 1.2 or the EM783 data sheet).
4. Program the System PLL Control register (Table 7) with MSEL = M-1 and PSEL = P-1.

3.9.6.2 Mode 3 (Power-down mode)

In this mode, the internal current reference is turned off, the oscillator and the phase-frequency detector are stopped and the dividers enter a reset state. While in Power-down mode, the lock output is low, to indicate that the PLL is not in lock. When the Power-down mode is terminated by making pd low, the PLL resumes normal operation, and makes the lock signal high once it has regained lock on the input clock.

3.10 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 Section 18.9.1).

Remark: Improper setting of this register may result in incorrect operation of the EM783 flash memory.

4.1 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.

4.2 Features

- Integral part of the ARM Cortex-M0
- Tightly coupled interrupt controller provides low interrupt latency
- Controls system exceptions and peripheral interrupts
- In the EM783, the NVIC supports 24 vectored interrupts
- Four programmable interrupt priority levels with hardware priority level masking
- Relocatable vector table
- Software interrupt generation

4.3 Interrupt sources

Table 36 lists the interrupt sources in the EM783. Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller. Each line may represent more than one interrupt source.

Table 36. Connection of interrupt sources to the Vectored Interrupt Controller

Exception number	IRQ number	Vector address	Source	Flag(s)
16-23	0-7	0x40-0x5C	GPIO	Pin interrupts 0-7
24	8	0x60-64	GPIO	Port interrupts 0
26	10	0x68	ME	Metrology Engine interrupt 0[1]
27	11	0x6C	DAC	Event triggered
30-28	14-12	-	-	Reserved
31	15	0x7C	I^2C	SI (state change)
32	16	0x80	CT16B0	Match 0-3 Capture 0-3
33	17	0x84	ME	Metrology Engine interrupt 1[2]
34	18	0x88	CT32B0	Match 0-3 Capture 0-3
35	19	0x8C	-	Reserved
36	20	0x90	SSP0	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun

Table 36. Connection of interrupt sources to the Vectored Interrupt Controller *…continued*

Exception number	IRQ number	Vector address	Source	Flag(s)
37	21	0x94	USART	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)
38-39	22-23	0x98-0x9C	-	Reserved
40	24	0xA0	ME	Metrology Engine interrupt 2[2]
41	25	0xA4	WDT	Watchdog time-out
42	26	0xA8	BOD	Brownout detect
43	27	0xAC	Flash interface	
44-47	28-31	0xB0-0xBC	-	Reserved

[1] The interrupt priority for IRQ10 must be set to 0x00.

[2] The interrupt priority for IRQ17 and IRQ24 must be set to 0x40.

The IRQ numbers in Table 36 correspond to bit numbers in the following registers that are described in the Cortex-M0 Appendix:

- Section 23.5.2.2 “Interrupt Set-enable Register”
- Section 23.5.2.3 “Interrupt Clear-enable Register”
- Section 23.5.2.4 “Interrupt Set-pending Register”
- Section 23.5.2.5 “Interrupt Clear-pending Register”

The IRQ numbers also map to the fields in Section 23.5.2.6 “Interrupt Priority Registers”.

5.1 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 pins in the SYSCON block (Table 29) and enable the clock to the pin interrupt register block in the SYSAHBCLKCTRL register (Table 19, bit 19).
- For the group interrupt feature, enable the clock to the GROUP0 and GROUP1 register interfaces in the SYSAHBCLKCTRL register ((Table 19, bit 19).
- For the GPIO port registers, enable the clock to the GPIO port register in the SYSAHBCLKCTRL register (Table 19, bit 6).

5.2 Features

5.2.1 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.

5.2.2 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 mode.

5.2.3 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.

5.3 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.

5.3.1 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 29). 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.

5.3.2 GPIO group interrupt

For each 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 is level or edge triggered and whether it is 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 generates an interrupt. If the part is in a power-savings mode it will first asynchronously wake the part up before asserting the interrupt request. The interrupt request line can be cleared by writing a one to the interrupt status bit in the control register.

5.3.3 GPIO port

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

5.4 Register description

The GPIO consists of the following blocks:

- The GPIO pin interrupts block at address 0x4004 C000. Registers in this block enable up to 8 pin interrupts selected in the syscon block PINTSEL registers (see Table 29) and configure the level and edge sensitivity for each selected pin interrupt. The GPIO interrupt registers are listed in Table 37 and Section 5.4.1
- The GPIO GROUP0 interrupt block at address 0x4005 C000. Registers in this block allow configuration of any pin on port 0 to contribute to a combined interrupt. The GPIO GROUP0 registers are listed in Table 38 and Section 5.4.2.
- The GPIO GROUP1 interrupt block at address 0x4005 8000. Registers in this block allow to configure any pin on port 0 to contribute to a combined interrupt. The GPIO GROUP1 registers are listed in Table 39 and Section 5.4.2.
- The GPIO port block at address 0x5000 0000. Registers in this block allow read and write to port pins and configure port pins as inputs or outputs.The GPIO port registers are listed in Table 40 and Section 5.4.3.

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

Table 37. 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 41
IENR	R/W	0x004	Pin interrupt level (rising edge) interrupt enable register	0	Table 42
SIENR	WO	0x008	Pin interrupt level (rising edge) interrupt set register	NA	Table 43
CIENR	WO	0x00C	Pin interrupt level (rising edge interrupt) clear register	NA	Table 44
IENF	R/W	0x010	Pin interrupt active level (falling edge) interrupt enable register	0	Table 45
SIENF	WO	0x014	Pin interrupt active level (falling edge) interrupt set register	NA	Table 46
CIENF	WO	0x018	Pin interrupt active level (falling edge) interrupt clear register	NA	Table 47
RISE	R/W	0x01C	Pin interrupt rising edge register	0	Table 48
FALL	R/W	0x020	Pin interrupt falling edge register	0	Table 49
IST	R/W	0x024	Pin interrupt status register	0	Table 50

Table 38. 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 51
PORT_POL0	R/W	0x020	GPIO grouped interrupt port 0 polarity register	0xFFFF FFFF	Table 52
PORT_ENA0	R/W	0x040	GPIO grouped interrupt port 0 enable register	0	Table 53

Table 39. 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 51
PORT_POL0	R/W	0x020	GPIO grouped interrupt port 0 polarity register	0xFFFF FFFF	Table 52
PORT_ENA0	R/W	0x040	GPIO grouped interrupt port 0 enable register	0	Table 53

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

Table 40. Register overview: GPIO port (base address: 0x5000 0000)

Name	Access	Address offset	Description	Reset value	Width	Reference
B0 to B26	R/W	0x0000 to 0x001A	Byte pin registers port 0; pins P0_0 to P0_26	ext[1]	byte (8 bit)	Table 54
W0 to W26	R/W	0x1000 to 0x1068	Word pin registers port 0	ext[1]	word (32 bit)	Table 55
DIR0	R/W	0x2000	Direction registers port 0	0	word (32 bit)	Table 56
MASK0	R/W	0x2080	Mask register port 0	0	word (32 bit)	Table 57
PIN0	R/W	0x2100	Port pin register port 0	ext[1]	word (32 bit)	Table 58
MPIN0	R/W	0x2180	Masked port register port 0	ext[1]	word (32 bit)	Table 59
SET0	R/W	0x2200	Write: Set register for port 0 Read: output bits for port 0	0	word (32 bit)	Table 60
CLR0	WO	0x2280	Clear port 0	NA	word (32 bit)	Table 61
NOT0	WO	0x2300	Toggle port 0	NA	word (32 bit)	Table 62

[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.

5.4.1 GPIO pin interrupts register description

5.4.1.1 Pin interrupt mode register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), one bit in the ISEL register determines whether the interrupt is edge or level sensitive.

Table 41. 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.	-	-

5.4.1.2 Pin interrupt level (rising edge) interrupt enable register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), 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 IENF register configures the active level (HIGH or LOW) for this interrupt.

Table 42. 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.	-	-

5.4.1.3 Pin interrupt level (rising edge) interrupt set register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), one bit in the SIENR register sets 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 set.
- If the pin interrupt mode is level sensitive (PMODE = 1), the level interrupt is set.

Table 43. 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 IENR, thus enabling interrupts. Bit n sets bit n in the IENR register. 0 = No operation. 1 = Enable rising edge or level interrupt.	NA	WO
31:8	-	Reserved.	-	-

5.4.1.4 Pin interrupt level (rising edge interrupt) clear register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), 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 44. Pin interrupt level (rising edge interrupt) clear register (CIENR, 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.	-	-

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

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), one bit in the IENF register enables the falling edge interrupt or 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 45. 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.	-	-

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

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), one bit in the SIENF 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 46. 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.	-	-

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

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 29), 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 47. 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 clear 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.	-	-

5.4.1.8 Pin interrupt rising edge register

This register contains ones for pin interrupts selected in the PINTSELn registers (see Table 29) 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 PINTSELn registers, regardless of whether they are interrupt-enabled.

Table 48. 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.	-	-

5.4.1.9 Pin interrupt falling edge register

This register contains ones for pin interrupts selected in the PINTSELn registers (see Table 29) 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 PINTSELn registers, regardless of whether they are interrupt-enabled.

Table 49. 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.	-	-

5.4.1.10 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 50. 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 IENF register).	0	R/W
31:8	-	Reserved.	-	-

5.4.2 GPIO GROUP0/GROUP1 interrupt register description

5.4.2.1 Grouped interrupt control register

Table 51. 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

5.4.2.2 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.

Table 52. 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	-

5.4.2.3 GPIO grouped interrupt port enable registers

The grouped interrupt port enable registers enable the pins which contribute to the grouped interrupt.

Table 53. 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 i nterrupt. 1 = the port 0 pin is enabled and contributes to the grouped interrupt.	0	-

5.4.3 GPIO port register description

5.4.3.1 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. It 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 54. GPIO port 0 byte pin registers (B0 to B26, addresses 0x5000 0000 to 0x5000 001A) 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 output bit of the pin.	ext	R/W
7:1		Reserved (0 on read, ignored on write)	0	-

5.4.3.2 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 is all zeros if the pin is low or all ones if the pin is high, regardless of direction, masking, or alternate function. Pins configured as analog I/O always read as zeros. If the value written is all zeros, any write clears the output bit of the pin. Otherwise it sets the output bit of the pin.

Table 55. GPIO port 0 word pin registers (W0 to W31, addresses 0x5000 1000 to 0x5000 1068) 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 sets the output bit.	ext	R/W

5.4.3.3 GPIO port direction registers

The GPIO port direction register is used for configuring the port pins as inputs or outputs.

Table 56. 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

5.4.3.4 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 57. 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

5.4.3.5 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 58. 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

5.4.3.6 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. Writing to one of these registers only affects output register bits that are enabled by zeros in the corresponding MASK register

Table 59. 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

5.4.3.7 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 60. 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

5.4.3.8 GPIO port clear registers

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

Table 61. 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

5.4.3.9 GPIO port toggle registers

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

Table 62. 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

5.5 Functional description

5.5.1 Reading pin state

Software can read the state of all GPIO pins except those pins 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 the port can be read as a byte, halfword, or word from a PORT register.
- The state of a selected subset of the pins in the port can be read from a Masked Port (MPORT) register. Pins having a 1 in the port’s Mask register read as 0 from its MPORT register.

5.5.2 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. Select the pin for GPIO operation in the “I/O Configuration” block.
2. Select the pin for output by a 1 in the 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 PORT register loads the output bits of all the pins written to.
- Writing to MPORT register loads the output bits of pins identified by zeros in corresponding positions of the port’s MASK register.
- Writing ones to the port’s SET register sets output bits.
- Writing ones to the port’s CLR register clears output bits.
- Writing ones to the port’s NOT register toggles/complements/inverts output bits.

The state of the port’s output bits can be read from its SET register. Reading any of the registers described in Section 5.5.1 returns the state of pins, regardless of their direction or alternate functions.

5.5.3 Masked I/O

The 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 the port's MASK register contains all zeros, its PORT and MPORT registers operate identically for reading and writing.

Users of previous NXP Semiconductors devices with similar GPIO blocks should be aware of an incompatibility: on the EM783, 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 perform Masked GPIO operation, must treat the 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.

5.5.4 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 the port can be selected to contribute to a common interrupt.

5.5.4.1 Pin interrupts

In this interrupt facility, up to 8 pins are identified as interrupt sources by the Pin Interrupt Select registers (PINTSEL0-7). All registers in the pin interrupt block contain 8 bits, corresponding to the pins called out by the PINTSEL0-7 registers. The ISEL register defines whether each interrupt pin is edge- or level-sensitive. The RISE and FALL registers detect edges on each interrupt pin, and can be written to clear (and set) edge detection. The IST register indicates whether each interrupt pin is currently requesting an interrupt, and this register can also be written to clear interrupts.

The other pin interrupt registers play different roles for edge-sensitive and level-sensitive pins, as described in Table 63.

Table 63. Pin interrupt registers for edge- and level-sensitive pins

Name	Edge-sensitive function	Level-sensitive function
IENR	Enables rising-edge interrupts.	Enables level interrupts.
SIENR	Write to enable rising-edge interrupts.	Write to enable level interrupts.
CIENR	Write to disable rising-edge interrupts.	Write to disable level interrupts.

Table 63. Pin interrupt registers for edge- and level-sensitive pins *...continued*

Name	Edge-sensitive function	Level-sensitive function
IENF	Enables falling-edge interrupts.	Selects active level.
SIENF	Write to enable falling-edge interrupts.	Write to select high-active.
CIENF	Write to disable falling-edge interrupts.	Write to select low-active.

5.5.4.2 Group interrupts

In this interrupt facility, an interrupt can be requested by selecting any subset of pins within the port. The pins that contribute to the 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. 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 Table 36).

5.5.5 Recommended practices

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

- For initial setup after Reset or reinitialization, 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 action 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.

6.1 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
- hysteresis
- analog/digital mode for pads with analog functions
- open-drain output option

6.2 General description

The IOCON registers control the GPIO or peripheral function, the input mode, and the hysteresis of all pins with a GPIO function. In addition, pins that have an I^2C function can be configured for different I^2C-bus modes. If a pin supports an analog function, the analog mode can be selected.

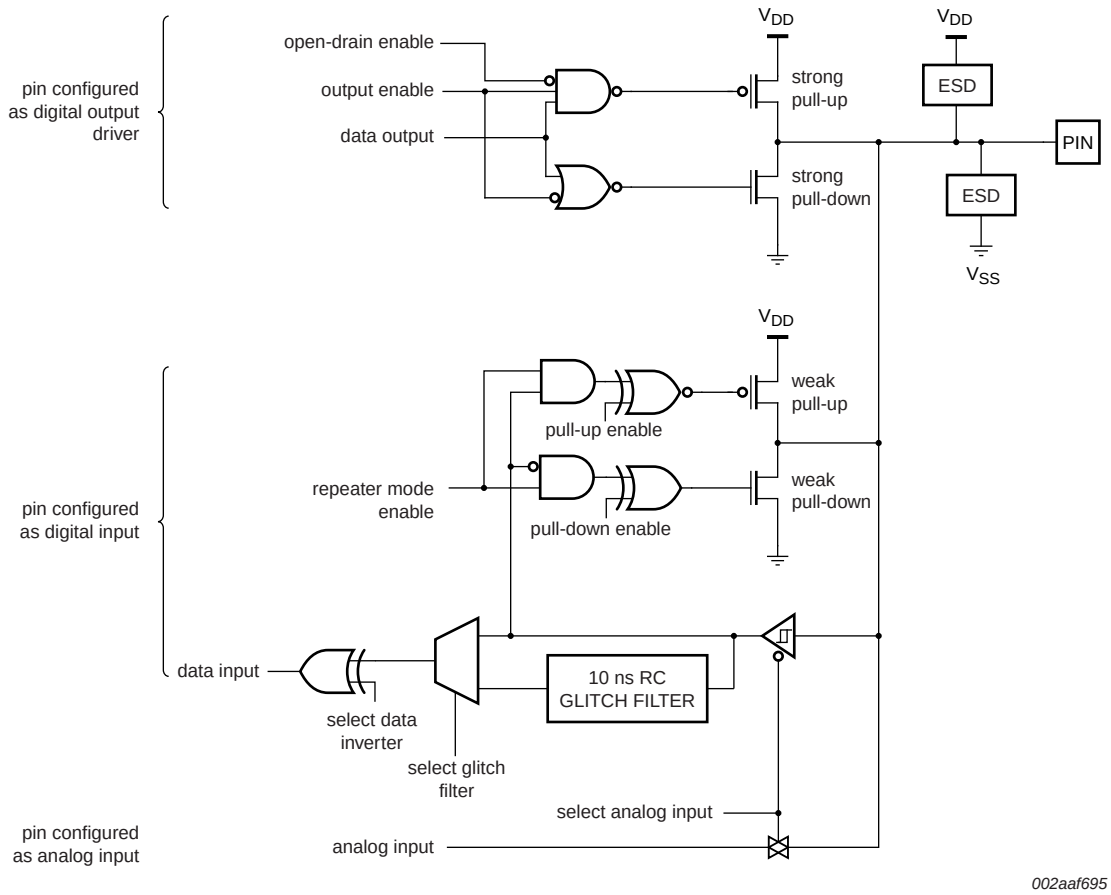


Fig 8. Standard I/O pin configuration

6.2.1 Pin function

The FUNC bits in the IOCON registers can be set to GPIO (typically 000) or to a special function. For pins set to GPIO, the DIR registers determine whether the pin is configured as an input or output (see Table 56). For any special function, the pin direction is controlled automatically depending on the function. The DIR registers have no effect for special functions.

6.2.2 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.

If the pin is high the repeater mode enables the pull-up resistor, and if the pin is low it enables the pull-down resistor. This action causes the pin to retain its last known state if it is configured as an input and is not driven externally. 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.

6.2.3 Hysteresis

The input buffer for digital functions can be configured with or without hysteresis.

6.2.4 Input inversion

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

6.2.5 Analog/digital mode

In Analog mode, the digital receiver is disconnected to eliminate its effect on analog functionality. If Analog mode is selected, the MODE field should be “Inactive” (00); the HYS, INV, FILTR, SLEW, and OD settings have no effect.

For an unconnected pin that has an analog function, keep the ADMODE bit set to “digital” and the MODE field nonzero.

6.2.6 I^2C mode

The HS and HIDRIVE bits in registers P0_2 and P0_3 (Table 67) are provided because these pins are the primary I^2C connections on the device. These options can also be used for other applications.

- For Standard mode or Fast-mode I^2C operation, clear the HS bit so that the input glitch filter is enabled.
- For Fast-mode Plus I^2C operation, clear HS to enable the input glitch filter and set the HIDRIVE bit to select 20 mA sink current.
- In non-I^2C operation, these pins remain open-drain and can only drive to V_{SS}, regardless of how HS and HIDRIVE are set. Leave HS 1 and HIDRIVE 0 to maximize compatibility with other pins. Clearing HS enables a glitch filter that suppresses pulses up to 50 nS in width.

6.2.7 Output slew rate

Set the SLEW bits of digital outputs that do not need to switch state very quickly to “slow”. This setting allows more outputs to switch simultaneously without noticeably degrading the power/ground distribution of the device, and has only a small effect on signal transition time. This action is important if analog accuracy is significant to the application.

6.2.8 Open-Drain Mode

When output is selected, either by selecting a special function in the FUNC field, or by selecting GPIO function for a pin having a 1 in its PnDIR register, a 1 in the OD bit selects open-drain operation. That is, a 1 disables the high-drive transistor. This option has no effect on the primary I^2C pins.

6.2.9 $\overline{\text{RESET}}$ (pin RESET/P0_0)

See Figure 9 for the reset pad configuration. The reset pin includes a fixed 20 ns glitch filter.

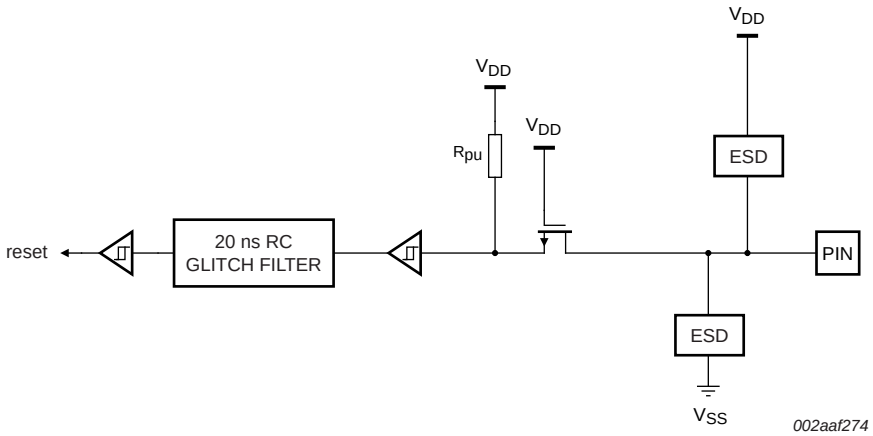


Fig 9. Reset pad configuration

6.3 Register description

The I/O configuration registers control the GPIO port pins and the inputs and outputs of all peripherals and functional blocks.

Each GPIO pin has one IOCON register which controls the function and electrical characteristics of the pin.

The IOCON registers are listed in order of their port numbers and their addresses in tables 64, 66, 68, and 70.

Remark: The reset value in Table 64 refers to the register value after the boot loader has executed.

Table 64. Register overview: I/O configuration (base address 0x4004 4000)

Name	Access	Address offset	Description	Type	Reset value	Reference
RESET_P0_0	R/W	0x000	I/O configuration for pin $\overline{\text{RESET}}$/P0_0	D	0x090	Tables 65, 66
P0_1	R/W	0x004	I/O configuration for pin P0_1/RXD/CLKOUT/ CT32B0_MAT2/SSEL0/CLKIN	D	0x090	Tables 65, 66
P0_2	R/W	0x008	I/O configuration for pin P0_2/SCL/SWCLK/ CT16B0_CAP0	I	0x180	Tables 67, 68
P0_3	R/W	0x00C	I/O configuration for pin P0_3/SDA/SWDIO	I	0x180	Tables 67, 68
P0_4	R/W	0x010	I/O configuration for pin P0_4/AOUT/CT16B0_MAT1/MOSI0	A	0x190	Tables 69, 70
SWCLK_P0_5	R/W	0x014	I/O configuration for pin SWCLK/P0_5/ CT16B0_MAT2/SCK0	A	0x190	Tables 69, 70
		0x018-0x020	Reserved			

Table 64. Register overview: I/O configuration (base address 0x4004 4000) *...continued*

Name	Access	Address offset	Description	Type	Reset value	Reference
P0_9	R/W	0x024	I/O configuration for pin R/P0_9/CT16B0_MAT1/$\overline{\text{CTS}}$	A	0x190	Tables 69, 70
SWDIO_P0_10	R/W	0x028	I/O configuration for pin SWDIO/P0_10/ CT16B0_MAT2/$\overline{\text{RTS}}$	A	0x190	Tables 69, 70
P0_11	R/W	0x02C	I/O configuration for pin P0_11/SCLK/CT32B0_CAP0	A	0x190	Tables 69, 70
P0_12	R/W	0x030	I/O configuration for pin P0_12/RXD/ CT32B0_MAT0/SCL/CLKIN	D	0x090	Tables 65, 66
P0_13	R/W	0x034	I/O configuration for pin P0_13/TXD/ CT32B0_MAT1/SDA	A	0x190	Tables 69, 70
		0x038	Reserved			
P0_15	R/W	0x03C	I/O configuration for pin P0_15/TXD/CT32B0_CAP2/SDA	A	0x190	Tables 69, 70
P0_16	R/W	0x040	I/O configuration for pin P0_16/ATRG0/CT16B0_CAP1/SCL	A	0x190	Tables 69, 70
		0x044	Reserved			
P0_18	R/W	0x048	I/O configuration for pin P0_18/SSEL0/ CT16B0_CAP0	D	0x090	Tables 65, 66
P0_19	R/W	0x04C	I/O configuration for pin P0_19/CLKIN/CLKOUT/ MOSI0	D	0x090	Tables 65, 66
P0_20	R/W	0x050	I/O configuration for pin P0_20/SCK0	D	0x090	Tables 65, 66
P0_21	R/W	0x054	I/O configuration for pin P0_21/$\overline{\text{CTS}}$/SCLK	D	0x090	Tables 65, 66
P0_22	R/W	0x058	I/O configuration for pin P0_22/MISO0	A	0x190	Tables 69, 70
P0_23	R/W	0x05C	I/O configuration for pin P0_23/$\overline{\text{RTS}}$/ CT32B0_CAP0/SCLK	D	0x090	Tables 65, 66
P0_24	R/W	0x060	I/O configuration for pin P0_24/SCL/CLKIN	D	0x090	Tables 65, 66
P0_25	R/W	0x064	I/O configuration for pin P0_25/SDA	D	0x090	Tables 65, 66
P0_26	R/W	0x068	I/O configuration for pin P0_26/TXD/CT32B0_CAP2	D	0x090	Tables 65, 66

6.3.1 I/O configuration registers

EM783 IOCON registers are of 3 types, called D, I, and A. For details on the I/O configuration settings, see Section 6.2.

6.3.1.1 Type D IOCON registers

Table 65. Type D IOCON registers bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See Table 66 for specific values.	000

Table 65. Type D IOCON registers bit description *...continued*

Bit	Symbol	Value	Description	Reset value
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		00	Inactive (no pull-down/pull-up resistor enabled).	
		01	Pull-down resistor enabled.	
		10	Pull-up resistor enabled.	
		11	Repeater mode.	
5	HYS		Hysteresis.	0[1]
		0	Disable.	
		1	Enable.	
6	INV		Input polarity	0
		0	Non-inverted (HIGH on pin = 1)	
		1	Inverted (HIGH on pin = 0)	
8:7	-	-	Reserved.	01
9	SLEW		Driver slew rate	0
		0	Slow (more outputs can be switched simultaneously)	
		1	Fast	
10	OD		Controls open-drain mode	0
		0	Totem-pole output	
		1	Open-drain output (high drive disabled)	
31:11	-	-	Reserved.	0

[1] Except RESET/P0_0, for which HYS resets to 1 (enable hysteresis)

Table 66. Type D IOCON registers: FUNC values and pin functions

Shaded areas in this table indicate Reserved values, which should not be programmed.

	Value of FUNC field							
Register	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
RESET_P0_0	$\overline{\text{RESET}}$	P0_0						
P0_1	P0_1	RXD	CLKOUT	CT32B0_MAT2	SSEL0	CLKIN		
P0_12	P0_12	RXD		CT32B0_MAT0	SCL	CLKIN		
P0_18	P0_18		SSEL0	CT16B0_CAP0				
P0_19	P0_19	CLKIN	CLKOUT	MOSI0				
P0_20	P0_20		SCK0					
P0_21	P0_21	$\overline{\text{CTS}}$			SCLK			
P0_23	P0_23	$\overline{\text{RTS}}$		CT32B0_CAP0	SCLK			
P0_24	P0_24	SCL	CLKIN					
P0_25	P0_25	SDA						
P0_26	P0_26	TXD			CT32B0_CAP2			

6.3.1.2 Type I IOCON registers

Table 67. Type I IOCON registers bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See Table 68 for specific values. The pins are true open-drain for all pin functions.	0
7:3		-	Reserved.	10000
8	HS		Disables I^2C features for faster operation.	1
		0	I^2C glitch filter and slew rate enabled.	
		1	I^2C glitch filter and slew rate disabled.	
9	HIDRIVE	0	Sink current is standard 4 mA.	0
		1	Sink current is 20 mA.	
31:10	-	-	Reserved.	-

Table 68. Type I IOCON registers: FUNC values and pin functions

Shaded areas in this table indicate Reserved values, which should not be programmed.

	Value of FUNC field							
Register	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
P0_2	P0_2	SCL		SWCLK	CT16B0_CAP0			
P0_3	P0_3	SDA		SWDIO				

6.3.1.3 Type A IOCON registers

Table 69. Type A IOCON registers bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. See Table 70 for specific values.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	10
		00	Inactive (no pull-down/pull-up resistor enabled).	
		01	Pull-down resistor enabled.	
		10	Pull-up resistor enabled.	
		11	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Input polarity	0
		0	Non-inverted (HIGH on pin = 1)	
		1	Inverted (HIGH on pin = 0)	
7	ADMODE		Select Analog/Digital mode.	1
		0	Analog mode.	
		1	Digital mode.	
8	FILTR		Controls glitch filter	1
		0	Noise pulses are filtered	
		1	No filtering is done	

Table 69. Type A IOCON registers bit description *...continued*

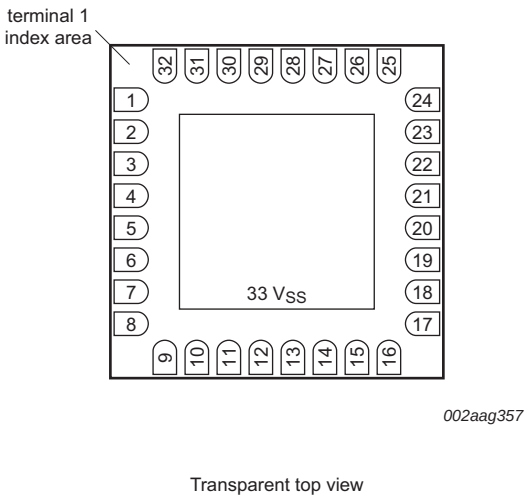
Bit	Symbol	Value	Description	Reset value
9	SLEW		Driver slew rate	0
		0	Slow (more outputs can be switched simultaneously)	
		1	Fast	
10	OD		Controls open-drain mode	0
		0	Totem-pole output	
		1	Open-drain output (high drive disabled)	
31:11	-	-	Reserved.	0

Table 70. Type A IOCON registers: FUNC values and pin functions

Shaded areas in this table indicate Reserved values, which should not be programmed.

	Value of FUNC field							
Register	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
P0_4	P0_4		AOUT	CT16B0_MAT1	MOSI0			
TCK_P0_5	TCK/SWCLK	P0_5		CT16B0_MAT2	SCK0			
P0_9		P0_9			CT16B0_MAT1	$\overline{\text{CTS}}$		
SWDIO_P0_10	SWDIO	P0_10			CT16B0_MAT2	$\overline{\text{RTS}}$		
P0_11	P0_11	SCLK			CT32B0_CAP0			
P0_13	P0_13	TXD		CT32B0_MAT1	SDA			
P0_15	P0_15	TXD		CT32B0_CAP2	SDA			
P0_16	P0_16	ATRG0		CT16B0_CAP1	SCL			
P0_22	P0_22	MISO0						

7.1 Pin configuration



Transparent top view

Fig 10. Pin configuration HVQFN 33 package

7.2 Pin description

All functional pins except the metrology inputs on the EM783 are mapped to GPIO port 0 (see Table 71). The port pins are multiplexed to accommodate more than one function (see Table 72).

The pin function is controlled by the pin's IOCONFIG register. The standard I/O pad configuration is illustrated in Figure 8 and a detailed pin description is given in Table 72.

Table 71. Pin multiplexing

Function	Type	Port	Glitch filter	Pin
System clocks, reset, and wake-up				
CLKIN	I	P0_1	no	3
		P0_12	no	31
		P0_19	no	9
		P0_24	no	7
CLKOUT	O	P0_1	no	3
		P0_19	no	9
XTALIN	I (analog)	-	-	4
XTALOUT	O (analog)	-	-	5

Table 71. Pin multiplexing *…continued*

Function	Type	Port	Glitch filter	Pin
$\overline{\text{RESET}}$	I	P0_0	20 ns [1]	2
Serial Wire Debug (SWD)				
SWCLK	I	P0_2	50 ns [2]	10
		P0_5	10 ns [2]	19
SWDIO	I/O	P0_3	50 ns [2]	11
		P0_10	10 ns [2]	25
Metrology engine				
V1	I (analog)	-	no	28
VBIAS	I (analog)	-	no	14
V1	I (analog)	-	no	22
VBIAS	I (analog)	-	no	23
I1	I (analog)	-	no	20
I1_L	I (analog)	-	no	20
I1_H	I (analog)	-	no	21
I5	I (analog)	-	no	21
I2_L	I (analog)	-	no	24
I2	I (analog)	-	no	24
I2_H	I (analog)	-	no	25
V2	I (analog)	-	no	25
I6	I (analog)	-	no	25
I3_L	I (analog)	-	no	26
I3	I (analog)	-	no	26
I3_H	I (analog)	-	no	27
I4	I (analog)	-	no	27
V3	I (analog)	-	no	27
Analog peripherals				
AOUT	O (analog)	P0_4	no	18
ATRG0	I	P0_16	10 ns [2]	13
I^2C-bus interface				
SCL	I/O	P0_2	50 ns [2]	10
		P0_12	no	31
		P0_16	10 ns [2]	13
		P0_24	no	7
SDA	I/O	P0_3	50 ns [2]	11
		P0_13	10 ns [2]	32
		P0_15	10 ns [2]	27
		P0_25	no	12
SSP0 controller				
MISO0	I/O	P0_22	10 ns [2]	17

Table 71. Pin multiplexing *…continued*

Function	Type	Port	Glitch filter	Pin
MOSI0	I/O	P0_4	10 ns [2]	18
		P0_19	no	9
SCK0	I/O	P0_5	10 ns [2]	19
		P0_20	no	15
SSEL0	I/O	P0_1	no	3
		P0_18	no	8
USART				
RXD	I	P0_1	no	3
		P0_12	no	31
TXD	O	P0_13	no	32
		P0_15	no	27
		P0_26	no	1
SCLK	I/O	P0_11	10 ns [2]	26
		P0_21	no	16
		P0_23	no	30
$\overline{CTS}$	I	P0_9	10 ns [2]	24
		P0_21	no	16
$\overline{RTS}$	O	P0_10	no	25
		P0_23	no	30
16-bit counter/timer CT16B0				
CT16B0_CAP0	I	P0_2	50 ns [2]	10
		P0_18	no	8
CT16B0_CAP1	I	P0_16	10 ns [2]	13
CT16B0_MAT1	O	P0_4	no	18
		P0_9	no	24
CT16B0_MAT2	O	P0_5	no	19
		P0_10	no	25
32-bit counter/timer CT32B0				
CT32B0_CAP0	I	P0_11	10 ns [2]	26
		P0_23	no	30
CT32B0_CAP2	I	P0_15	10 ns [2]	27
		P0_26	no	1
CT32B0_MAT0	O	P0_12	no	31
CT32B0_MAT1	O	P0_13	no	32
CT32B0_MAT2	O	P0_1	no	3
Supply and ground pins				
$V_{DD(IO)}$	Supply	-	-	6
$V_{DD(3V3)}$	Supply	-	-	29
V_{SS}	Ground	-	-	33
$V_{SS(IO)}$	Ground	-	-	33

[1] Always on.

[2] Programmable on/off. By default, the glitch filter is disabled.

Table 72 shows all pins in order of increasing pin number. The default function after reset is listed first. All port pins P0_0 to P0_26 have internal pull-up resistors enabled after reset with the exception of the true open-drain pins P0_2 and P0_3.

Pull-up/pull-down configuration, repeater, and open-drain modes can be programmed through the IOCONFIG registers for each of the port pins.

Table 72. EM783 pin description

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
1	P0_26/TXD/CT32B0_CAP2					[2]	I/O	I; PU	**P0_26 —** General purpose digital input/output pin.
							O	-	**TXD —** Transmitter data output for USART.
							I	-	**CT32B0_CAP2 —** Capture input 2 for 32-bit timer 0.
2	$\overline{\text{RESET}}$/P0_0					[3]	I	I; PU	**$\overline{\text{RESET}}$ —** External reset input with fixed 20 ns glitch filter: A LOW going pulse 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.
							I/O	-	**P0_0 —** General purpose digital input/output pin.
3	P0_1/RXD/CLKOUT/CT32B0_MAT2/SSEL0/CLKIN					[2]	I/O	I; PU	**P0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler.
							I	-	**RXD —** Receiver data input for USART.
							O	-	**CLKOUT —** Clock output.
							O	-	**CT32B0_MAT2 —** Match output 2 for 32-bit timer 0.
							I/O	-	**SSEL0 —** Slave Select for SSP0.
							I	-	**CLKIN —** External clock input.
4	XTALIN					[4]	-	-	Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V.
5	XTALOUT					[4]	-	-	Output from the oscillator amplifier.
6	$V_{DD(IO)}$					[5] [6]	-	-	3.3 V input/output supply voltage.
7	P0_24/SCL/CLKIN					[2]	I/O	I; PU	**P0_24 —** General purpose digital input/output pin.
							I/O	-	**SCL —** I^2C-bus clock input/output. This is not an I^2C-bus open-drain pin[10].
							I	-	**CLKIN —** External clock input.

Table 72. EM783 pin description *…continued*

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
8	P0_18/SSEL0/CT16B0_CAP0					[2]	I/O	I; PU	**P0_18 —** General purpose digital input/output pin.
							I/O	-	**SSEL0 —** Slave Select for SSP0.
							I	-	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
9	P0_19/CLKIN/CLKOUT/MOSI0					[2]	I/O	I; PU	**P0_19 —** General purpose digital input/output pin.
							I	-	**CLKIN —** External clock input.
							O	-	**CLKOUT —** Clock output.
							I/O	-	**MOSI0 —** Master Out Slave In for SSP0.
10	P0_2/SCL/SWCLK/CT16B0_CAP0					[7]	I/O	I; IA	**P0_2 —** General purpose digital input/output pin. High-current sink (20 mA) or standard-current sink (4 mA) programmable; true open-drain for all pin functions. Input glitch filter (50 ns) capable.
							I/O	-	**SCL —** I^2C-bus clock (true open-drain) input/output with selectable 50 ns input glitch filter. Input glitch filter (50 ns) capable.
							I	-	**SWCLK —** Serial Wire Debug Clock (secondary). Input glitch filter (50 ns) capable.
							I	-	**CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0.
11	P0_3/SDA/SWDIO					[7]	I/O	I; IA	**P0_3 —** General purpose digital input/output pin. High-current sink (20 mA) or standard-current sink (4 mA) programmable; true open-drain for all pin functions. Input glitch filter (50 ns) capable.
							I/O	-	**SDA —** I^2C-bus data (true open-drain) input/output. Input glitch filter (50 ns) capable.
							I/O	-	**SWDIO —** Serial Wire Debug I/O (secondary). Input glitch filter (50 ns) capable. **NOTE**: for SP/TP/MC3/MC6 variants user should keep an option to use pin 11 or pin 25 for SWDIO.
12	P0_25/SDA					[2]	I/O	I; PU	**P0_25 —** General purpose digital input/output pin.
							I/O	-	**SDA —** I^2C-bus data input/output. This is not an I^2C-bus open-drain pin[10].

Table 72. EM783 pin description *…continued*

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
13	P0_16/ATRG0/ CT16B0_CAP1/SCL					[8]	I/O	I; PU	**P0_16 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							I	-	**ATRG0 —** Conversion trigger for DAC. Input glitch filter (10 ns) capable.
							I	-	**CT16B0_CAP1 —** Capture input 1 for 16-bit timer 0. Input glitch filter (10 ns) capable.
							I/O	-	**SCL —** I^2C-bus clock input/output. This is not an I^2C-bus open-drain pin[10]. Input glitch filter (10 ns) capable.
14	VBIAS					[8]	I	-	**VBIAS —** Bias voltage input for metrology engine.
15	P0_20/SCK0					[2]	I/O	I; PU	**P0_20 —** General purpose digital input/output pin.
							I/O	-	**SCK0 —** Serial clock for SSP0.
16	P0_21/$\overline{\text{CTS}}$/SCLK					[2]	I/O	I; PU	**P0_21 —** General purpose digital input/output pin. If configured as output, this pin is a high-current source output driver (20 mA).
							I	-	**$\overline{\text{CTS}}$ —** Clear To Send input for USART.
							I/O	-	**SCLK —** Serial clock for USART.
17	P0_22/MISO0					[2]	I/O	I; PU	**P0_22 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							I/O	-	**MISO0 —** Master In Slave Out for SSP0. Input glitch filter (10 ns) capable.
18	P0_4/AOUT/CT16B0_MAT1/MOSI0					[9]	I/O	I; PU	**P0_4 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**AOUT —** DAC output.
							O	-	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
							I/O	-	**MOSI0 —** Master Out Slave In for SSP0. Input glitch filter (10 ns) capable.
19	SWCLK/P0_5/CT16B0_MAT2/SCK0					[8]	I	I; PU	**SWCLK —** Primary (default) Serial Wire Debug Clock. Input glitch filter (10 ns) capable.
							I/O	-	**P0_5 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
							I/O	-	**SCK0 —** Serial clock for SSP0. Input glitch filter (10 ns) capable.

Table 72. EM783 pin description *…continued*

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
20	I1_L	I1_L	I1	I1_L	I1	[9]	I	-	**I1_L —** Low-gain current input for metrology engine of SC, SP and MC3 variant.
							I	-	**I1 —** Current input for metrology engine of TP and MC6 variant.
21	I1_H	I1_H	R	I1_H	I5	[8]	I	-	**I1_H —** High-gain current input for metrology engine of SC, SP and MC3 variant.
							I	-	**I5 —** Current input for metrology engine of MC6 variant.
							I	I; PU	**R —** Reserved
22	V1					[8]	I	-	**V1 —** Voltage input for metrology engine.
23	VBIAS					[8]	I	-	**VBIAS —** Bias voltage input for metrology engine.
24	R/P0_9/CT16B0_MAT1/$\overline{\text{CTS}}$	I2_L	I2	I2_L	I2	[8]	I	I; PU	**R —** Reserved
							I/O	-	**P0_9 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**CT16B0_MAT1 —** Match output 1 for 16-bit timer 0.
							I	-	**$\overline{\text{CTS}}$ —** Clear To Send input for USART. Input glitch filter (10 ns) capable.
							I	-	**I2_L —** Low-gain current input for metrology engine of SP and MC3 variant.
							I	-	**I2 —** Current input for metrology engine of TP and MC6 variant.
25	SWDIO/P0_10/CT16B0_MAT2/$\overline{\text{RTS}}$	SWDI0/	SWDI0/	SWDI0/	SWDI0/	[8]	I/O	I; PU	**SWDIO —** Primary (default) Serial Wire Debug I/O. Input glitch filter (10 ns) capable. **NOTE**: for SP/TP/MC3/MC6 variants user should keep an option to use pin 25 or pin 11 (P0_3) for SWDIO.
							I/O	-	**P0_10 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**CT16B0_MAT2 —** Match output 2 for 16-bit timer 0.
							O	-	**$\overline{\text{RTS}}$ —** Request To Send output for USART.
							I	-	**I2_H —** High-gain current input for metrology engine of SP and MC3 variant.
							I	-	**I6 —** Current input for metrology engine of MC6 variant.
							I	-	**V2 —** Voltage input for metrology engine of TP variant.

Table 72. EM783 pin description *…continued*

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
26	P0_11/SCLK/CT32B0_CAP0		I3	I3_L	I3	[8]	I/O	I; PU	**P0_11 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							I/O	-	**SCLK —** Serial clock for USART. Input glitch filter (10 ns) capable.
							I	-	**CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0. Input glitch filter (10 ns) capable.
							I	-	**I3_L —** Low-gain current input for metrology engine of MC3 variant.
							I	-	**I3 —** Current input for metrology engine of TP and MC6 variant.
27	P0_15/TXD/CT32B0_CAP2/SDA		V3	I3_H	I4	[8]	I/O	I; PU	**P0_15 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**TXD —** Transmitter data output for USART.
							I	-	**CT32B0_CAP2 —** Capture input 2 for 32-bit timer 0. Input glitch filter (10 ns) capable.
							I/O	-	**SDA —** I^2C-bus data input/output. This is not an I^2C-bus open-drain pin[10]. Input glitch filter (10 ns) capable.
							I	-	**I3_H —** High-gain current input for metrology engine of MC3 variant.
							I	-	**I4 —** Current input for metrology engine of MC6 variant.
							I	-	**V3 —** Voltage input for metrology engine of TP variant.
28	V1					[2]	I	-	**V1 —** Voltage input for metrology engine.
29	$V_{DD(3V3)}$					[5] [6]	-	-	3.3 V supply voltage to the metrology engine, internal regulator, and internal clock generator circuits. Also used as the metrology engine reference voltage.
30	P0_23/$\overline{\text{RTS}}$/CT32B0_CAP0/SCLK					[2]	I/O	I; PU	**P0_23 —** 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 for USART.
31	P0_12/RXD/CT32B0_MAT0/SCL/CLKIN					[2]	I/O	I; PU	**P0_12 —** General purpose digital input/output pin.
							I	-	**RXD —** Receiver data input for USART.
							O	-	**CT32B0_MAT0 —** Match output 0 for 32-bit timer 0.
							I/O	-	**SCL —** I^2C-bus clock input/output. This is not an I^2C-bus open-drain pin [10].
							I	-	**CLKIN —** External clock input.

Table 72. EM783 pin description *…continued*

Pin no.	EM783 symbol						Type	Reset state [1]	Description
	SC	SP	TP	MC3	MC6				
32	P0_13/TXD/CT32B0_MAT1/SDA					[8]	I/O	I; PU	**P0_13 —** General purpose digital input/output pin. Input glitch filter (10 ns) capable.
							O	-	**TXD —** Transmitter data output for USART.
							O	-	**CT32B0_MAT1 —** Match output 1 for 32-bit timer 0.
							I/O	-	**SDA —** I^2C-bus data input/output. This is not an I^2C-bus open-drain pin [10]. Input glitch filter (10 ns) capable.
33	$V_{SS(IO)}/V_{SS}$					[11]	-	-	Ground.

[1] Pin state at reset for default function: I = Input; O = Output; PU = internal pull-up resistor (weak PMOS device) enabled; IA = inactive, no pull-up/down enabled.

[2] 5 V tolerant pin providing standard digital I/O functions with configurable modes and configurable hysteresis (Figure 8).

[3] See Figure 9 for the reset configuration.

[4] 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. Refer to *EM783 data sheet, Section 12.1 "XTAL input"*, if an external clock is connected to the XTALIN pin.

[5] If separate supplies are used for $V_{DD(3V3)}$ and $V_{DD(IO)}$, ensure that the power supply pins are filtered for noise. Using separate filtered supplies reduces the noise to the metrology engine and analog blocks.

[6] If separate supplies are used for $V_{DD(3V3)}$ and $V_{DD(IO)}$, ensure that the voltage difference between both supplies is smaller than or equal to 0.5 V.

[7] 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.

[8] 5 V tolerant pin providing standard digital I/O functions with configurable modes, configurable hysteresis, and analog I/O. When configured as an analog I/O, digital section of the pin is disabled, and the pin is not 5 V tolerant (Figure 8).

[9] Not a 5 V tolerant pin due to special analog functionality. Pin provides standard digital I/O functions with configurable modes, configurable hysteresis, and analog I/O. When configured as an analog I/O, the digital section of the pin is disabled (Figure 8).

[10] I^2C-bus pins are standard digital I/O pins and have limited performance and electrical characteristics compared to the full I^2C-bus specification. Pins can be configured with an on-chip pull-up resistor (pMOS device) and with open-drain mode. In this mode, typical bit rates of up to 100 kbit/s with 20 pF load are supported if the internal pull-ups are enabled. Higher bit rates can be achieved with an external resistor.

[11] Thermal pad. Connect to ground.

8.1 How to read this chapter

The USART block is identical for all EM783 parts.

8.2 Basic configuration

The USART is configured as follows:

- Pins: The USART pins must be configured in the corresponding IOCON registers before the USART clocks are enabled (see Section 6.3).
- The USART block is enabled through the SYSAHBCLKCTRL register (see Table 19).
- The peripheral USART clock (PCLK), which is used by the USART baud rate generator, is controlled by the UARTCLKDIV register (see Table 21).

8.3 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.
- Optional $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ flow control signals.
- Optional 1X-clock send or receive.
- Optional ISO 7816-3 compliant smart card interface.

8.4 Pin description

Table 73. 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.**
SCLK	I/O	**Serial Clock.**

8.5 Register description

The USART contains registers organized as shown in Table 74. The Divisor Latch Access Bit (DLAB) is contained in LCR[7] and enables access to the Divisor Latches.

Offsets/addresses not shown in Table 74 are **Reserved**.

Table 74. Register overview: USART (base address: 0x4000 8000)

Name	Access	Address offset	Description	Reset Value [1]
RBR	RO	0x0	Receiver Buffer Register. Contains the next received character to be read. (DLAB=0)	NA
THR	WO	0x0	Transmit Holding Register. The next character to be transmitted is written here. (DLAB=0)	NA
DLL	R/W	0x0	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
DLM	R/W	0x4	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
IER	R/W	0x4	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential USART interrupts. (DLAB=0)	0
IIR	RO	0x8	Interrupt ID Register. Identifies which interrupt(s) are pending.	0x01
FCR	WO	0x8	FIFO Control Register. Controls USART FIFO usage and modes.	0
LCR	R/W	0xC	Line Control Register. Contains controls for frame formatting and break generation.	0
MCR	R/W	0x10	Modem Control Register.	0
LSR	RO	0x14	Line Status Register. Contains flags for transmit and receive status, including line errors.	0x60
MSR	RO	0x18	Modem Status Register.	0
SCR	R/W	0x1C	Scratch Pad Register. Eight-bit temporary storage for software.	0
ACR	R/W	0x20	Auto-baud Control Register. Contains controls for the auto-baud feature.	0
ICR	R/W	0x24	IrDA Control Register. Enables and configures the IrDA (remote control) mode.	0
FDR	R/W	0x28	Fractional Divider Register. Generates a clock input for the baud rate divider.	0x10
OSR	R/W	0x2C	Oversampling Register. Controls the degree of oversampling during each bit time.	0xF0
TER	R/W	0x30	Transmit Enable Register. Turns off USART transmitter for use with software flow control.	0x80
SCICTRL	R/W	0x48	Smart Card Interface Control register. Enables and configures the Smart Card Interface feature.	0
RS485CTRL	R/W	0x4C	RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes.	0
ADRMATCH	R/W	0x50	RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode.	0
RS485DLY	R/W	0x54	RS-485/EIA-485 direction control delay.	0
SYNCCTRL	R/W	0x58	Synchronous mode control register.	0

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

8.5.1 USART Receiver Buffer Register (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 zeroes.

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 86) correspond to the byte on the top of the RBR FIFO (that is, 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 75. USART Receiver Buffer Register when DLAB = 0 (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	-

8.5.2 USART Transmitter Holding Register (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 76. USART Transmitter Holding Register when DLAB = 0 (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	-

8.5.3 USART Divisor Latch LSB and MSB Registers (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 8.5.16.

Table 77. 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 78. USART Divisor Latch MSB Register (DLM - address 0x4000 8004 when DLAB = 1) 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	-

8.5.4 USART Interrupt Enable Register (DLAB = 0)

The IER is used to enable the various USART interrupt sources.

Table 79. 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.	
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.	

Table 79. USART Interrupt Enable Register when DLAB = 0 (IER - address 0x4000 8004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
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

8.5.5 USART Interrupt Identification Register

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 80. USART Interrupt Identification Register (IIR - address 0x4004 8008, RO) bit description

Bit	Symbol	Value	Description	Reset value
0	INTSTATUS		Interrupt status. 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 as follows 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, 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 81. 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 before 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 an 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) clears 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 81. 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
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	MSR Read

[1] Values “0000”, “0011”, “0101”, “0111”, “1000”, “1001”, “1010”, “1011”, “1101”, “1110”, “1111” are reserved.

[2] For details see Section 8.5.9 “USART Line Status Register (LSR - 0x4000 8014, Read Only)”

[3] For details see Section 8.5.1 “USART Receiver Buffer Register (DLAB = 0, Read Only)”

[4] For details see Section 8.5.5 “USART Interrupt Identification Register” and Section 8.5.2 “USART Transmitter Holding Register (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.

8.5.6 USART FIFO Control Register

The FCR controls the operation of the USART RX and TX FIFOs.

Table 82. USART FIFO Control Register (FCR - address 0x4000 8008, Write Only) 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 automatically clears 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] clears 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] clears 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

Table 82. USART FIFO Control Register (FCR - address 0x4000 8008, Write Only) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
7:6	RXTL		RX Trigger Level. These 2 bits determine how many receiver 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	-

8.5.7 USART Modem Control Register

The MCR enables the modem loopback mode and controls the modem output signals.

Table 83. USART Modem Control Register (MCR - address 0x4000 8010) bit description

Bit	Symbol	Value	Description	Reset value
0	DTRCON		Source for modem output pin $\overline{\text{DTR}}$. This bit reads as 0 when modem loopback mode is active.	0
1	RTSCON		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 4 bits of the MSR are driven by the lower 4 bits of the MCR. This permits modem status interrupts to be generated in loopback mode by writing the lower 4 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
		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	-

8.5.7.1 Auto-flow control

If auto-RTS mode is enabled, the receiver FIFO hardware of the USART controls its $\overline{RTS}$ output. If the auto-CTS mode is enabled, the transmitter of the USART will only start sending if the $\overline{CTS}$ pin is low.

8.5.7.1.1 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 $\overline{RTS}$ is 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 82 on page 73). The $\overline{RTS}$ output is reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

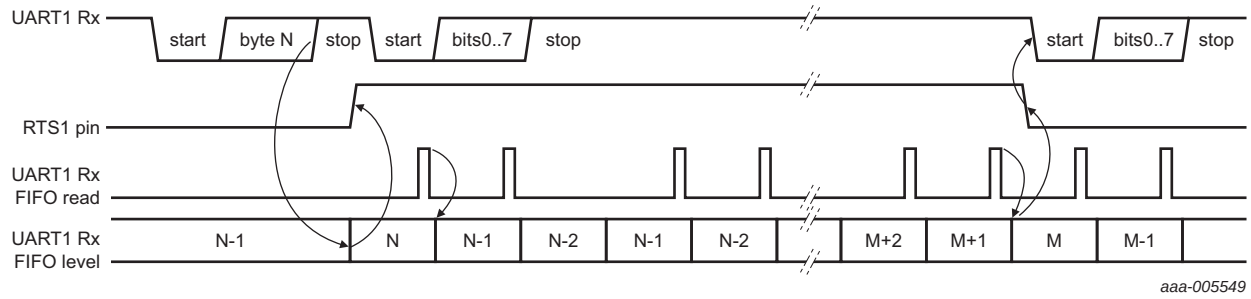


Fig 11. Auto-RTS Functional Timing

8.5.7.1.2 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 84 lists the conditions for generating a Modem Status interrupt.

Table 84. 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 $\overline{\text{CTS}}$ 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 12 illustrates the Auto-CTS functional timing.

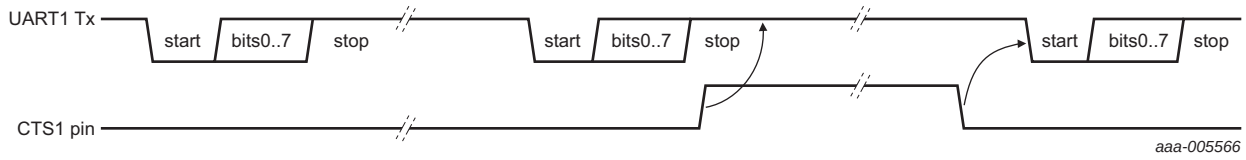


Fig 12. Auto-CTS Functional Timing

During transmission of the second character the $\overline{\text{CTS}}$ signal is negated. The third character is not sent thereafter. The USART maintains 1 on TXD as long as $\overline{\text{CTS}}$ is negated (high). As soon as $\overline{\text{CTS}}$ is asserted, transmission resumes and a start bit is sent followed by the data bits of the next character.

8.5.8 USART Line Control Register

The LCR determines the format of the data character that is to be transmitted or received.

Table 85. 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.	

Table 85. USART Line Control Register (LCR - address 0x4000 800C) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
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
		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	-

8.5.9 USART Line Status Register (LSR - 0x4000 8014, Read Only)

The LSR is a Read Only register that provides status information on the USART TX and RX blocks.

Table 86. USART Line Status Register (LSR - address 0x4000 8014, Read Only) 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. An 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.	

Table 86. USART Line Status Register (LSR - address 0x4000 8014, Read Only) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
2	PE		Parity Error. When the parity bit of a received character is in the wrong state, a parity error occurs. An 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. An 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 resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. **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). An 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.	
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.	

Table 86. USART Line Status Register (LSR - address 0x4000 8014, Read Only) bit description *...continued*

Bit	Symbol	Value	Description	Reset Value
7	RXFE		Error in RX FIFO. LSR[7] is set when a character with an 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	-

8.5.10 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 87: 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.	
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

Table 87: USART Modem Status Register (MSR - address 0x4000 8018) bit description
…continued

Bit	Symbol	Value	Description	Reset value
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

8.5.11 USART Scratch Pad Register (SCR - 0x4000 801C)

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 88. 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	-

8.5.12 USART Auto-baud Control Register (ACR - 0x4000 8020)

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 89. 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	
		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

Table 89. Auto-baud Control Register (ACR - address 0x4000 8020) bit description
…continued

Bit	Symbol	Value	Description	Reset value
8	ABEOINTCLR		End of auto-baud interrupt clear bit (write only accessible).	0
		0	Writing a 0 has no impact.	
		1	Writing a 1 clears 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 clears 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

8.5.13 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 measures the bit time of the receive data stream and sets the divisor latch registers DLM and DLL so.

Auto-baud is started by setting the ACR Start bit. Auto-baud can be stopped by clearing the ACR Start bit. The Start bit clears once auto-baud has finished and reading the bit returns 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).

If a time-out occurs (the rate measurement counter overflows), the ACR AutoRestart bit can be used to automatically restart baud rate measurement. If this bit is set, the rate measurement restarts at the next falling edge of the USART Rx pin.

The auto-baud function can generate two interrupts.

- The IIR ABTOInt interrupt is set if the interrupt is enabled (IER ABToIntEn is set and the auto-baud rate measurement counter overflows).
- The IIR ABEOInt interrupt is set if the interrupt is enabled (IER ABEOIntEn is set and the auto-baud has completed successfully).

Clear the auto-baud interrupts 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 UART_PCLK and the number of data bits, stop bits and parity bits.

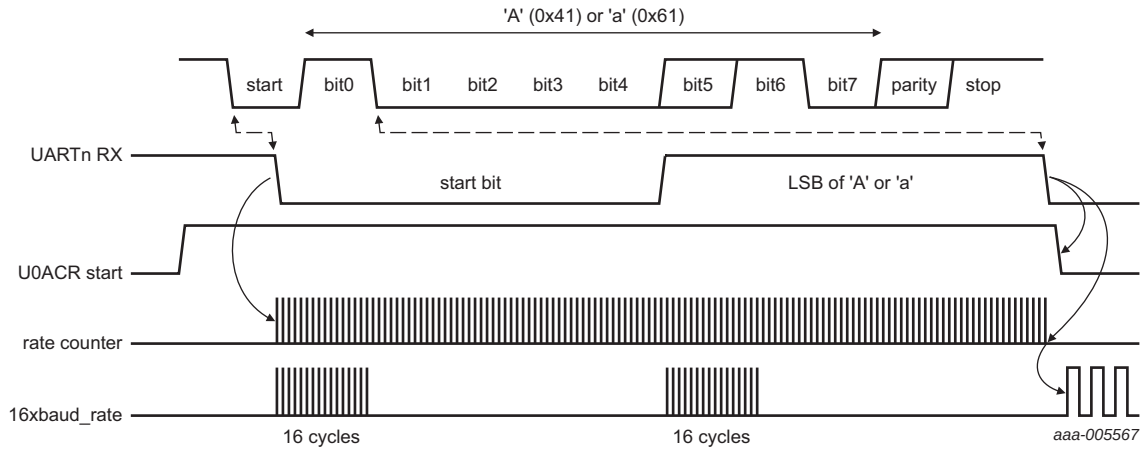
(1)

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

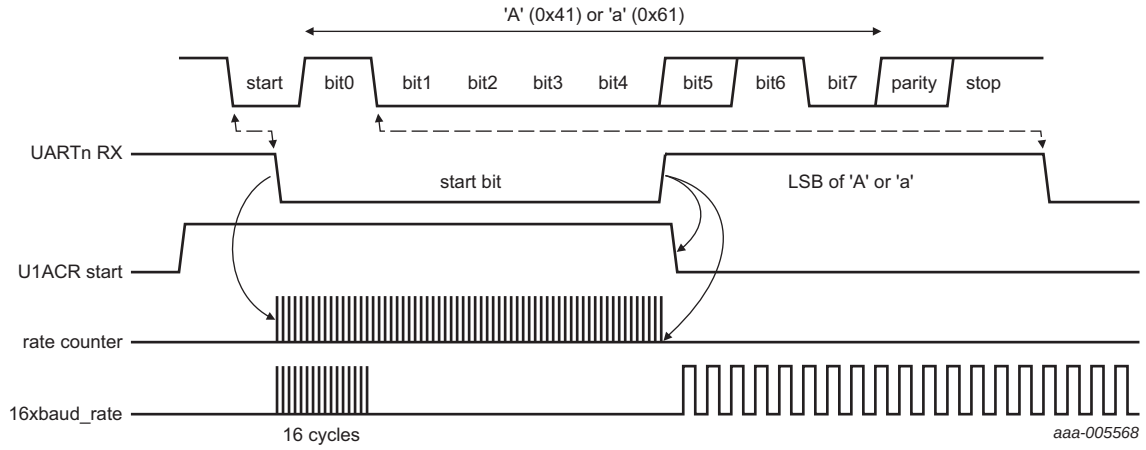
8.5.14 Auto-baud modes

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 executes 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 starts 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 continues incrementing with the pre-scaled USART input clock (UART_PCLK).
5. If Mode = 0, the rate counter stops on next falling edge of the USART Rx pin. If Mode = 1, the rate counter stops on the next rising edge of the USART Rx pin.
6. The rate counter is loaded into DLM/DLL and the baud rate is switched to normal operation. After setting the DLM/DLL, the end of auto-baud interrupt IIR ABEOInt is set, if enabled. The RSR now continues 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 13. Auto-baud a) mode 0 and b) mode 1 waveform

8.5.15 IrDA Control Register (ICR - 0x4000 8024)

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 90: IrDA Control Register (ICR - 0x4000 8024) bit description

Bit	Symbol	Value	Description	Reset value
0	IRDAEN		IrDA mode enable	0
		0	IrDA mode on USART is disabled, USART acts as a standard USART.	
		1	IrDA mode on USARTn is enabled.	
1	IRDAINV		Serial input inverter	0
		0	The serial input is not inverted.	

Table 90: IrDA Control Register (ICR - 0x4000 8024) bit description *...continued*

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 × T_{PCLK}	
		0x2	4 × T_{PCLK}	
		0x3	8 × T_{PCLK}	
		0x4	16 × T_{PCLK}	
		0x5	32 × T_{PCLK}	
		0x6	64 × T_{PCLK}	
		0x7	128 × 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). Set the value of these bits so that the resulting pulse width is at least 1.63 μs. Table 91 shows the possible pulse widths.

Table 91: IrDA Pulse Width

FixPulseEn	PulseDiv	IrDA Transmitter Pulse width (μs)
0	x	3 / (16 × baud rate)
1	0	2 × T_{PCLK}
1	1	4 × T_{PCLK}
1	2	8 × T_{PCLK}
1	3	16 × T_{PCLK}
1	4	32 × T_{PCLK}
1	5	64 × T_{PCLK}
1	6	128 × T_{PCLK}
1	7	256 × T_{PCLK}

8.5.16 USART Fractional Divider Register (FDR - 0x4000 8028)

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 92. 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 ensuring that USART is fully software and hardware compatible with USARTs not equipped with this feature.

The USART baud rate can be calculated as:

$$UART_{baudrate} = \frac{PCLK}{16 \times (256 \times U0DLM + U0DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)} \quad (2)$$

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.

8.5.16.1 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.

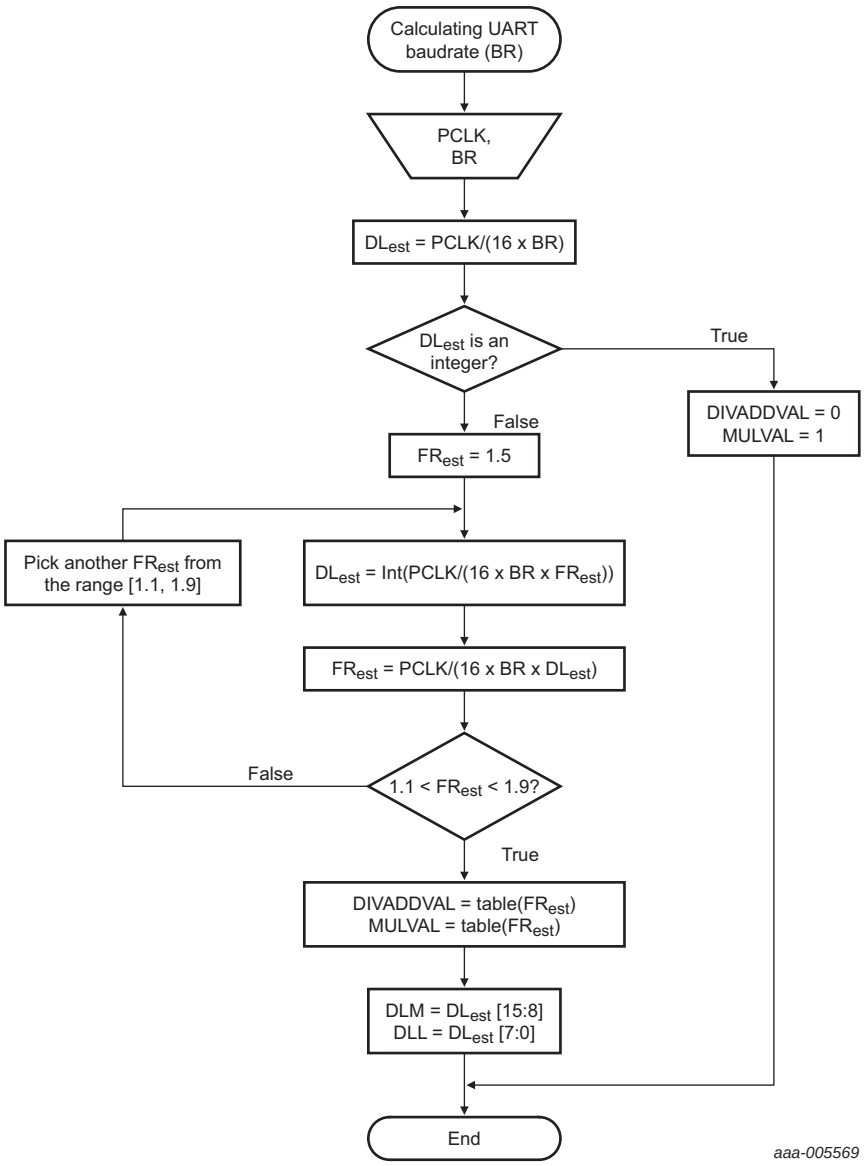


Fig 14. Algorithm for setting USART dividers

Table 93. 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

Table 93. Fractional Divider setting look-up table *...continued*

FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal	FR	DivAddVal/ MulVal
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

8.5.16.1.1 Example 1: UART_PCLK = 14.7456 MHz, BR = 9600

According to the provided algorithm DL_{est} = PCLK/(16 × BR) = 14.7456 MHz/(16 × 9600) = 96. Since this DL_{est} is an integer number, DIVADDVAL = 0, MULVAL = 1, DLM = 0, and DLL = 96.

8.5.16.1.2 Example 2: UART_PCLK = 12.0 MHz, BR = 115200

According to the provided algorithm DL_{est} = PCLK/(16 × BR) = 12 MHz/(16 × 115200) = 6.51. This DL_{est} is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of FR_{est} = 1.5 a new DL_{est} = 4 is calculated and FR_{est} is recalculated as FR_{est} = 1.628. Since FR_{est} = 1.628 is within the specified range of 1.1 and 1.9, DIVADDVAL and MULVAL values can be obtained from the attached look-up table.

The closest value for FR_{est} = 1.628 in the look-up Table 93 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 2, the baud rate of the USART is 115384. This rate has a relative error of 0.16 % from the originally specified 115200.

8.5.17 USART Oversampling Register (OSR - 0x4000 802C)

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 94. 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. 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).

8.5.18 USART Transmit Enable Register

This register enables implementation of software flow control. When TXEN = 1, the USART transmitter keeps sending data as long as it is available. As soon as TXEN becomes 0, USART transmission stops at the end of the current character.

Although Table 95 describes how to use TXEN bit in order to achieve hardware flow control, if the signals are available it is better to use the USART hardware-implemented flow-control features, and use TXEN only for software flow control.

Table 95. USART Transmit Enable Register (TER - address 0x4000 8030) bit description

Bit	Symbol	Description	Reset Value
6:0	-	Reserved	-
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 a hardware-handshaking Tx-permit signal (typically $\overline{CTS}$) 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
31:1	-	Reserved	-

8.5.19 Smart Card Interface Control register (SCICTRL - 0x4000 8048)

This register allows the USART to be used in asynchronous smart card applications.

Table 96. 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	
7:5	TXRETRY		When the protocol selection T bit (see preceding) is 0, the field controls the maximum number of retransmissions that the USART attempts 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 (see preceding) 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

8.5.19.1 Smart Card Connection

When the SCIEN bit in the SCICTRL register is set as described above, the USART provides bidirectional serial data on the TXD pin. No RXD pin is used when SCIEN is 1. If a USART SCLK function is enabled in the I/O Configuration block, a serial clock is output on the pin: use of such a clock is optional for smart cards. Software must use timers to implement character and block waiting times (no hardware support via trigger signals is provided on the EM783). GPIO pins can be used to control the smart card reset and power pins.

8.5.19.2 Smart Card Setup

The following must be set up in smart card applications:

- If necessary, program PRESETCTRL (Section 3.4.2) so that the USART is (reset? then) not continuously reset.
- Program one IOCON register to enable a USART TXD function.

- If the smart card to be communicated with requires (or may require) a clock, program one IOCON register for the USART SCLK function. The USART will use it as an output.
- Program UARTCLKDIV (Section 3.4.17) for an initial USART frequency of 3.58 MHz.
- Program the OSR (Section 8.5.17) for 372x oversampling.
- If necessary, program the DLM and DLL (Section 8.5.3) to 00 and 01 respectively, to pass the USART clock through without division.
- Program the LCR (Section 8.5.8) for 8-bit characters, parity enabled, even parity.
- Program the GPIO signals associated with the smart card so that (in this order):
  - Reset is low.
  - VCC is provided to the card (GPIO pins do not have the required 200 mA drive).
  - VPP (if provided to the card) is at “idle” state.
- Program SCICTRL (Section 8.5.19) 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 (Section 3.4.15) to enable the USART clock.

Thereafter, software should monitor the USART and timer status to interact with the smart card as described in ISO 7816 3.2.b and subsequently.

8.5.20 USART RS485 Control register

The RS485CTRL register controls the configuration of the USART in RS-485/EIA-485 mode.

Table 97. 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.	
		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 $\overline{\text{RTS}}$ is used for direction control.	
		1	If direction control is enabled (bit DCTRL = 1), pin $\overline{\text{DTR}}$ is used for direction control.	

Table 97. USART RS485 Control register (RS485CTRL - address 0x4000 804C) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
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 RTS (or DTR) pin.	0
		0	The direction control pin is driven to logic 0 when the transmitter has data to be sent. It is driven to logic 1 after the last bit of data has been transmitted.	
		1	The direction control pin is driven to logic 1 when the transmitter has data to be sent. It is 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

8.5.21 USART RS-485 Address Match register

The RS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

Table 98. 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	-

8.5.22 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 RTS (or DTR). This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

Table 99. USART RS-485 Delay value register (RS485DLY - address 0x4000 80454) 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

8.5.23 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 identifies an address character by setting the parity (ninth) 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 are ignored and not stored in the RXFIFO. When an address byte is detected (parity bit = ‘1’) it is 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 are accepted and stored in the RXFIFO regardless of whether they are data or address. When an address character is received a parity error interrupt is generated and the processor can decide whether 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 compares 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 is 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 is pushed onto the RXFIFO along with the parity bit, and the receiver is automatically enabled (RS485CTRL bit 1 is cleared by hardware). The receiver also generates an Rx Data Ready Interrupt.

While the receiver is enabled (RS485CTRL bit 1 = ‘0’), all bytes received is accepted and stored in the RXFIFO until an address byte which does not match the RS485ADRMATCH value is received. When this occurs, the receiver is automatically disabled in hardware (RS485CTRL bit 1 is set), The received non-matching address character is not stored in the RXFIFO.

RS-485/EIA-485 Auto Direction Control

RS485/EIA-485 mode includes the option of allowing the transmitter automatically to 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, uses the $\overline{\text{RTS}}$ pin.

When Auto Direction Control is enabled, the selected pin is asserted (driven LOW) when the CPU writes data into the TXFIFO. The pin is 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 except for 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 is driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin is driven to logic 0 after the last bit of data has been transmitted.

8.5.24 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 100. 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 register bypass.	0
		0	The input clock is synchronized before being used in clock edge detection logic.	
		1	The input clock is not synchronized before being used in clock edge detection logic. This allows for a higher input clock rate at the expense of potential metastability.	
4	CSCEN		Continuous master clock enable (used only when CSRC is 1)	0
		0	SCLK cycles only when characters are being sent on TxD	
		1	SCLK runs continuously (characters can be received on RxD independently from transmission on TxD)	

Table 100. USART Synchronous mode control register (SYNCCTRL - address 0x4000 8058) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
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
		0	CSCEN is under software control.	
		1	Hardware clears CSCEN after each character is received.	
31:6	-		Reserved. The value read from a reserved bit is not defined.	NA

After reset, synchronous mode is disabled. Synchronous mode is enabled by setting the SYNC bit. When SYNC is 1, the 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 does not send or sample 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 EM783 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 EM783 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.

8.6 Architecture

The architecture of the USART is shown in Figure 15 “USART 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 UART_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.

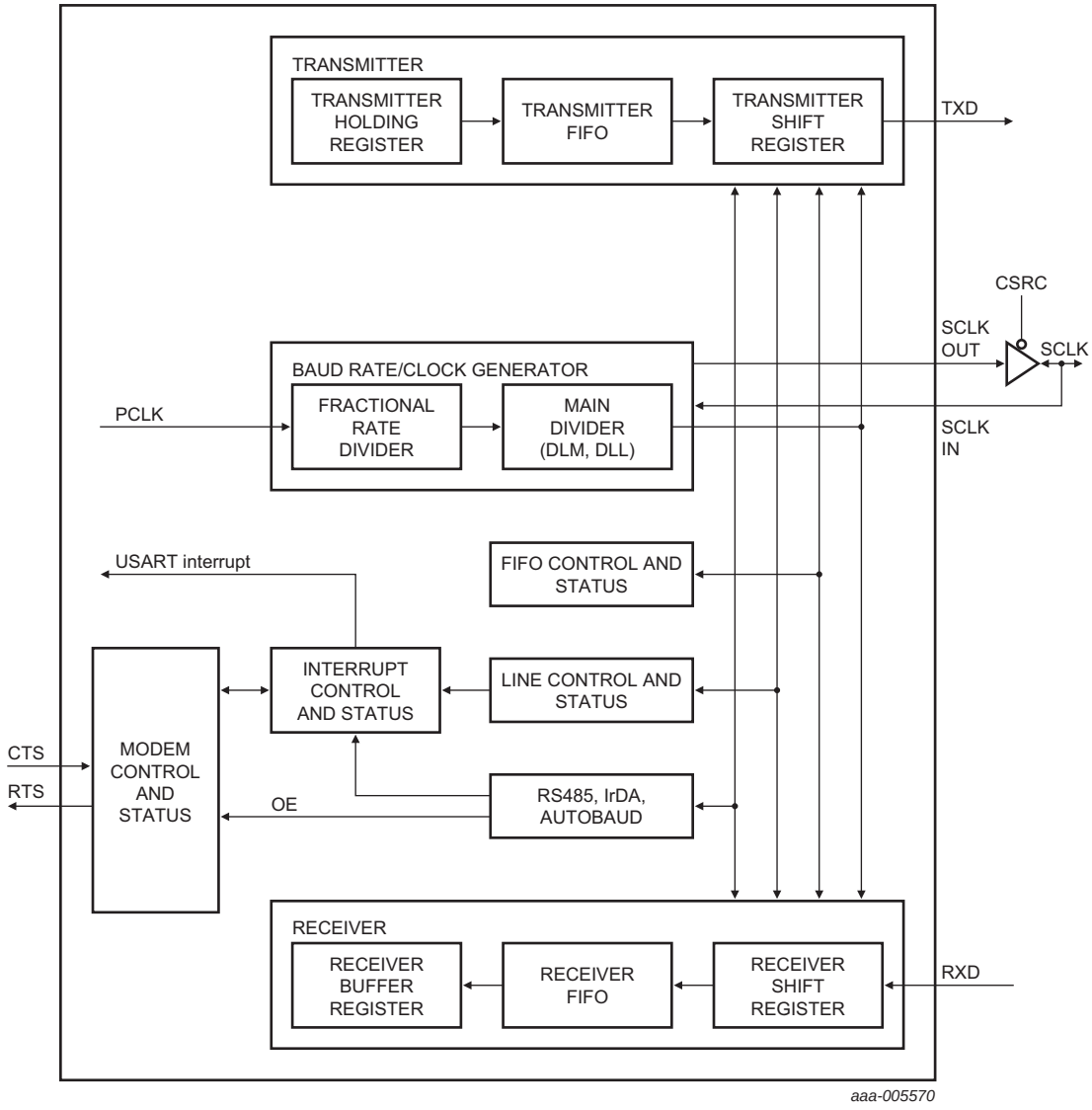


Fig 15. USART block diagram

9.1 How to read this chapter

The I²C-bus block is identical for all EM783 parts.

9.2 Basic configuration

The I²C-bus interface is configured using the following registers:

1. Primary I2C pins (P0_2 and P0_3): Configure the I2C pin functions and the I2C mode for the primary I2C pins in the IOCONFIG register block (Table 67).
2. Alternate I2C pins: Configure the I2C pin functions and the I2C mode for the alternate I2C pins in the IOCONFIG register block (Table 65).
3. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 5 (Table 19).

9.3 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.
- Two pins include standard I²C-compliant transceivers.
- Clock and data functions each have 3 other alternative pins with pseudo-open-drain drivers, maximizing the possibility of combining I²C with other necessary functions.

9.4 Applications

Interfaces to external I²C standard parts, such as serial RAMs, LCDs, tone generators, other microcontrollers, etc.

9.5 General description

A typical I^2C-bus configuration is shown in Figure 16. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I^2C-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 returns an acknowledge bit. Next the data bytes are transmitted from the slave to the master. The master returns an acknowledge bit after each received byte other than the last byte. At the end of the last received byte, a “not acknowledge” is returned. The master device generates the serial clock pulses and START and STOP conditions. A transfer ends with a STOP condition or a Repeated START condition. The I^2C bus is released after a STOP condition but not for a Repeated START (which begins another serial transfer).

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.

When the I^2C is connected via the “primary” pins shown in Table 101, the electrical characteristics are fully compliant with the I^2C specification, including the ability to power-down the EM783 without interfering with other devices on the I^2C-bus. (This is not true of the alternate pins).

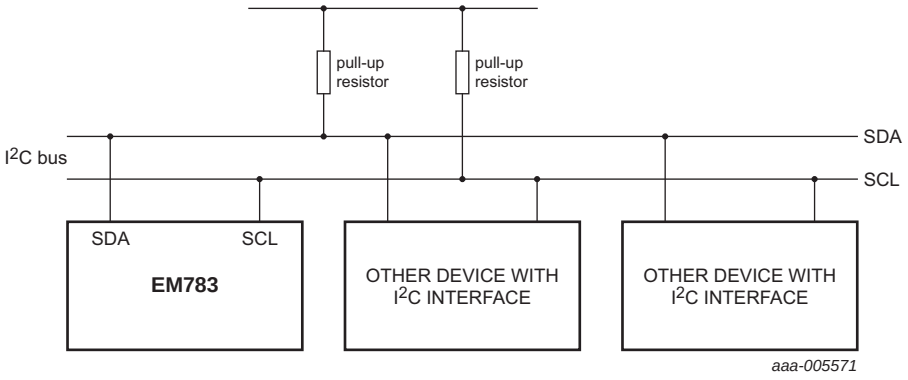


Fig 16. I^2C-bus configuration

9.5.1 I^2C Fast-mode Plus

Fast-Mode Plus supports transfer at up to1 Mbit/sec. In order to use Fast-Mode Plus, the primary I^2C pins must be used and properly configured in the IOCONFIG register block.

9.6 Pin description

If the Primary pins are used for I^2C, the HS and HD fields of their I/O Configuration registers (Table 67 and Table 68) should be programmed as described in Section 6.2.6. If alternate pins are used, the OD bits of their I/O Configuration registers (Table 65 and Table 69) should be set, and optionally internal pull-ups can be selected in the MODE

fields. Using internal pull-ups may result in rise times that are not compatible with the desired speed, in which case external resistors can be used with the MODE fields set to “Inactive” state.

Table 101. I^2C-bus pin description

Function	Primary Pin	Alternate Pins	Type	Description
SCL	P0_2	P0_12, P0_16, P0_24	Input/Output	I^2C Serial Clock
SDA	P0_3	P0_13, P0_15, P0_25	Input/Output	I^2C Serial Data

9.6.1 External Pull-ups

External pull-up resistors on alternate pins should not be connected to a voltage higher than V_{DD}. This restriction does not apply to the primary pins.

9.7 Register description

Table 102. Register overview: I^2C (base address 0x4000 0000)

Name	Access	Address offset	Description	Reset value[1]
CONSET	R/W	0x000	**I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I^2C control register is set. Writing a zero has no effect on the corresponding bit in the I^2C control register.	0x00
STAT	RO	0x004	**I2C Status Register.** During I^2C operation, this register provides detailed status codes that allow software to determine the next action needed.	0xF8
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
ADR0	R/W	0x00C	**I2C Slave Address Register 0.** Contains the 7-bit slave address for operation of the I^2C 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
SCLH	R/W	0x010	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I^2C clock.	0x04
SCLL	R/W	0x014	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I^2C clock. SCLL and SCLH together determine the clock frequency generated by an I^2C master and certain times used in slave mode.	0x04
CONCLR	WO	0x018	**I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I^2C control register is cleared. Writing a zero has no effect on the corresponding bit in the I^2C control register.	NA
MMCTRL	R/W	0x01C	**Monitor mode control register.**	0x00
ADR1	R/W	0x020	**I2C Slave Address Register 1.** Contains the 7-bit slave address for operation of the I^2C 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
ADR2	R/W	0x024	**I2C Slave Address Register 2.** Contains the 7-bit slave address for operation of the I^2C 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 102. Register overview: I²C (base address 0x4000 0000) *…continued*

Name	Access	Address offset	Description	Reset value[1]
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
DATA_ BUFFER	RO	0x02C	**Data buffer register.** The contents of the 8 MSBs of the DAT shift register will be transferred to the DATA_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus.	0x00
MASK0	R/W	0x030	**I2C Slave address mask register 0**. This mask register is associated with ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’).	0x00
MASK1	R/W	0x034	**I2C Slave address mask register 1**. This mask register is associated with ADR1 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’).	0x00
MASK2	R/W	0x038	**I2C Slave address mask register 2**. This mask register is associated with ADR2 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’).	0x00
MASK3	R/W	0x03C	**I2C Slave address mask register 3**. This mask register is associated with ADR3 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’).	0x00

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

9.7.1 I²C Control Set register (CONSET)

The CONSET register controls 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 103. 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
5	STA	START flag.	0
6	I2EN	I²C interface enable.	0
7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

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”.

Do not use I2EN to temporarily release the I^2C-bus since, when I2EN is reset, the I^2C-bus status is lost. Use the AA flag instead.

STA is the START flag. Setting this bit causes the I^2C interface to enter master mode and transmit a START condition or transmit a Repeated START condition if it is already in master mode.

When STA is 1 and the I^2C 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 frees the bus) and generates a START condition after a delay of a half clock period of the internal clock generator. If the I^2C interface is already in master mode and data has been transmitted or received, it transmits a Repeated START condition. STA may be set at any time, including when the I^2C 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 is generated.

If STA and STO are both set, then a STOP condition is transmitted on the I^2C-bus if the interface is in master mode, and transmits a START condition thereafter. If the I^2C interface is in slave mode, an internal STOP condition is generated, but is not transmitted on the bus.

STO is the STOP flag. Setting this bit causes the I^2C interface to transmit a STOP condition in master mode, or recover from an error condition in slave mode. When STO is 1 in master mode, a STOP condition is transmitted on the I^2C-bus. When the bus detects the STOP condition, STO is cleared automatically.

In slave mode, setting this bit can recover from an error condition. In this case, no STOP condition is transmitted to the bus. The hardware behaves as if a STOP condition has been received and it switches to “not addressed” slave receiver mode. The STO flag is cleared by hardware automatically.

SI is the I^2C Interrupt Flag. This bit is set when the I^2C 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 CONCLR register.

AA is the Assert Acknowledge Flag. When set to 1, an acknowledge (low level to SDA) is returned during the acknowledge clock pulse on the SCL line in 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 ADR 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) is returned during the acknowledge clock pulse on the SCL line in the following situations:

1. A data byte has been received while the I²C is in the master receiver mode.
2. A data byte has been received while the I²C is in the addressed slave receiver mode.

9.7.2 I²C Status register (STAT)

The I²C Status register reflects the condition of the I²C interface. The I²C Status register is Read-Only.

Table 104. I²C 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 I²C interface.	0x1F

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 I²C states. When any of these states are entered, the SI bit is set. For a complete list of status codes, refer to tables from Table 119 to Table 124.

9.7.3 I²C 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 remains stable as long as the SI bit is set. 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.

Table 105. I²C 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

9.7.4 I²C Slave Address register 0 (ADR0)

This register is readable and writable and is only used when an I²C interface is set to slave mode. In master mode, this register has no effect. The LSB of ADR is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

Any of these registers which contain the bit 00x will be disabled and not match any address on the bus. The slave address register is cleared to this disabled state on reset. See also Table 112.

Table 106. 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

9.7.5 I²C SCL HIGH and LOW duty cycle registers (SCLH - 0x4000 0010 and SCLL- 0x4000 0014)

Table 107. 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

Table 108. 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

9.7.5.1 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):

$$I^2C_{bitfrequency} = \frac{I2CPCLK}{SCLH + SCLL} \quad (3)$$

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 109 gives some examples of I²C-bus rates based on I2C_PCLK frequency and SCLL and SCLH values.

Table 109. 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.

9.7.6 I²C Control Clear register (CONCLR)

The CONCLR register controls 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 110. 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

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.

9.7.7 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 111. 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 enters monitor mode. In this mode the SDA output is forced high. This prevents 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.	

Table 111. I²C Monitor mode control register (MMCTRL - 0x4000 001C) bit description

Bit	Symbol	Value	Description	Reset value
1	ENA_SCL		SCL output enable.	0
		0	When this bit is cleared to '0', the SCL output is forced high when the module is in monitor mode. As described above, this prevents 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]	
3	MATCH_ALL		Select interrupt register match.	0
		0	When this bit is cleared, an interrupt is only generated when a match occurs to one of the (up-to) four address registers described above. That is, the module responds 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 is generated on ANY address received. This enables the part to monitor all traffic on the bus.	

[1] When the ENA_SCL bit is cleared and the I²C is no longer able 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 9.7.9) 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' (that is, if the module is NOT in monitor mode).

9.7.7.1 Interrupt in Monitor mode

All interrupts occur as normal when the module is in monitor mode. This means that the first interrupt occurs 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 are 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 actually contains data transmitted by some other slave on the bus which was addressed by the master.

Following all of these interrupts, the processor may read the data register to see what was transmitted on the bus.

9.7.7.2 Loss of arbitration in Monitor mode

In monitor mode, the I²C module is not 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 most probably results 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 this 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.

9.7.8 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 ADR is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

Any of these registers which contain the bit 00x will be disabled and not match any address on the bus. All four registers are cleared to this disabled state on reset.

Table 112. 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

9.7.9 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 has 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 is transferred to the DATA_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus. This means that the processor has 9 bit transmission times to respond to the interrupt and read the data before it is overwritten.

The processor still has the ability to read DAT directly, as usual, and the behavior of DAT is not altered in any way.

Although the DATA_BUFFER register is primarily intended for use in monitor mode with the ENA_SCL bit = ‘0’, it is available for reading at any time under any mode of operation.

Table 113. 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

9.7.10 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’ causes 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 always read back as zeros.

When an address-match interrupt occurs, the processor has to read the data register (DAT) to determine what the received address was that caused the match.

Table 114. I^2C 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 always reads back as 0.	0
7:1	MASK	Mask bits.	0x00
31:8	-	Reserved. User software should not write ones to reserved bits. These bits always read back as 0’s.	0

9.8 Functional description

Figure 17 shows how the on-chip I^2C-bus interface is implemented, and the following text describes the individual blocks.

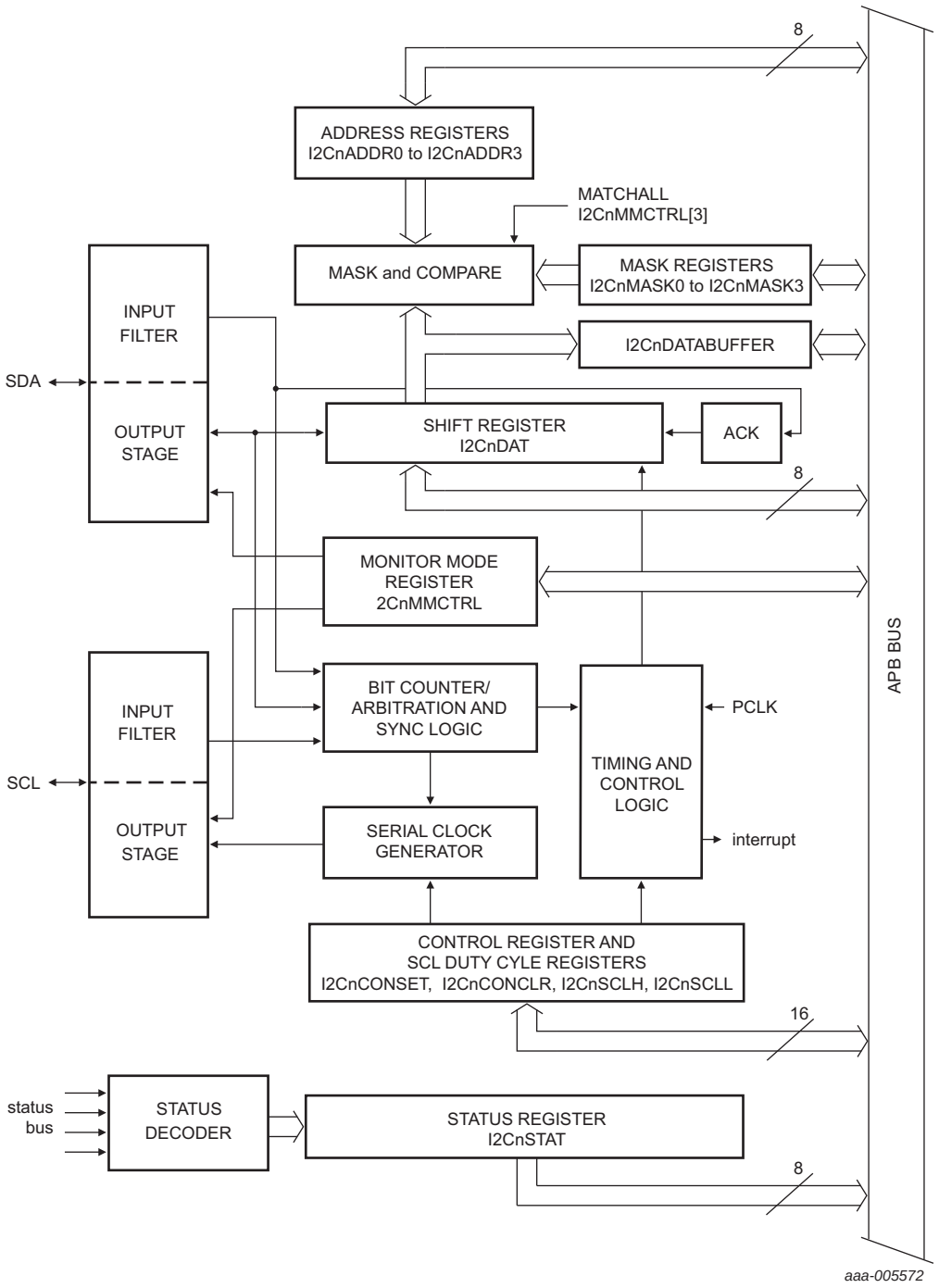


Fig 17. I²C serial interface block diagram

9.8.1 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.

9.8.2 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 responds 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.

9.8.3 Address masks, MASK0 to MASK3

The four mask registers each contain seven active bits (7:1). Any bit in these registers which is set to '1' causes 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.

If the ADRn bit 0 (GC enable bit) is as set and bits(7:1) are all zeroes, then the part responds to a received address = "0000000" regardless of the state of the associated mask register.

When an address-match interrupt occurs, the processor has to read the data register (DAT) to determine what the received address was that caused the match.

9.8.4 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.

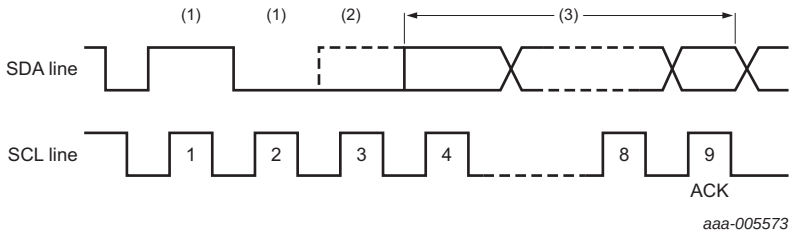
9.8.5 Shift register, DAT

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.

9.8.6 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 continues 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 18 shows the arbitration procedure.



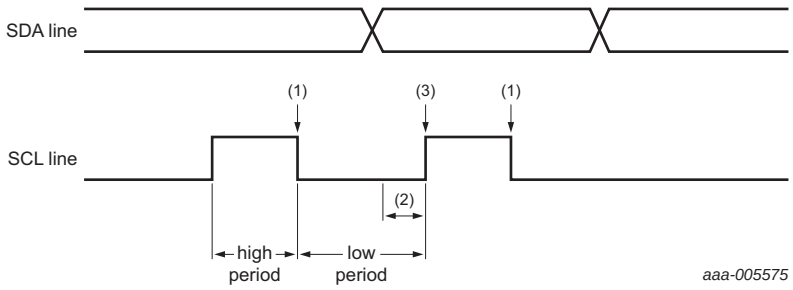
(1) Another device transmits serial data.

(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 18. Arbitration procedure

The synchronization logic synchronizes 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”. The “space” duration is determined by the device that generates the longest “spaces”. Figure 19 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 19. 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 stretches 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.

9.8.7 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 a slave mode. The I²C output clock frequency and duty cycle is programmable

via the I²C Clock Control Registers. See the description of the SCLL and SCLH registers for details. The output clock pulses have a duty cycle as programmed unless the bus is synchronizing with other SCL clock sources as earlier described.

9.8.8 Timing and control

The timing and control logic generates the timing and control signals for serial byte handling. This logic block does as follows:

- 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
- Monitors the I²C-bus status.

9.8.9 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 sets bits in the I²C control register that correspond to ones in the value written. Conversely, writing to CONCLR clears bits in the I²C control register that correspond to ones in the value written.

9.8.10 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).

9.9 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.

9.9.1 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 115. I2EN must be set to 1 to enable the I^2C function. If the AA bit is 0, the I^2C interface will not acknowledge any address when another device is master of the bus, so it cannot 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 115. 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 enters master transmitter mode when software sets the STA bit. The I^2C logic sends 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 loads the slave address and Write bit to the DAT register, and then clears 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. If the slave mode was enabled (by setting AA to 1) the possible status codes are 0x68, 0x78, or 0xB0m. The appropriate actions to be taken for each of these status codes are shown in Table 119 to Table 124.

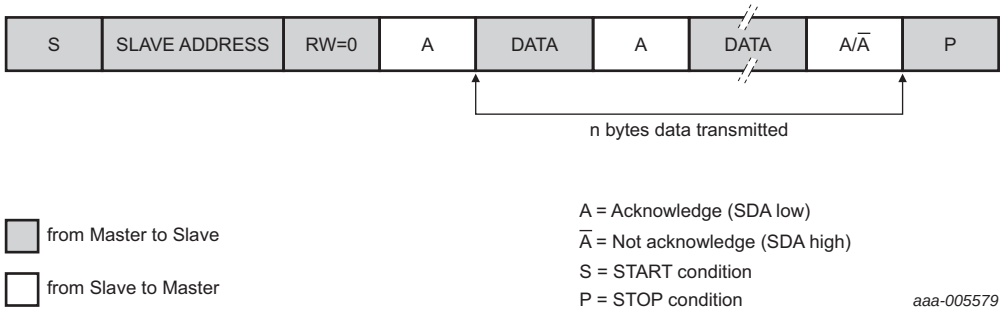


Fig 20. Format in the Master Transmitter mode

9.9.2 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 shows 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 120.

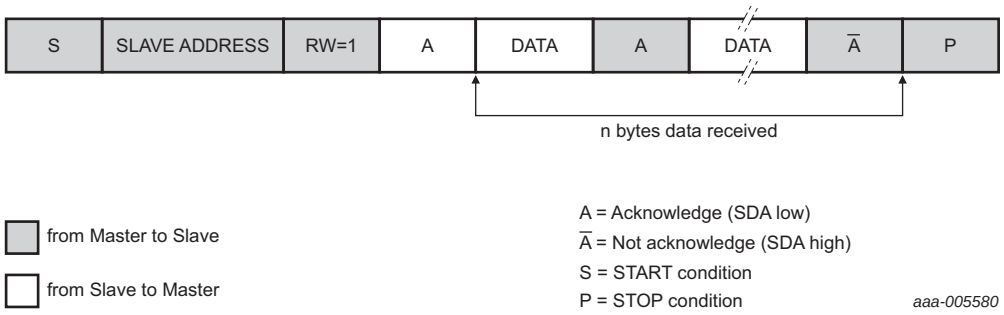


Fig 21. Format of Master Receiver mode

After a Repeated START condition, I^2C may switch to the master transmitter mode.

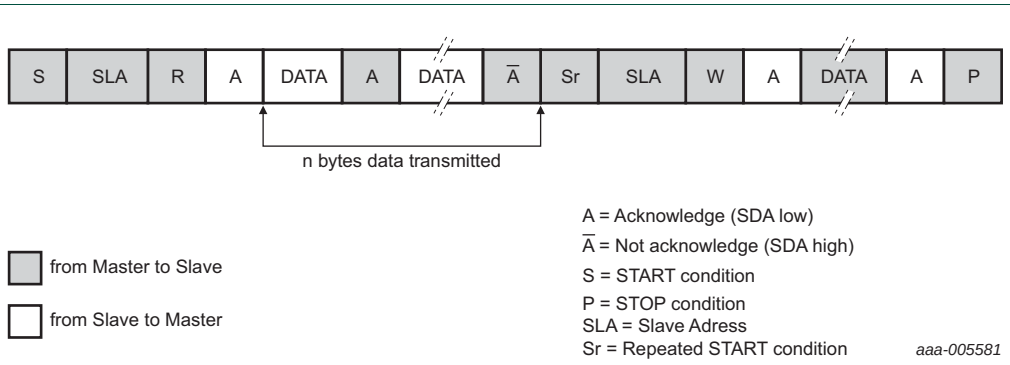


Fig 22. A Master Receiver switches to Master Transmitter after sending Repeated START

9.9.3 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 116.

Table 116. 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 cleared 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 123 for the status codes and actions.

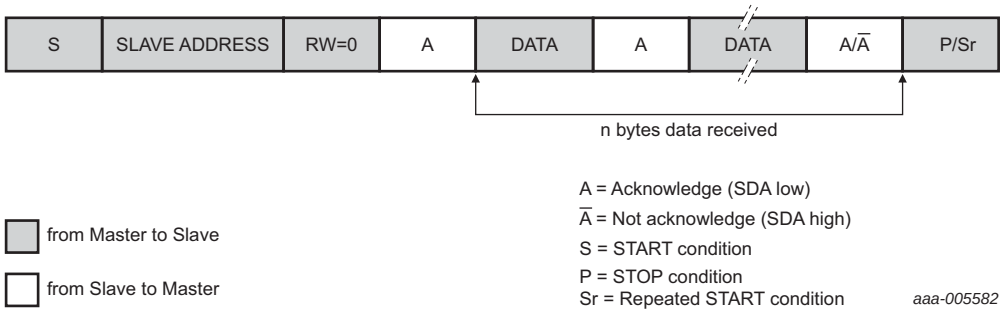


Fig 23. Format of Slave Receiver mode

9.9.4 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.

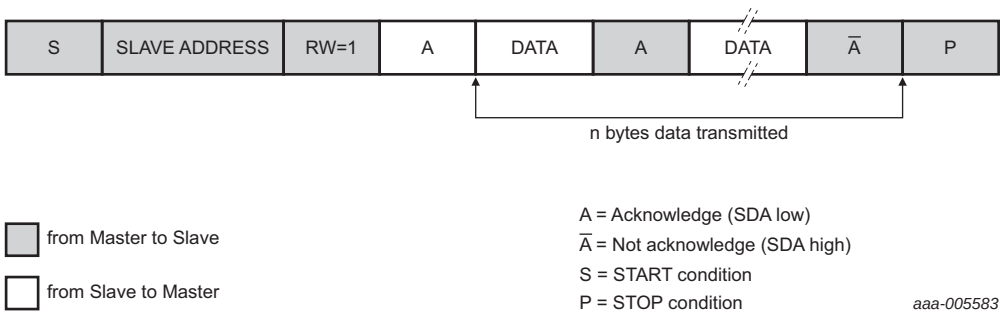


Fig 24. Format of Slave Transmitter mode

9.10 Details of I²C 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 25, Figure 26, Figure 27, Figure 28, and Figure 29. Table 117 lists abbreviations used in these figures when describing the I²C operating modes.

Table 117. Abbreviations used to describe I²C 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 25 to Figure 29, 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 119 to Table 125.

9.10.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 25). Before the master transmitter mode can be entered, CON must be initialized as follows:

Table 118. CON for 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 I²C rate must also be configured in the SCLL and SCLH registers. I2EN must be set to 1 to enable the I²C block. If the AA bit is reset, the I²C 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 I²C interface cannot enter a slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The I^2C logic now tests 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 119. 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 119. Master Transmitter 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+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.

Table 119. Master Transmitter 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	
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.

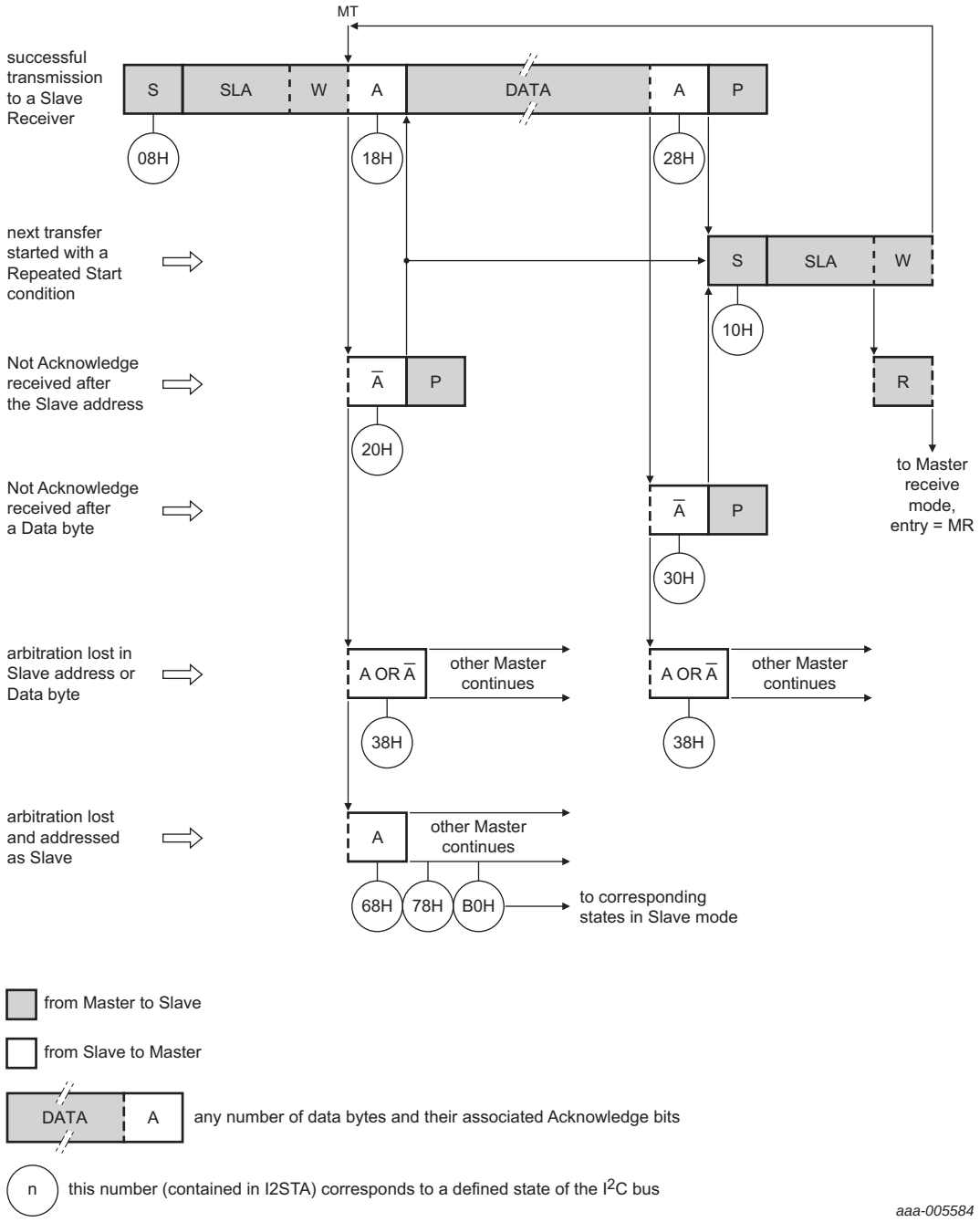


Fig 25. Format and states in Master Transmitter mode

9.10.2 Master Receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see Figure 26). 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 120. 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 120. 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 a 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.

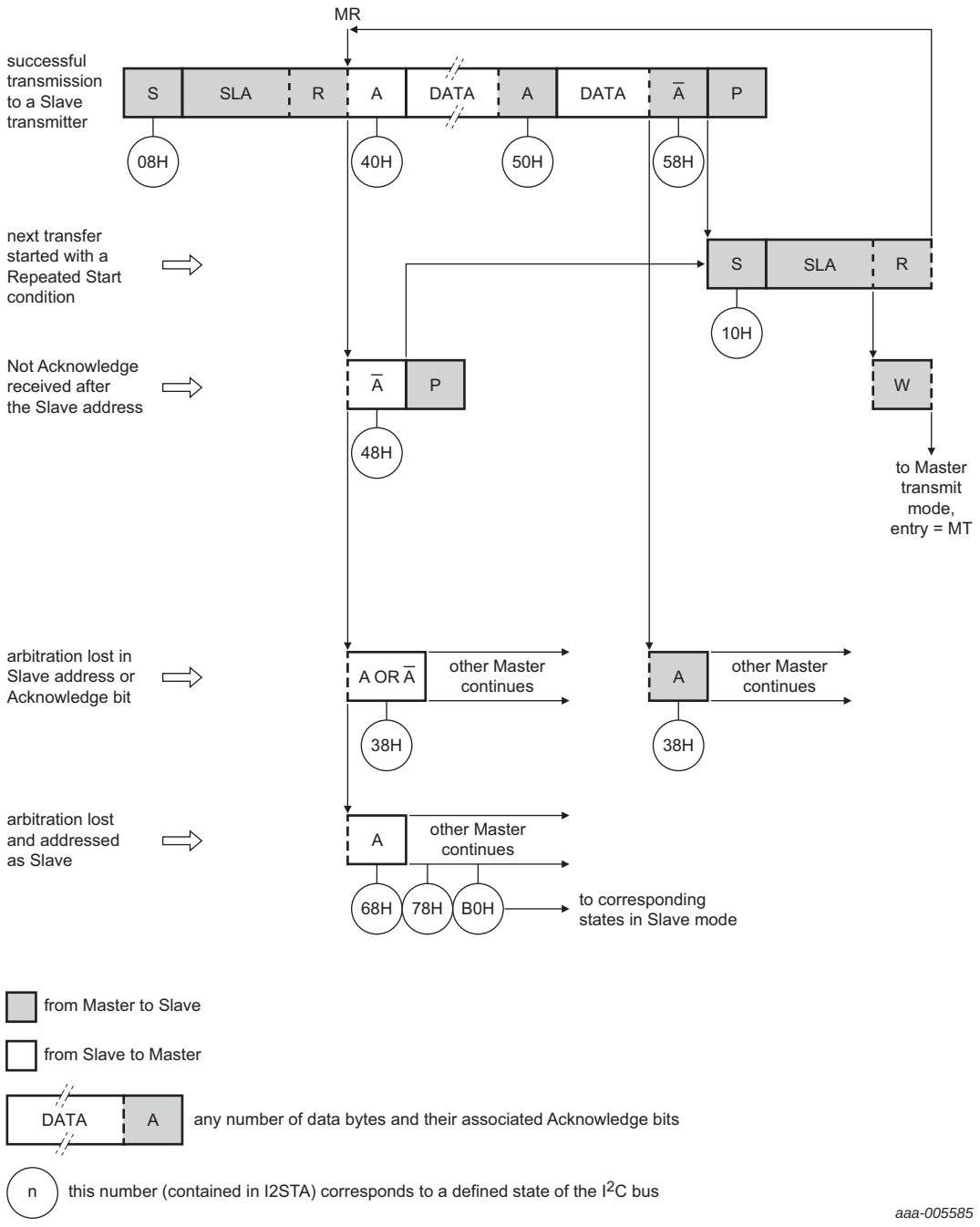


Fig 26. Format and states in Master Receiver mode

9.10.3 Slave Receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 27). To initiate the slave receiver mode, ADR and CON must be loaded as follows:

Table 121. 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 responds when addressed by a master. If the LSB (GC) is set, the I^2C block responds to the General Call address (0x00); otherwise it ignores the General Call address.

Table 122. CON for 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 123. 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 returns 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 123. Slave 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	
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.

Table 123. Slave Receiver mode *…continued*

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	
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 123. 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.

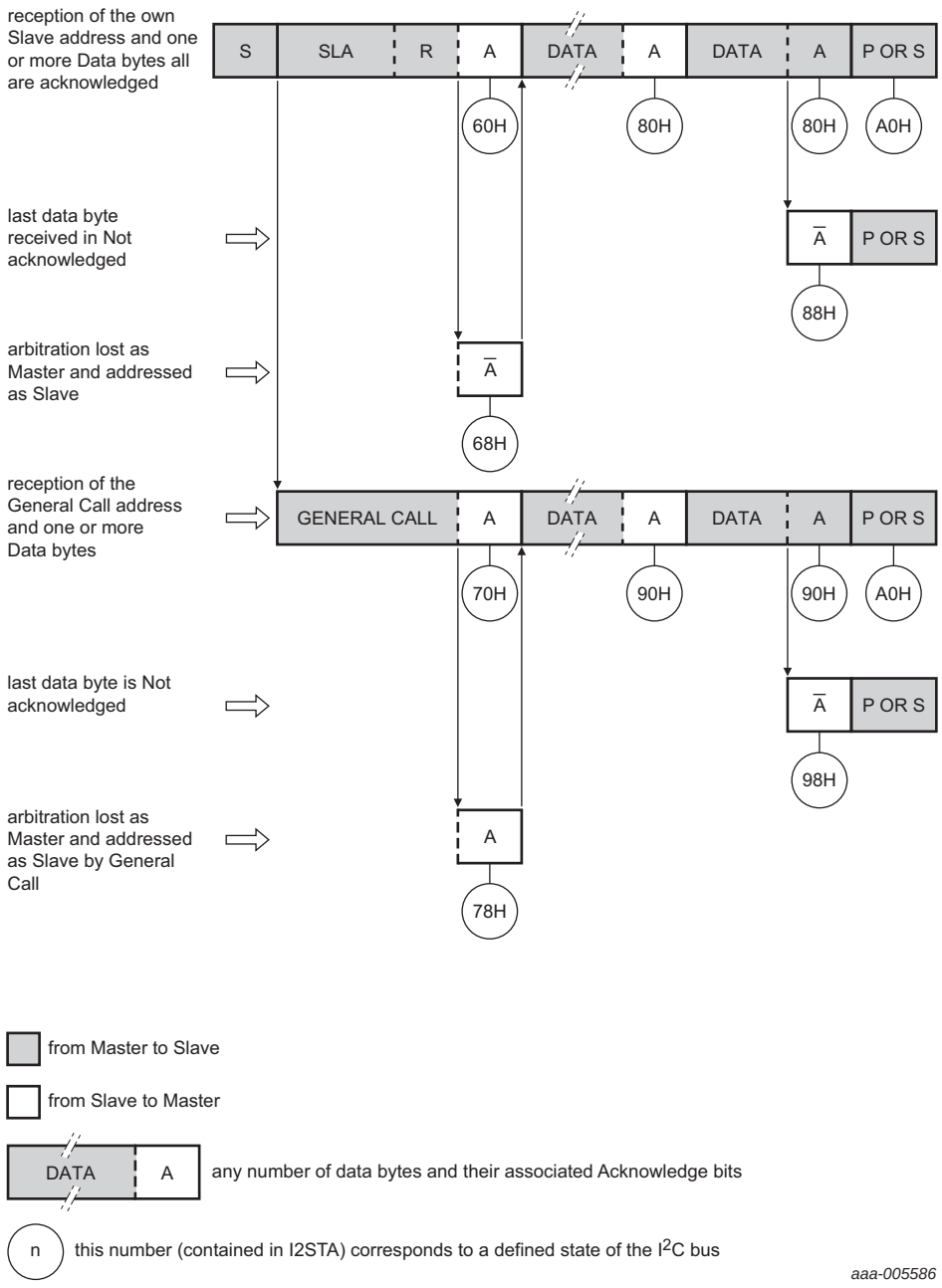


Fig 27. Format and states in Slave Receiver mode

9.10.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 28). 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 124. 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 transmits the last byte of the transfer and enters state 0xC0 or 0xC8. The I^2C block is switched to the not addressed slave mode and ignores 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 124. Slave Transmitter 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	
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.

Table 124. Slave Transmitter mode *…continued*

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	
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.

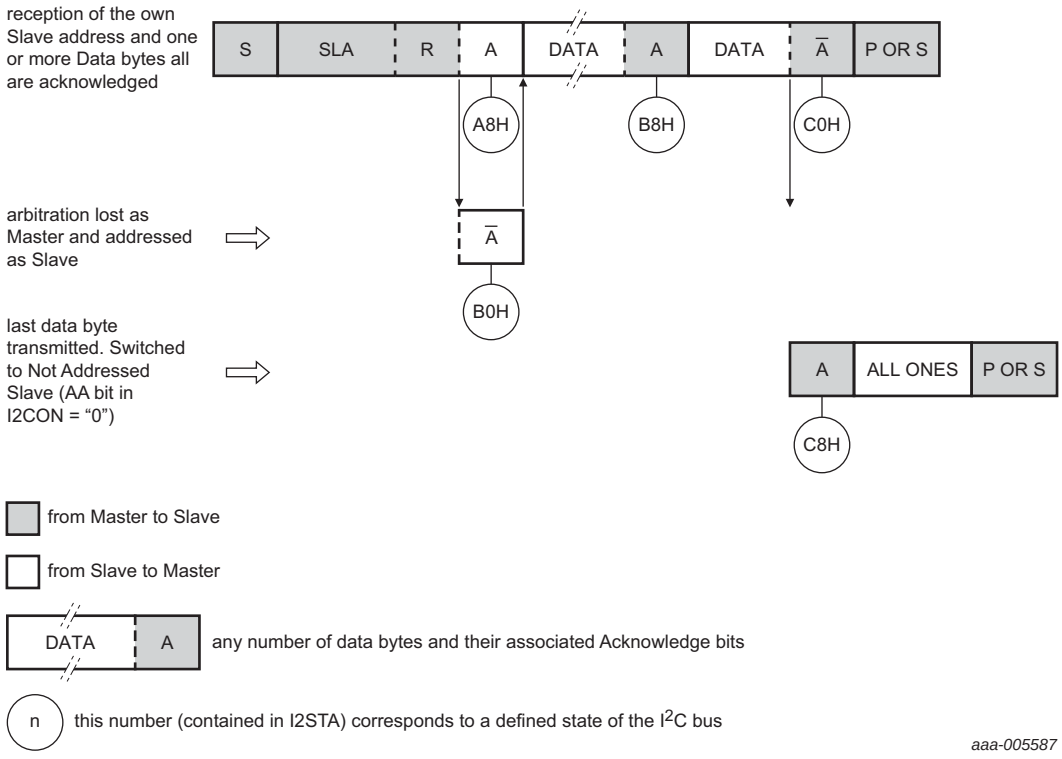


Fig 28. Format and states in Slave Transmitter mode

9.10.5 Miscellaneous states

There are two STAT codes that do not correspond to a defined I^2C hardware state (see Table 125). These codes are discussed as follows:

9.10.5.1 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.

9.10.5.2 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 I^2C 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 125. Miscellaneous States

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	
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 I^2C 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 I^2C block is switched to the not addressed SLV mode. STO is reset.

9.10.6 Some special cases

The I^2C 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 I^2C-bus
- I^2C-bus obstructed by a LOW level on SCL or SDA
- Bus error

9.10.6.1 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 29). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

If the I^2C hardware detects a Repeated START condition on the I^2C-bus before generating a Repeated START condition itself, it will release the bus, and no interrupt request is generated. If another master frees the bus by generating a STOP condition, the I^2C block transmits a normal START condition (state 0x08), and a retry of the total serial data transfer can commence.

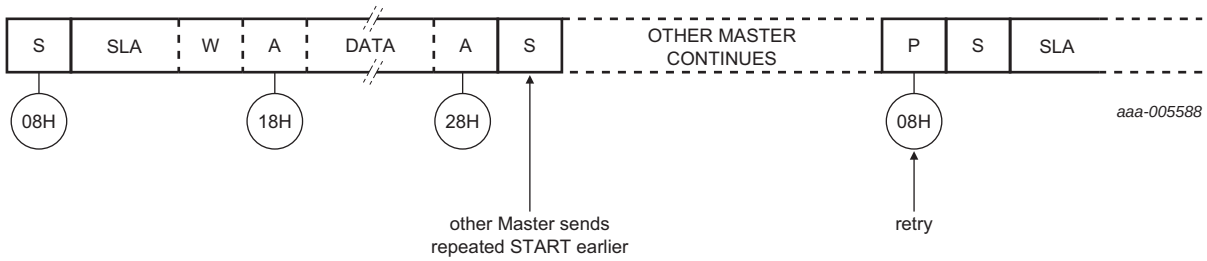


Fig 29. Simultaneous Repeated START conditions from two masters

9.10.6.2 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see Figure 18). Loss of arbitration is indicated by the following states in STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 25 and Figure 26).

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. A retry of the total serial transfer can now commence.

9.10.6.3 Forced access to the I^2C-bus

In some applications, it may be possible for an uncontrolled source to cause a bus hang-up. In such situations, the problem may be caused by interference, temporary interruption of the bus or a temporary short-circuit between SDA and SCL.

If an uncontrolled source generates a superfluous START or masks a STOP condition, then the I^2C-bus stays busy indefinitely. If the STA flag is set and bus access is not obtained within a reasonable amount of time, then a forced access to the I^2C-bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I^2C 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 30).

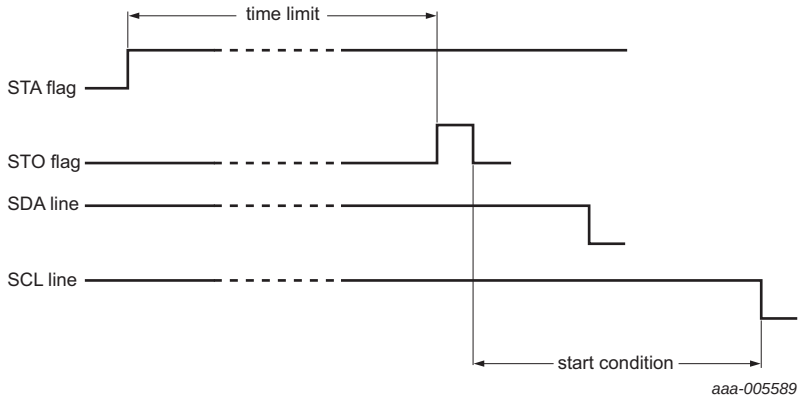
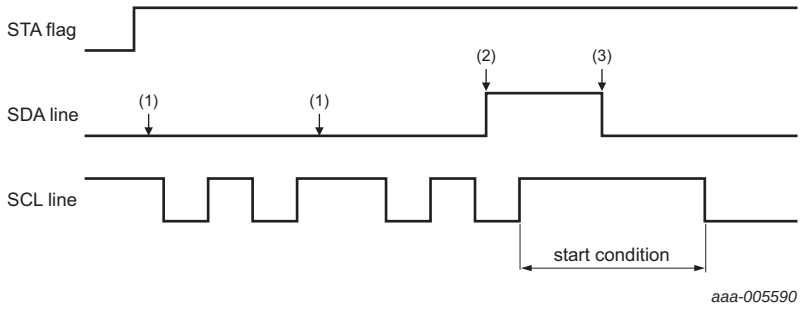


Fig 30. Forced access to a busy I²C-bus

9.10.6.4 I²C-bus obstructed by a Low 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 31). 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 31. Recovering from a bus obstruction caused by a LOW level on SDA

9.10.6.5 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 I^2C hardware only reacts to a bus error when it is involved in a serial transfer either as a master or an addressed slave. When a bus error is detected, the I^2C 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 125.

9.10.7 I^2C state service routines

This section provides examples of operations that must be performed by various I^2C state service routines. This includes:

- Initialization of the I^2C block after a Reset.
- I^2C Interrupt Service
- The 26 state service routines providing support for all four I^2C operating modes.

9.10.8 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.

9.10.9 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.

9.10.10 The state service routines

Each state routine is part of the I^2C interrupt routine and handles one of the 26 states.

9.10.11 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 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.

9.11 Software example

9.11.1 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.

9.11.2 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

9.11.3 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

9.11.4 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.

9.11.5 Non mode-specific states

9.11.5.1 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

9.11.5.2 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.

9.11.5.3 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

9.11.5.4 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

9.11.6 Master Transmitter states

9.11.6.1 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

9.11.6.2 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

9.11.6.3 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

9.11.6.4 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

9.11.6.5 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

9.11.7 Master Receive states

9.11.7.1 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

9.11.7.2 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

9.11.7.3 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

9.11.7.4 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

9.11.8 Slave Receiver states

9.11.8.1 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

9.11.8.2 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

9.11.8.3 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

9.11.8.4 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

9.11.8.5 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

9.11.8.6 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

9.11.8.7 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

9.11.8.8 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

9.11.8.9 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

9.11.9 Slave Transmitter states

9.11.9.1 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

9.11.9.2 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

9.11.9.3 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

9.11.9.4 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

9.11.9.5 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

10.1 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.

10.2 General description

The SPI/SSP is a Synchronous Serial Port (SSP) controller capable of operation on a SPI, 4-wire SSI, or Microwire bus. It can interact with multiple masters and slaves on the bus. Only a single master and a single slave can communicate on the bus during a given data transfer. Data transfers are in principle full duplex, with frames of 4 bits to 16 bits of data flowing from the master to the slave and from the slave to the master. In practice, often only one of these data flows carries meaningful data.

The EM783 has an SPI/Synchronous Serial Port controller.

10.3 Pin description

Table 126. SSP pin descriptions

Pin name	Type	Interface pin name/function			Pin description
		SPI	SSI	Microwire	
SCK0	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 SPI/SSP 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 SPI/SSP interface either holds it in its inactive state or does not drive it (leaves it in high-impedance state).
SSEL0	I/O	SSEL	FS	CS	**Frame Sync/Slave Select.** When the SPI/SSP 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 SPI/SSP 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 corresponding input of the slave. 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	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 SPI/SSP is a slave, serial data is output on this signal. When the SPI/SSP is a master, it clocks in serial data from this signal. When the SPI/SSP is a slave and is not selected by FS/SSEL, it does not drive this signal (leaves it in high-impedance state).
MOSI0	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 SPI/SSP is a master, it outputs serial data on this signal. When the SPI/SSP is a slave, it clocks in serial data from this signal.

Remark: Most or all of these signals can be assigned to a choice of pins. In order to use an SPI/SSP controller, each of its signals must be selected in only one IOCON register. See Section 6.3.1.

10.4 Clocking and power control

The SSP block is gated by the SYSAHBCLKCTRL register (see Table 19). The peripheral SSP clock, which is used by the SSP clock divider and prescaler, is controlled by the SSP0CLKDIV register (see Section 3.4.16).

The SPI0_PCLK clocks can be disabled in SSP0CLKDIV registers (see Section 3.4.16), and the SSP block can be disabled in the SYSAHBCLKCTRL register (Table 19) for power savings.

10.5 Register description

The register addresses of the SSP controller are shown in Table 127.

Table 127. Register overview: SPI0 (base address 0x4004 0000)

Name	Access	Address offset	Description	Reset Value[1]
CR0	R/W	0x000	Control Register 0. Selects the serial clock rate, bus type, and data size.	0
CR1	R/W	0x004	Control Register 1. Selects master/slave and other modes.	0
DR	R/W	0x008	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	0
SR	RO	0x00C	Status Register	-
CPSR	R/W	0x010	Clock Prescale Register	0
IMSC	R/W	0x014	Interrupt Mask Set and Clear Register	0
RIS	R/W	0x018	Raw Interrupt Status Register	-
MIS	R/W	0x01C	Masked Interrupt Status Register	0
ICR	R/W	0x020	Interrupt Clear Register	NA

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

10.5.1 SPI/SSP Control Register 0

This register controls the basic operation of the SPI/SSP controller.

Table 128: SPI/SSP Control Register 0 (CR0 - address 0x4004 0000 (SSP0)) 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	
		0x4	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.	

Table 128: SPI/SSP Control Register 0 (CR0 - address 0x4004 0000 (SSP0)) bit description

Bit	Symbol	Value	Description	Reset Value
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
31:9	-		Reserved	-

10.5.2 SPI/SSP0 Control Register 1

This register controls certain aspects of the operation of the SPI/SSP controller.

Table 129: SPI/SSP Control Register 1 (CR1 - address 0x4004 0004 (SSP0)) bit description

Bit	Symbol	Value	Description	Reset Value
0	LBM		Loop Back Mode.	0
		0	During normal operation.	
		1	Serial input is taken from the serial output (MOSI or MISO) rather than the serial input pin (MISO or MOSI respectively).	
1	SSE		SSP Enable.	0
		0	The SSP controller is disabled.	
		1	The SSP controller interacts with other devices on the serial bus. Software should write the appropriate control information to the other SPI/SSP registers and interrupt controller registers, before setting this bit.	
2	MS		Master/Slave Mode.This bit can only be written when the SSE bit is 0.	0
		0	The SSP controller acts as a master on the bus, driving the SCK, MOSI, and SSEL lines and receiving the MISO line.	
		1	The SSP controller acts as a slave on the bus, driving MISO line and receiving SCK, MOSI, and SSEL lines.	
3	SOD		Slave Output Disable. This bit is relevant only in slave mode (MS = 1). If it is 1, this blocks this SSP controller from driving the transmit data line (MISO).	0
7:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
31:8	-		Reserved	-

10.5.3 SPI/SSP Data Register

Software can write data to be transmitted to this register and read data that has been received.

Table 130: SPI/SSP Data Register (DR - address 0x4004 0008 (SSP0)) 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 SSP controller is not busy on the bus, transmission of the data begins immediately. Otherwise the data written to this register is 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 SSP 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	-

10.5.4 SPI/SSP Status Register

This read-only register reflects the current status of the SSP controller.

Table 131: SPI/SSP Status Register (SR - address 0x4004 000C (SSP0)) 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 SSP 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

10.5.5 SPI/SSP Clock Prescale Register

This register controls the factor by which the Prescaler divides the SSP 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 132: SPI/SSP Clock Prescale Register (CPSR - address 0x4004 0010 (SSP0)) 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, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Important: the SSPnCPSR value must be properly initialized, or the SSP controller will not be able to transmit data correctly.

In Slave mode, the SSP clock rate provided by the master must not exceed 1/12 of the SSP peripheral clock selected in Section 3.4.16. The content of the SSPnCPSR register is not relevant.

In master mode, $CPSDVSR_{min}$ = 2 or larger (even numbers only).

10.5.6 SPI/SSP Interrupt Mask Set/Clear Register

This register controls whether each of the four possible interrupt conditions in the SSP controller are enabled. Note that ARM (the designer of this peripheral) uses the word “masked” in the opposite sense from classic computer terminology, in which “masked” meant “disabled”. ARM uses the word “masked” to mean “enabled”. To avoid confusion, this chapter does not use the word “masked”.

Table 133: SPI/SSP Interrupt Mask Set/Clear register (IMSC - address 0x4004 0014 (SSP0)) 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”.	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

10.5.7 SPI/SSP Raw Interrupt Status Register

This read-only register contains a 1 for each interrupt condition that is asserted, regardless of whether the interrupt is enabled in the SSPIMSC registers.

Table 134: SPI/SSP Raw Interrupt Status register (RIS - address 0x4004 0018 (SSP0)) bit description

Bit	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”.	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

10.5.8 SPI/SSP Masked Interrupt Status Register

This read-only register contains a 1 for each interrupt condition that is asserted and enabled in the SSPIMSC registers. When an SSP interrupt occurs, the interrupt service routine should read this register to determine the causes of the interrupt.

Table 135: SPI/SSP Masked Interrupt Status register (MIS - address 0x4004 001C (SSP0)) 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.	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

10.5.9 SPI/SSP Interrupt Clear Register

Software can write one or more ones to this write-only register, to clear the corresponding interrupt conditions in the SSP controller. Note that the other two interrupt conditions can be cleared by writing or reading the appropriate FIFO or disabled by clearing the corresponding bit in SSPIMSC registers.

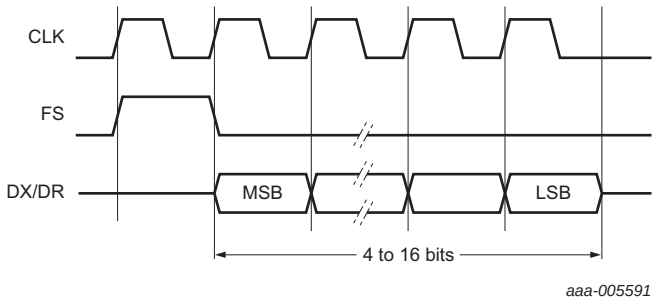
Table 136: SPI/SSP interrupt Clear Register (ICR - address 0x4004 0020 (SSP0)) bit description

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 time-out period” interrupt.	NA
31:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

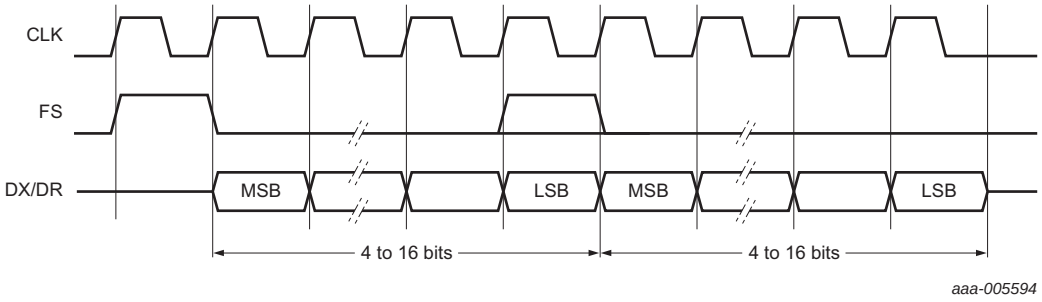
10.6 Functional description

10.6.1 Texas Instruments synchronous serial frame format

Figure 32 shows the 4-wire Texas Instruments synchronous serial frame format supported by the SSP module.



a. Single frame transfer



b. Continuous/back-to-back frames transfer

Fig 32. 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.

10.6.2 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.

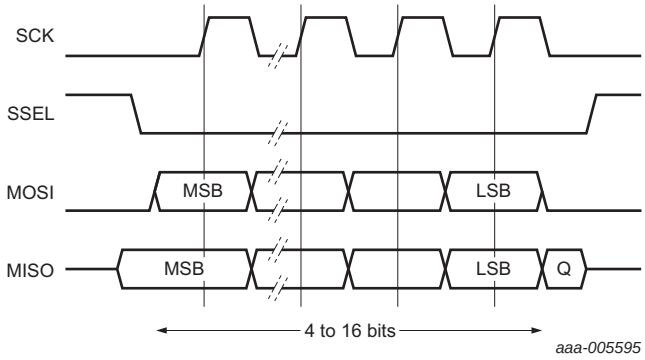
10.6.2.1 Clock Polarity (CPOL) and Clock Phase (CPHA) control

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.

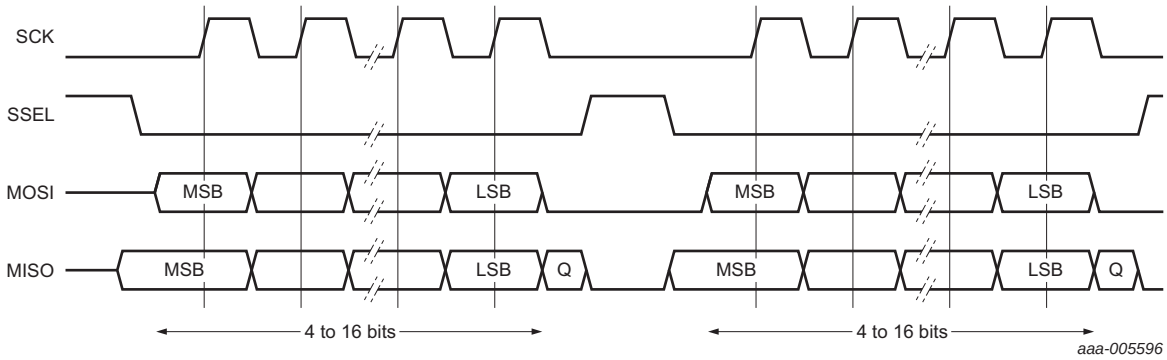
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is LOW, data is captured on the first clock edge transition. If the CPHA clock phase control bit is HIGH, data is captured on the second clock edge transition.

10.6.2.2 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 33.



a. Single transfer with CPOL=0 and CPHA=0



b. Continuous transfer with CPOL=0 and CPHA=0

Fig 33. 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 SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. This causes slave data to be enabled onto the MISO input line of the master. MOSI of the master 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.

10.6.2.3 SPI format with CPOL=0, CPHA=1

The transfer signal sequence for SPI format with CPOL=0, CPHA=1 is shown in Figure 34, which covers both single and continuous transfers.

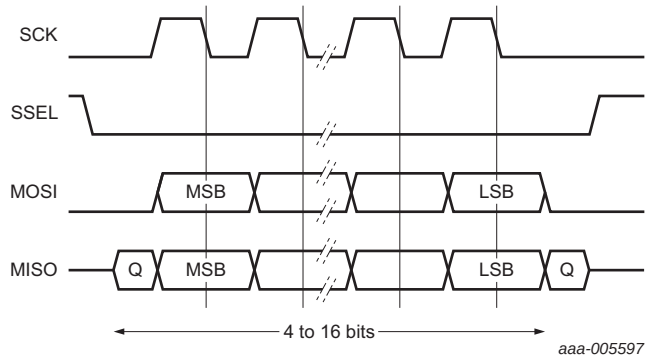


Fig 34. SPI frame format with CPOL=0 and CPHA=1

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 SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. MOSI pin of the master 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.

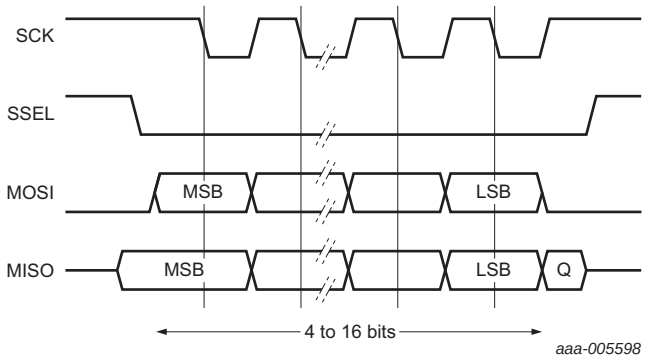
Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

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.

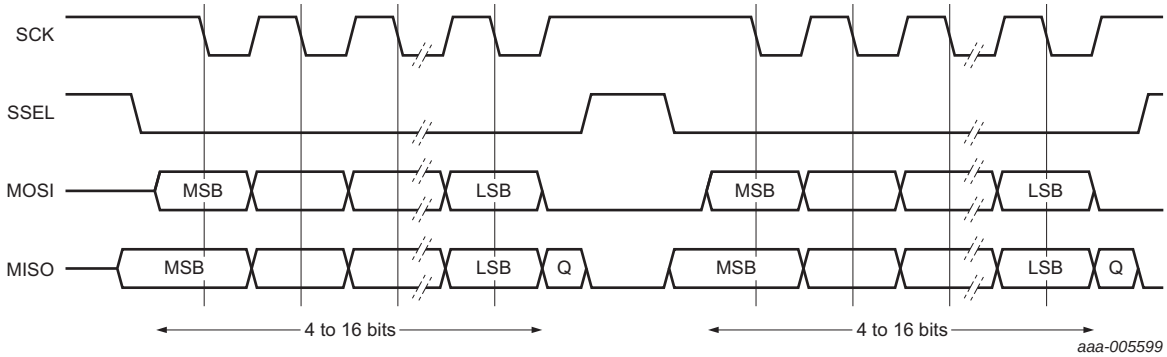
For continuous back-to-back transfers, the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

10.6.2.4 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 35.



a. Single transfer with CPOL=1 and CPHA=0



b. Continuous transfer with CPOL=1 and CPHA=0

Fig 35. SPI frame format with CPO=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 SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW, which causes slave data to be immediately transferred onto the MISO line of the master. MOSI pin of the master 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.

10.6.2.5 SPI format with CPOL=1, CPHA=1

The transfer signal sequence for SPI format with CPOL=1, CPHA=1 is shown in Figure 36, which covers both single and continuous transfers.

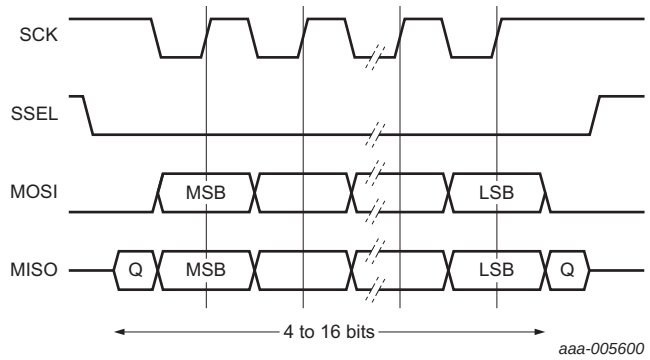


Fig 36. SPI Frame Format with CPOL=1 and CPHA=1

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 SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. MOSI pin of the master 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 earlier. 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.

10.6.3 National Semiconductor Microwire Format

Figure 37 shows the Microwire frame format for a single frame. Figure 38 shows the same format when back-to-back frames are transmitted.

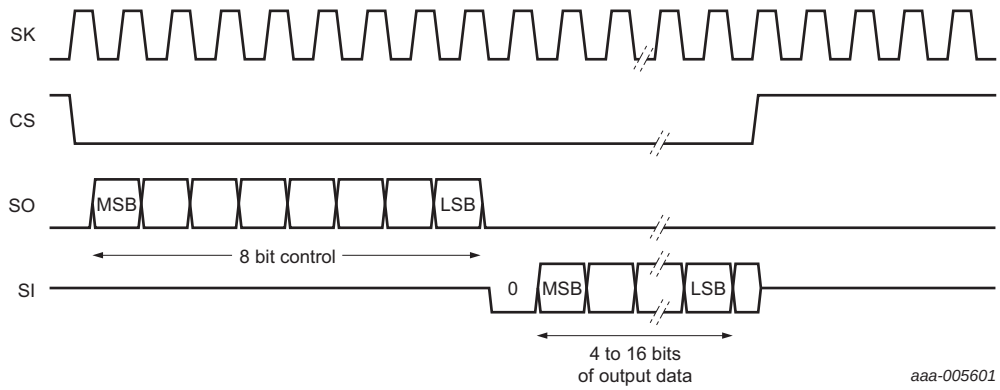


Fig 37. Microwire frame format (single transfer)

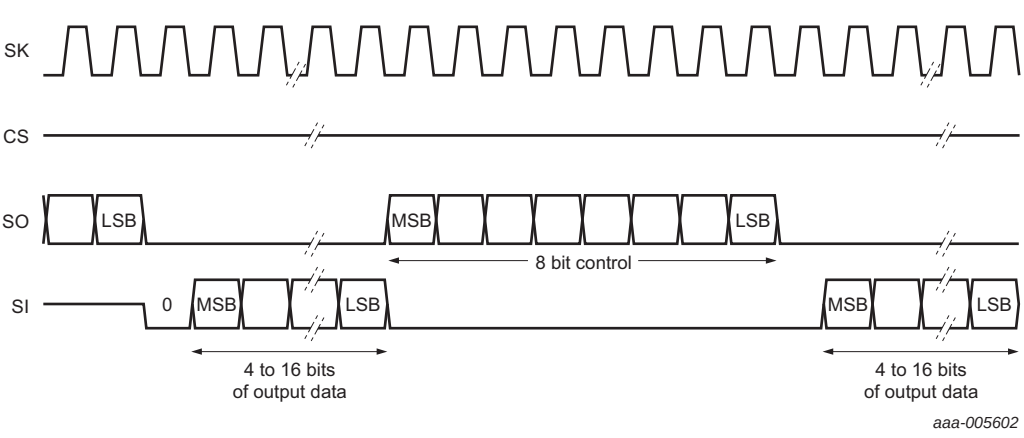


Fig 38. Microwire frame format (continuous transfers)

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 SPI/SSP to the off-chip slave device. During this transmission, no incoming data is received by the SPI/SSP. After the message has been sent, the off-chip slave decodes it and, after waiting

one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 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 tristated 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 SPI/SSP. Each bit is driven onto SI line on the falling edge of SK. The SPI/SSP in turn latches each bit on the rising edge of SK. At the end of the frame, for single transfers, the CS signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

Note: The off-chip slave device can tristate the receive line either on the falling edge of SK after the LSB has been latched by the receive shifter, or when the CS pin goes HIGH.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the CS line is continuously asserted (held LOW) and transmission of data occurs back to back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge SK, after the LSB of the frame has been latched into the SPI/SSP.

10.6.3.1 Setup and hold time requirements on CS with respect to SK in Microwire mode

In the Microwire mode, the SPI/SSP slave samples the first bit of receive data on the rising edge of SK after CS has gone LOW. Masters that drive a free-running SK must ensure that the CS signal has sufficient setup and hold margins with respect to the rising edge of SK.

Figure 39 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 SPI/SSP slave, CS must have a setup of at least two times the period of SK on which the SPI/SSP operates. With respect to the SK rising edge previous to this edge, CS must have a hold of at least one SK period.

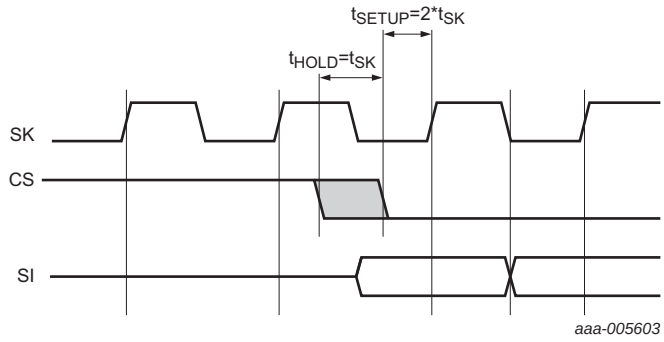


Fig 39. Microwire frame format setup and hold details

11.1 Basic configuration

The CT16B0 counter/timers are configured through the following registers:

- Pins: Configure the CT16B0 pins in the IOCON register block (Table 66).
- Power: In the SYSAHBCLKCTRL register, set bit 7 in Table 19.
- The peripheral clock is determined by the system clock (see Table 18).

11.2 Features

- 16-bit counter/timer with a programmable 16-bit prescaler.
- Counter or timer operation.
- 16-bit capture channels can take a snapshot of the timer value when an input signal transitions. Such a capture event can optionally generate an interrupt.
- 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.
- External outputs controlled by match registers, with the following capabilities:
  - Set LOW on match.
  - Set HIGH on match.
  - Toggle on match.
  - Do nothing on match.
- Up to four match registers can be configured as “PWM”, which allows the use of up to three match outputs as single edge controlled PWM outputs.
- The timer and prescaler can be cleared on a designated capture event. This 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.

11.3 Applications

- Interval timer for counting internal events
- Pulse Width Demodulator via capture input
- Free-running timer
- Pulse Width Modulator via match outputs

11.4 Description

The 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. The counter/timer also includes capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, match registers can be used to provide a single-edge controlled PWM output on the match output pins. It is best to use a match register that is not pinned out to control the PWM cycle length.

The following pages describe 4 MATj outputs and 3 CAPj inputs, which may not all be connected to pins.

11.5 Pin description

Table 137 gives a brief summary of each of the counter/timer related pins.

Table 137. Counter/timer pin description

Pin	Type	Description
CT16B0_CAPj	Input	Capture Signals: A transition on a pin or on-chip signal can be configured to load the corresponding Capture Register with the value in the counter/timer 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 11.6.11.
CT16B0_MATj	Output	External Match Outputs of CT16B0: When a match register (MRj) equals the timer counter (TC), output ‘j’ 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 these outputs.

11.6 Register description

The 16-bit counter/timers contain the registers shown in Table 138. “j” in “MATj” or “CAPj” can be 0 through 3. Address offsets other than those shown are reserved and should not be written. More detailed descriptions follow.

Table 138. Register overview: 16-bit counter/timer CT16B0 (base address 0x4000 C000)

Name	Access	Address offset	Description	Reset value[1]
IR	R/W	0x000	Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending.	0
TCR	R/W	0x004	Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0
TC	R/W	0x008	Timer Counter (TC). The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0
PR	R/W	0x00C	Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0

Table 138. Register overview: 16-bit counter/timer CT16B0 (base address 0x4000 C000) *…continued*

Name	Access	Address offset	Description	Reset value[1]
PC	R/W	0x010	Prescale Counter (PC). 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
MCR	R/W	0x014	Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0
MR0	R/W	0x018	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT16B0_MATj output (if any) when the TC matches MRj.	0
MR1	R/W	0x01C	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT16B0_MATj output (if any) when the TC matches MRj.	0
MR2	R/W	0x020	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT16B0_MATj output (if any) when the TC matches MRj.	0
MR3	R/W	0x024	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT16B0_MATj output (if any) when the TC matches MRj.	0
CCR	R/W	0x028	Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether an interrupt is generated when a capture takes place.	0
CR0	RO	0x02C	Capture Register. Each CRj register can be loaded with the value of TC when there is an event on its CT16B0_CAPj input or internal signal.	0
CR1	RO	0x030	Capture Register. Each CRj register can be loaded with the value of TC when there is an event on its CT16B0_CAPj input or internal signal.	0
CR2	RO	0x034	Capture Register. Each CRj register can be loaded with the value of TC when there is an event on its CT16B0_CAPj input or internal signal.	0
CR3	RO	0x038	Capture Register. Each CRj register can be loaded with the value of TC when there is an event on its CT16B0_CAPj input or internal signal.	0
EMR	R/W	0x03C	External Match Register (EMR). The EMR controls the match function and the external match pins CT16B0_MATj.	0
CTCR	R/W	0x070	Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0
PWMC	R/W	0x074	PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT16B0_MATj.	0

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

11.6.1 Interrupt Registers (IR)

The Interrupt Registers include up to 4 bits for the match interrupts and 4 bits for the capture interrupt. If an interrupt is generated then the corresponding bit in the IR will be 1. Otherwise, the bit will be 0. Writing a 1 to an IR bit clears the bit and thus the interrupt. Writing a zero has no effect.

Table 139. Interrupt Register (IR - address 0x4000 C000) 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
6	CR2INT	Interrupt flag for capture channel 2 event.	0
7	CR3INT	Interrupt flag for capture channel 3 event.	0
31:8	-	Reserved	-

11.6.2 Timer Control Register (TCR)

The Timer Control Register (TCR) is used to control the operation of the counter/timer.

Table 140. Timer Control Register (TCR, address 0x4000 C004) bit description

Bit	Symbol	Value	Description	Reset value
0	C EN		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

11.6.3 Timer Counter registers (TC)

The 16-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count, or when a selected edge occurs on an input signal. Unless it is reset before reaching its upper limit, the TC counts up through 0xFFFF and then wraps back to 0x0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

Table 141: Timer counter registers (TC, address 0x4000 C008) bit description

Bit	Symbol	Description	Reset value
15:0	TC	Timer counter value.	0
31:16	-	Reserved.	-

11.6.4 Prescale Register (PR)

The 16-bit Prescale Register specifies the maximum value for the Prescale Counter.

Table 142: Prescale registers (PR, address 0x4000 C00C) bit description

Bit	Symbol	Description	Reset value
15:0	PCVAL	Prescale value.	0
31:16	-	Reserved.	-

11.6.5 Prescale Counter register (PC)

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, etc.

Table 143: Prescale counter registers (PC, address 0x4000 C010) bit description

Bit	Symbol	Description	Reset value
15:0	PC	Prescale counter value.	0
31:16	-	Reserved.	-

11.6.6 Match Control Register (MCR)

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 144.

Table 144. Match Control Register (MCR, address 0x4000 C014) 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 is reset if MR0 matches it.	0
		1	Enabled	
		0	Disabled	
2	MR0S		Stop on MR0: the TC and PC are stopped and TCR[0] is 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 is reset if MR1 matches it.	0
		1	Enabled	
		0	Disabled	

Table 144. Match Control Register (MCR, address 0x4000 C014) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
5	MR1S		Stop on MR1: the TC and PC are stopped and TCR[0] is 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 is reset if MR2 matches it.	0
		1	Enabled	
		0	Disabled	
8	MR2S		Stop on MR2: the TC and PC are stopped and TCR[0] is 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 is reset if MR3 matches it.	0
		1	Enabled	
		0	Disabled	
11	MR3S		Stop on MR3: the TC and PC are stopped and TCR[0] is 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

11.6.7 Match Registers (MR0/1/2/3)

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 145: Match registers (MR0 to 3, addresses 0x4000 C018 to 24) bit description

Bit	Symbol	Description	Reset value
15:0	MATCH	Timer counter match value.	0
31:16	-	Reserved.	-

11.6.8 Capture Control Register (CCR)

The Capture Control Register is used to control whether each Capture Register is loaded with the value in the Counter/timer when the corresponding 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.

Table 146. Capture Control Register (CCR, address 0x4000 C028) bit description

Bit	Symbol	Value	Description	Reset value
0	CAP0RE		Capture on CT16Bi_CAP0 rising edge: a sequence of 0 then 1 on CT16Bi_CAP0 causes CR0 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
1	CAP0FE		Capture on CT16Bi_CAP0 falling edge: a sequence of 1 then 0 on CT16Bi_CAP0 causes CR0 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
2	CAP0I		Interrupt on CT16Bi_CAP0 event: a CR0 load due to a CT16Bi_CAP0 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
3	CAP1RE		Capture on CT16Bi_CAP1 rising edge: a sequence of 0 then 1 on CT16Bi_CAP1 causes CR1 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
4	CAP1FE		Capture on CT16Bi_CAP1 falling edge: a sequence of 1 then 0 on CT16Bi_CAP1 causes CR1 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
5	CAP1I		Interrupt on CT16Bi_CAP1 event: a CR1 load due to a CT16Bi_CAP1 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
6	CAP2RE		Capture on CT16Bi_CAP2 rising edge: a sequence of 0 then 1 on CT16Bi_CAP2 causes CR2 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
7	CAP2FE		Capture on CT16Bi_CAP2 falling edge: a sequence of 1 then 0 on CT16Bi_CAP2 causes CR2 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
8	CAP2I		Interrupt on CT16Bi_CAP2 event: a CR2 load due to a CT16Bi_CAP2 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
9	CAP3RE		Capture on comparator output rising edge: a sequence of 0 then 1 on comparator output causes CR3 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	

Table 146. Capture Control Register (CCR, address 0x4000 C028) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
10	CAP3FE		Capture on comparator output falling edge: a sequence of 1 then 0 on comparator output causes CR3 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
11	CAP3I		Interrupt on comparator output event: a CR3 load due to a comparator output event generates an interrupt.	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

11.6.9 Capture Registers (CR0/1/2/3)

Each Capture register is associated with a signal and may be loaded with the Timer Counter value when a specified event occurs on that signal. 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 147. Capture registers (CR0 to 3, addresses 0x4000 C02C to 0x4000 C034) bit description

Bit	Symbol	Description	Reset value
15:0	CAP	Timer counter capture value.	0
31:16	-	Reserved.	-

11.6.10 External Match Register (EMR)

The External Match Register provides both control and status of the external match channels and external match pins CT16B0_MATj.

If the match outputs are configured as PWM output in the PWMCON registers (Section 11.6.12), the function of the external match registers is determined by the PWM rules (Section 11.6.13 “Rules for single edge controlled PWMs” on page 164).

Table 148. External Match Register (EMR, address 0x4000 C03C) bit description

Bit	Symbol	Value	Description	Reset value
0	EM0		External Match 0. This bit reflects the state of output CT16Bi_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 CT16Bi_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 CT16Bi_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 CT16Bi_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 match channel 2, 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. Note that on counter/timer 0 this match channel is not pinned out. This bit is driven to the CT16Bi_MAT2 pin 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 CT16Bi_MAT3, whether or not this output is connected to a 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 CT16Bi_MAT3 pin 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 (CT16B0_MATj pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16B0_MATj 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 (CT16B0_MATj pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16B0_MATj 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 (CT16B0_MATj pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16B0_MATj pin is HIGH if pinned out).	
		0x3	Toggle the corresponding External Match bit/output.	

Table 148. External Match Register (EMR, address 0x4000 C03C) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
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 (CT16B0_MATj pin is LOW if pinned out).	
		0x2	Set the corresponding External Match bit/output to 1 (CT16B0_MATj 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 149. 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 (CT16B0_MATj pin is LOW if pinned out).
10	Set the corresponding External Match bit/output to 1 (CT16B0_MATj pin is HIGH if pinned out).
11	Toggle the corresponding External Match bit/output.

11.6.11 Count Control Register (CTCR)

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the signal and edge(s) for counting.

When Counter Mode is selected in this register, the signal selected by bits 3:2 is sampled on every rising edge of the PCLK clock. By comparing two consecutive samples of this signal, rising and falling edges on the signal are detected. The Timer Counter register is incremented by an edge selected by bits 1:0 of this register.

Since the selected clock signal is sampled by PCLK to identify edges on the selected clock signal, the high and low pulse width on the clock signal must not be shorter than the PCLK period.

Bits 7:4 of this register control the capture-clears-timer feature. This feature allows a selected edge on a particular CAP input to reset the timer and prescaler to zero. Thus the timer can be cleared on the leading edge of an input pulse and captured on the trailing edge. The captured value then directly reflects the pulse width, without the need to perform a subtraction operation in software.

Table 150. Count Control Register (CTCR, address 0x4000 C070) bit description

Bit	Symbol	Value	Description	Reset value
1:0	CTM		Counter/Timer Mode. This field selects whether the timer/counter is clocked by PCLK or by another signal.	00
		0x0	Timer Mode: PC is incremented on every rising PCLK edge	
		0x1	Counter Mode: the TC is incremented on rising edges on the signal selected by bits 3:2.	
		0x2	Counter Mode: the TC is incremented on falling edges on the signal selected by bits 3:2.	
		0x3	Counter Mode: the TC is incremented on both edges on the signal 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 signal increments the counter:	00
		0x0	CT16Bi_CAP0 (bits 2:0 in the Capture Control Register (CCR) must be 000)	
		0x1	CT16Bi_CAP1 (bits 5:3 in the CCR must be 000)	
		0x2	CT16Bi_CAP2 (bits 8:6 in the CCR must be 000)	
		0x3	Analog comparator output (bits 11:9 in the CCR must be 000)	
4	EnCC		Enable Capture-Clear. A 1 in this bit enables clearing of the timer and prescaler when the event specified in bits 7:5 occurs.	0
7:5	SelCC		Capture-Clear Select. When bit 4 is 1, these bits select which capture input edge clears the timer and prescaler.	000
		0x0	A rising edge on CT16Bi_CAP0	
		0x1	A falling edge on CT16Bi_CAP0	
		0x2	A rising edge on CT16Bi_CAP1	
		0x3	A falling edge on CT16Bi_CAP1	
		0x4	A rising edge on CT16Bi_CAP2	
		0x5	A falling edge on CT16Bi_CAP2	
		0x6	A rising edge on the analog comparator output	
		0x7	A falling edge on the analog comparator output	
31:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

11.6.12 PWM Control register (PWMC)

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).

One match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set 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, match outputs configured as PWM outputs go LOW.

Table 151. PWM Control Register (PWMC, address 0x4000 C074) 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_MAT1 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 EM03.	
		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.	NA

11.6.13 Rules for single edge controlled PWMs

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 goes HIGH when its match value is reached. If no match occurs (that is, 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 is cleared at 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 is reset to LOW on the next clock tick. Therefore, the PWM output always consists of a one-clock-wide positive pulse with a period determined by the PWM cycle length (that is, the timer reload value).
5. If a match register is set to zero, then the PWM output goes to HIGH the first time the timer returns to zero and stays HIGH continuously.

Note: When the match outputs are selected to serve as PWM outputs, the timer reset (MRiR) and timer stop (MRiS) bits in the Match Control Register MCR must be cleared to 0 except for the match register that controls the PWM cycle length. For this register, set the MRiR bit to 1 to enable the timer reset when the timer value matches the value of the corresponding match register.

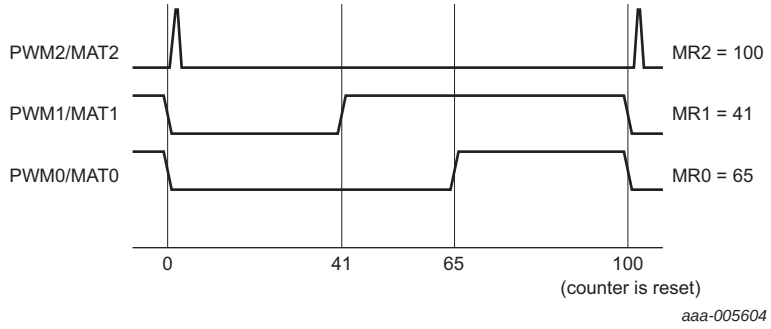


Fig 40. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.

11.7 Example timer operation

Figure 41 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 42 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.

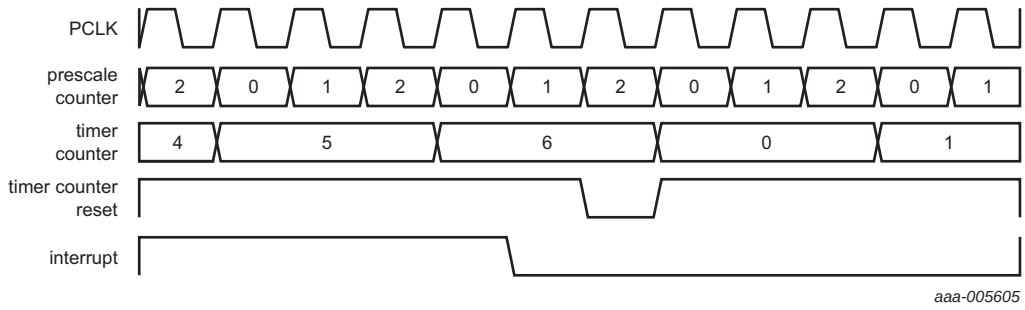


Fig 41. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled

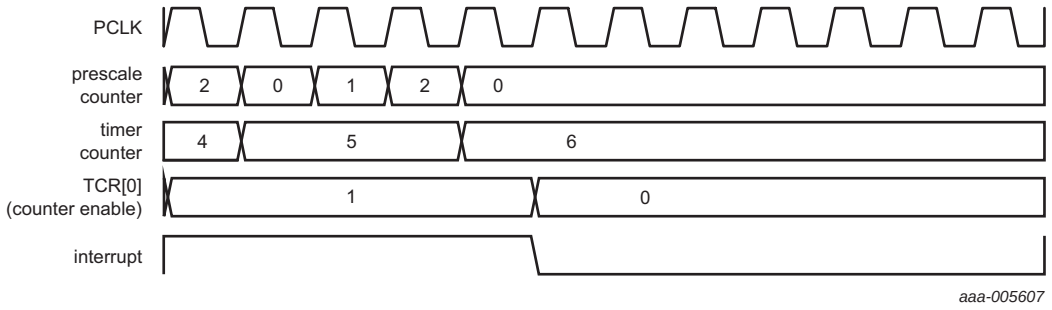


Fig 42. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled

11.8 Architecture

The 16-bit counter/timer block diagram is shown in Figure 43.

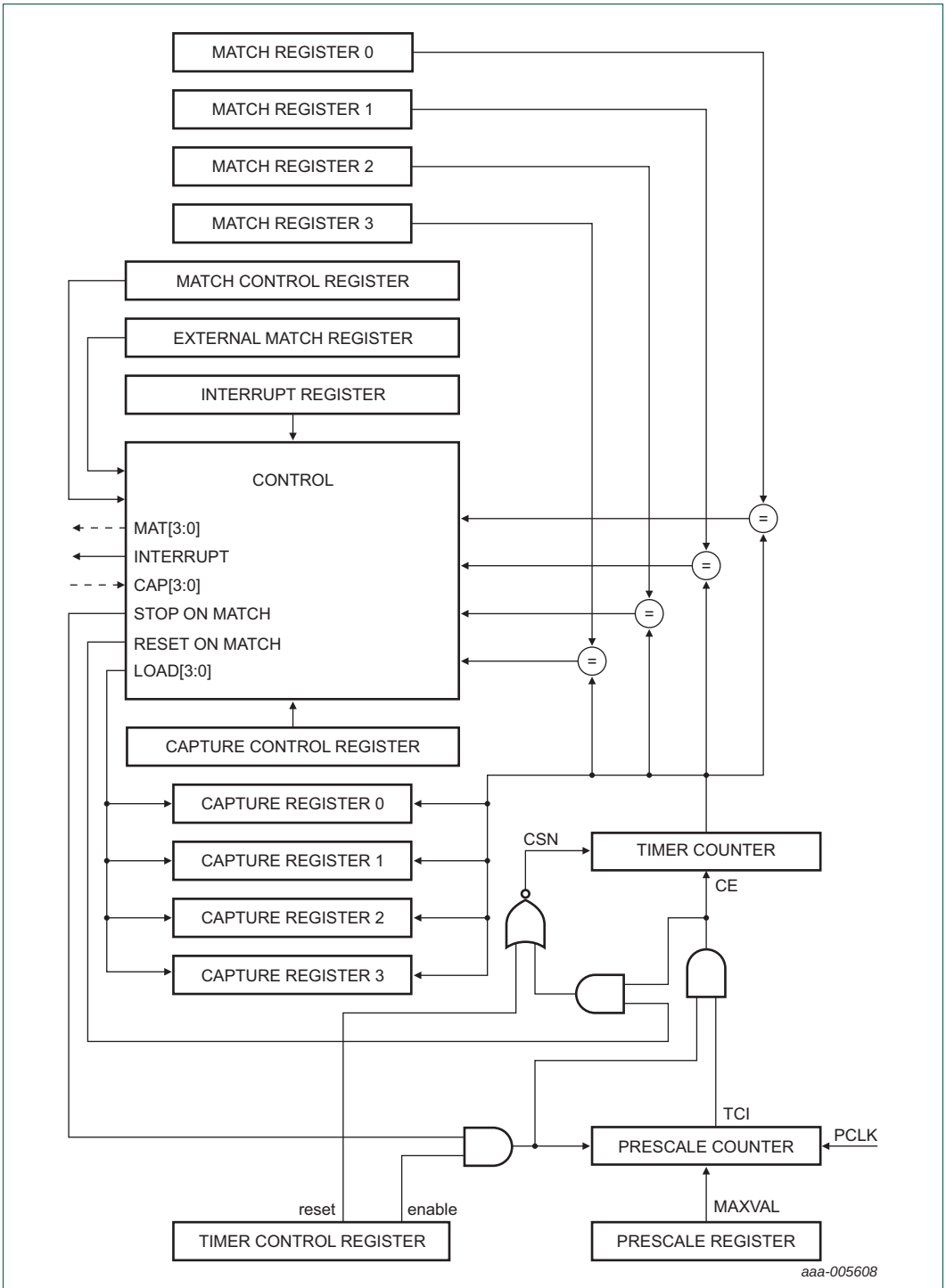


Fig 43. 16-bit counter/timer block diagram

12.1 Basic configuration

The CT32B0 counter/timers are configured through the following registers:

- Pins: Configure the CT32B0 pins in the IOCON register block (Table 66).
- Power: In the SYSAHBCLKCTRL register, set bit 9 in Table 19.
- The peripheral clock is determined by the system clock (see Table 18).

12.2 Features

- 32-bit counter/timer with a programmable 32-bit prescaler.
- Counter or Timer operation.
- 32-bit capture channels can take a snapshot of the timer value when an input signal transitions. Such a capture event can optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- External outputs controlled by match registers, with the following capabilities:
  - Set LOW on match.
  - Set HIGH on match.
  - Toggle on match.
  - Do nothing on match.
- Up to four match registers can be configured as “PWM”, which allows the use of up to three match outputs as single edge controlled PWM outputs.
- The timer and prescaler can be cleared on a designated capture event. This 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.

12.3 Applications

- Interval timer for counting internal events
- Pulse Width Demodulator via capture input
- Free running timer
- Pulse Width Modulator via match outputs

12.4 Description

The 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. The counter/timer also includes one capture input to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, match registers can be used to provide a single-edge controlled PWM output on the match output pins. It is best to use a match register that is not pinned out to control the PWM cycle length.

The following pages describe 4 MATj outputs and 3 CAPj inputs, which may not all be connected to pins.

12.5 Pin description

Table 152 gives a brief summary of each of the counter/timer related pins.

Table 152. Counter/timer pin description

Pin	Type	Description
CT32B0_CAPj	Input	Capture Signals: A transition on a pin or on-chip signal can be configured to load the corresponding Capture Registers with the value in the counter/timer 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 12.6.11 “Count Control Register (CTCR)” on page 177.
CT32B0_MATj	Output	External Match Output: When a match register (MRj) equals the timer counter (TC), a MATj pin 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 these outputs.

12.6 Register description

The 32-bit counter/timers contain the registers shown in Table 153. “j” in “MATj” or “CAPj” can be 0 through 3. Address offsets other than those shown are reserved and should not be written. More detailed descriptions follow.

Table 153. Register overview: 32-bit counter/timer CT32B0 (base address 0x4001 4000)

Name	Access	Address offset	Description	Reset value[1]
IR	R/W	0x000	Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending.	0
TCR	R/W	0x004	Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0
TC	R/W	0x008	Timer Counter (TC). The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0

Table 153. Register overview: 32-bit counter/timer CT32B0 (base address 0x4001 4000) *…continued*

Name	Access	Address offset	Description	Reset value[1]
PR	R/W	0x00C	Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	0
PC	R/W	0x010	Prescale Counter (PC). 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
MCR	R/W	0x014	Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0
MR0	R/W	0x018	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT32B0_MATj output (if any) when the TC matches MRj.	0
MR1	R/W	0x01C	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT32B0_MATj output (if any) when the TC matches MRj.	0
MR2	R/W	0x020	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT32B0_MATj output (if any) when the TC matches MRj.	0
MR3	R/W	0x024	Match Register. Each match can be enabled in the MCR to reset the TC, stop both the TC and PC, generate an interrupt, and/or switch a CT32B0_MATj output (if any) when the TC matches MRj.	0
CCR	R/W	0x028	Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether an interrupt is generated when a capture takes place.	0
CR0	RO	0x02C	Capture Registers. Each CRj register can be loaded with the value of TC when there is an event on its CT32B0_CAPj input or internal signal.	0
CR1	RO	0x030	Capture Registers. Each CRj register can be loaded with the value of TC when there is an event on its CT32B0_CAPj input or internal signal.	0
CR2	RO	0x034	Capture Registers. Each CRj register can be loaded with the value of TC when there is an event on its CT32B0_CAPj input or internal signal.	0
CR3	RO	0x038	Capture Registers. Each CRj register can be loaded with the value of TC when there is an event on its CT32B0_CAPj input or internal signal.	0
EMR	R/W	0x03C	External Match Register (EMR). The EMR controls the match function and the external match pins CT32B0_MATj.	0
CTCR	R/W	0x070	Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0
PWMC	R/W	0x074	PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT32B0_MATj.	0

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

12.6.1 Interrupt Registers (IR)

The Interrupt Registers include up to 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 1. Otherwise, the bit will be 0. Writing a 1 to an IR bit clears the bit and thus the interrupt. Writing a zero has no effect.

Table 154: 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
6	CR2INT	Interrupt flag for capture channel 2 event.	0
7	CR3INT	Interrupt flag for capture channel 3 event.	0
31:8	-	Reserved	-

12.6.2 Timer Control Register (TCR)

The Timer Control Register (TCR) is used to control the operation of the counter/timer.

Table 155. Timer Control Register (TCR, address 0x4001 4004 (CT32B0)) 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

12.6.3 Timer Counter (TC)

The 32-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count, or when a selected edge occurs on an input signal. Unless it is reset before reaching its upper limit, the TC counts up through 0xFFFF FFFF and then wraps back to 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

Table 156: Timer counter registers (TC, address 0x4001 4008 (CT32B0)) bit description

Bit	Symbol	Description	Reset value
31:0	TC	Timer counter value.	0

12.6.4 Prescale Register (PR)

The 32-bit Prescale Register specifies the maximum value for the Prescale Counter.

Table 157: Prescale registers (PR, address 0x4001 400C (CT32B0)) bit description

Bit	Symbol	Description	Reset value
31:0	PCVAL	Prescale value.	0

12.6.5 Prescale Counter Register (PC)

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 158: Prescale registers (PC, address 0x4001 4010 (CT32B0)) bit description

Bit	Symbol	Description	Reset value
31:0	PC	Prescale counter value.	0

12.6.6 Match Control Register (MCR)

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 159.

Table 159: Match Control Register (MCR, address 0x4001 4014 (CT32B0)) 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 is reset if MR0 matches it.	0
		1	Enabled	
		0	Disabled	
2	MR0S		Stop on MR0: the TC and PC are stopped and TCR[0] is 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 is reset if MR1 matches it.	0
		1	Enabled	
		0	Disabled	
5	MR1S		Stop on MR1: the TC and PC are stopped and TCR[0] is set to 0 if MR1 matches the TC.	0
		1	Enabled	
		0	Disabled	

Table 159: Match Control Register (MCR, address 0x4001 4014 (CT32B0)) bit description

Bit	Symbol	Value	Description	Reset value
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 is reset if MR2 matches it.	0
		1	Enabled	
		0	Disabled	
8	MR2S		Stop on MR2: the TC and PC are stopped and TCR[0] is 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 is reset if MR3 matches it.	0
		1	Enabled	
		0	Disabled	
11	MR3S		Stop on MR3: the TC and PC are stopped and TCR[0] is 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

12.6.7 Match Registers (MR0/1/2/3)

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 160: Match registers (MR0 to 3, addresses 0x4001 4018 to 24 (CT32B0)) bit description

Bit	Symbol	Description	Reset value
31:0	MATCH	Timer counter match value.	0

12.6.8 Capture Control Register (CCR)

The Capture Control Register is used to control whether each Capture Register is loaded with the value in the Timer Counter when the corresponding 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.

Table 161. Capture Control Register (CCR, address 0x4001 4028 (CT32B0)) bit description

Bit	Symbol	Value	Description	Reset value
0	CAP0RE		Capture on CT32Bi_CAP0 rising edge: a sequence of 0 then 1 on CT32Bi_CAP0 causes CR0 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
1	CAP0FE		Capture on CT32Bi_CAP0 falling edge: a sequence of 1 then 0 on CT32Bi_CAP0 causes CR0 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
2	CAP0I		Interrupt on CT32Bi_CAP0 event: a CR0 load due to a CT32Bi_CAP0 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
3	CAP1RE		Capture on CT32Bi_CAP1 rising edge: a sequence of 0 then 1 on CT32Bi_CAP1 causes CR1 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
4	CAP1FE		Capture on CT32Bi_CAP1 falling edge: a sequence of 1 then 0 on CT32Bi_CAP1 causes CR1 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
5	CAP1I		Interrupt on CT32Bi_CAP1 event: a CR1 load due to a CT32Bi_CAP1 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
6	CAP2RE		Capture on CT32Bi_CAP2 rising edge: a sequence of 0 then 1 on CT32Bi_CAP2 causes CR2 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
7	CAP2FE		Capture on CT32Bi_CAP2 falling edge: a sequence of 1 then 0 on CT32Bi_CAP2 causes CR2 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
8	CAP2I		Interrupt on CT32Bi_CAP2 event: a CR2 load due to a CT32Bi_CAP2 event generates an interrupt.	0
		1	Enabled	
		0	Disabled	
9	CAP3RE		Capture on comparator output rising edge: a sequence of 0 then 1 on comparator output causes CR3 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	

Table 161. Capture Control Register (CCR, address 0x4001 4028 (CT32B0)) bit description

Bit	Symbol	Value	Description	Reset value
10	CAP3FE		Capture on comparator output falling edge: a sequence of 1 then 0 on comparator output causes CR3 to be loaded with the contents of TC.	0
		1	Enabled	
		0	Disabled	
11	CAP3I		Interrupt on comparator output event: a CR3 load due to a comparator output event generates an interrupt.	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

12.6.9 Capture Registers (CR)

Each Capture register is associated with a signal and may be loaded with the Timer Counter value when a specified event occurs on that signal. 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 162: Capture registers (CR0 to 3, addresses 0x4001 402C to 0x4001 4038 (CT16B0)) bit description

Bit	Symbol	Description	Reset value
31:0	CAP	Timer counter capture value.	0

12.6.10 External Match Register (EMR)

The External Match Register provides both control and status of the external match channels and external match pins CT32B0_MATj.

If the match outputs are configured as PWM output in the PWMCON registers (Section 12.6.12), the function of the external match registers is determined by the PWM rules (Section 12.6.13 “Rules for single edge controlled PWMs” on page 179).

Table 163: External Match Register (EMR, address 0x4001 403C (CT32B0)) bit description

Bit	Symbol	Value	Description	Reset value
0	EM0		External Match 0. This bit reflects the state of output CT32Bi_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 CT32Bi_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 CT32Bi_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 CT32Bi_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 CT32Bi_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 CT32Bi_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 CT32Bi_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 CT32Bi_MAT3i 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).	
		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.	

Table 163: External Match Register (EMR, address 0x4001 403C (CT32B0)) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
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 164. External match control

EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4]	Function
0x0	Do Nothing.
0x1	Clear the corresponding External Match bit/output to 0 (CT32B0_MATj pin is LOW if pinned out).
0x2	Set the corresponding External Match bit/output to 1 (CT32B0_MATj pin is HIGH if pinned out).
0x3	Toggle the corresponding External Match bit/output.

12.6.11 Count Control Register (CTCR)

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the signal and edge(s) for counting.

When Counter Mode is selected in this register, the signal selected by bits 3:2 is sampled on every rising edge of the PCLK clock. By comparing two consecutive samples of this signal, rising and falling edges on the signal are detected. The Timer Counter register is incremented by an edge selected by bits 1:0 of this register.

Since the selected clock signal is sampled by PCLK to identify edges on the selected clock signal, the high and low pulse width on the clock signal must not be shorter than the PCLK period.

Bits 7:4 of this register control the capture-clears-timer feature. This feature allows a selected edge on a particular CAP input to reset the timer and prescaler to zero. Thus the timer can be cleared on the leading edge of an input pulse and captured on the trailing edge. The captured value then directly reflects the pulse width, without the need to perform a subtraction operation in software.

Table 165: Count Control Register (CTCR, address 0x4001 4070 (CT32B0)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	CTM		Counter/Timer Mode. This field selects whether the timer/counter is clocked by PCLK or by another signal.	00
		0x0	Timer Mode: PC is incremented on every rising PCLK edge	
		0x1	Counter Mode: the TC is incremented on rising edges on the signal selected by bits 3:2.	
		0x2	Counter Mode: the TC is incremented on falling edges on the signal selected by bits 3:2.	
		0x3	Counter Mode: the TC is incremented on both edges on the signal selected by bits 3:2.	
3:2	CIS		In counter mode (when bits 1:0 in this register are not 00), these bits select which signal increments the counter:	00
		0x0	CT32Bi_CAP0 (bits 2:0 in the Capture Control Register (CCR) must be 000)	
		0x1	CT32Bi_CAP1 (bits 5:3 in the CCR must be 000)	
		0x2	CT32Bi_CAP2 (bits 8:6 in the CCR must be 000)	
		0x3	Analog comparator output (bits 11:9 in the CCR must be 000)	
4	ENCC		Enable Capture-Clear. A 1 in this bit enables clearing of the timer and prescaler when the event specified in bits 7:5 occurs.	0
7:5	SELCC		Capture-Clear Select. When bit 4 is 1, these bits select which capture input edge clears the timer and prescaler.	000
		0x0	A rising edge on CT32Bi_CAP0	
		0x1	A falling edge on CT32Bi_CAP0	
		0x2	A rising edge on CT32Bi_CAP1	
		0x3	A falling edge on CT32Bi_CAP1	
		0x7	A rising edge on CT32Bi_CAP2	
		0x5	A falling edge on CT32Bi_CAP2	
		0x6	A rising edge on the analog comparator output	
		0x7	A falling edge on the analog comparator output	
31:8	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.6.12 PWM Control Register (PWMC)

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).

One match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set 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, match outputs configured as PWM outputs go LOW.

Table 166. PWM Control Register (PWMC, 0x4001 4074 (CT32B0)) 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_MAT0 is controlled by EM1.	
		1	PWM mode is enabled for CT32Bi_MAT1.	
2	PWMEN2		PWM mode enable for channel2.	0
		0	CT32Bi_MAT0 is controlled by EM2.	
		1	PWM mode is enabled for CT32Bi_MAT2.	
3	PWMEN3		PWM mode enable for channel3.	0
		0	CT32Bi_MAT0 is controlled by EM03.	
		1	PWM mode is enabled for CT32Bi_MAT3.	
31:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

12.6.13 Rules for single edge controlled PWMs

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 goes HIGH when its match value is reached. If no match occurs (that is, 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 is cleared at 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 is reset to LOW on the next clock tick. Therefore, the PWM output always consists of a one-clock-wide positive pulse with a period determined by the PWM cycle length (that is, the timer reload value).
5. If a match register is set to zero, then the PWM output goes to HIGH the first time the timer goes back to zero and stays HIGH continuously.

Note: When the match outputs are selected to function as PWM outputs, the timer reset (MRiR) and timer stop (MRiS) bits in the Match Control Register MCR must be cleared to 0 except for the match register that controls the PWM cycle length. For this register, set the MRiR bit to 1 to enable the timer reset when the timer value matches the value of the corresponding match register.

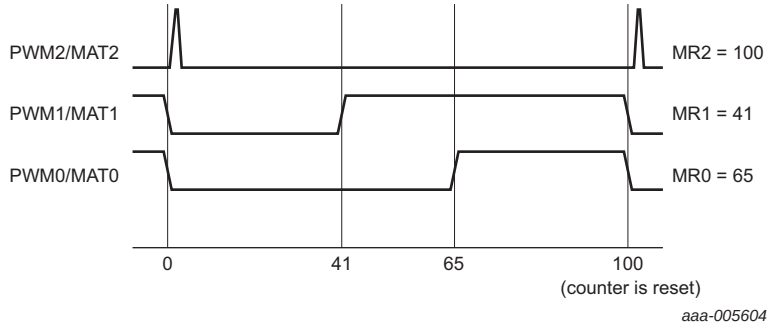


Fig 44. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.

12.7 Example timer operation

Figure 45 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 46 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.

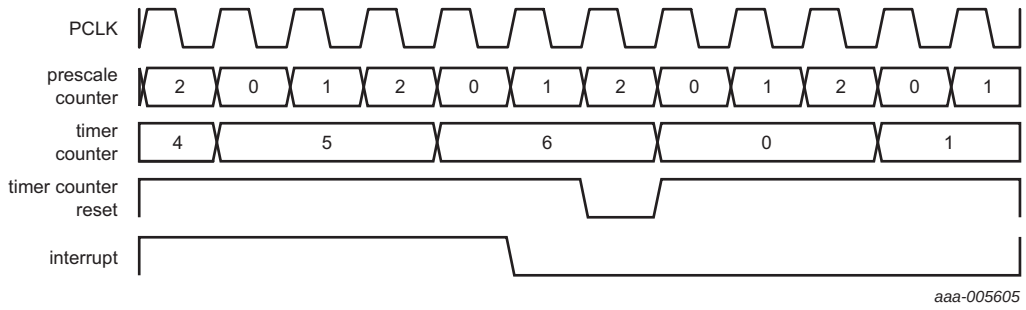


Fig 45. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled

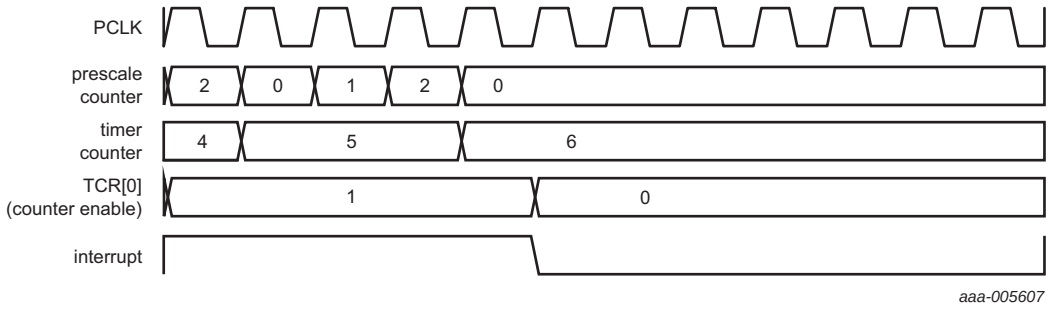


Fig 46. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled

12.8 Architecture

The 32-bit counter/timer block diagram is shown in Figure 47.

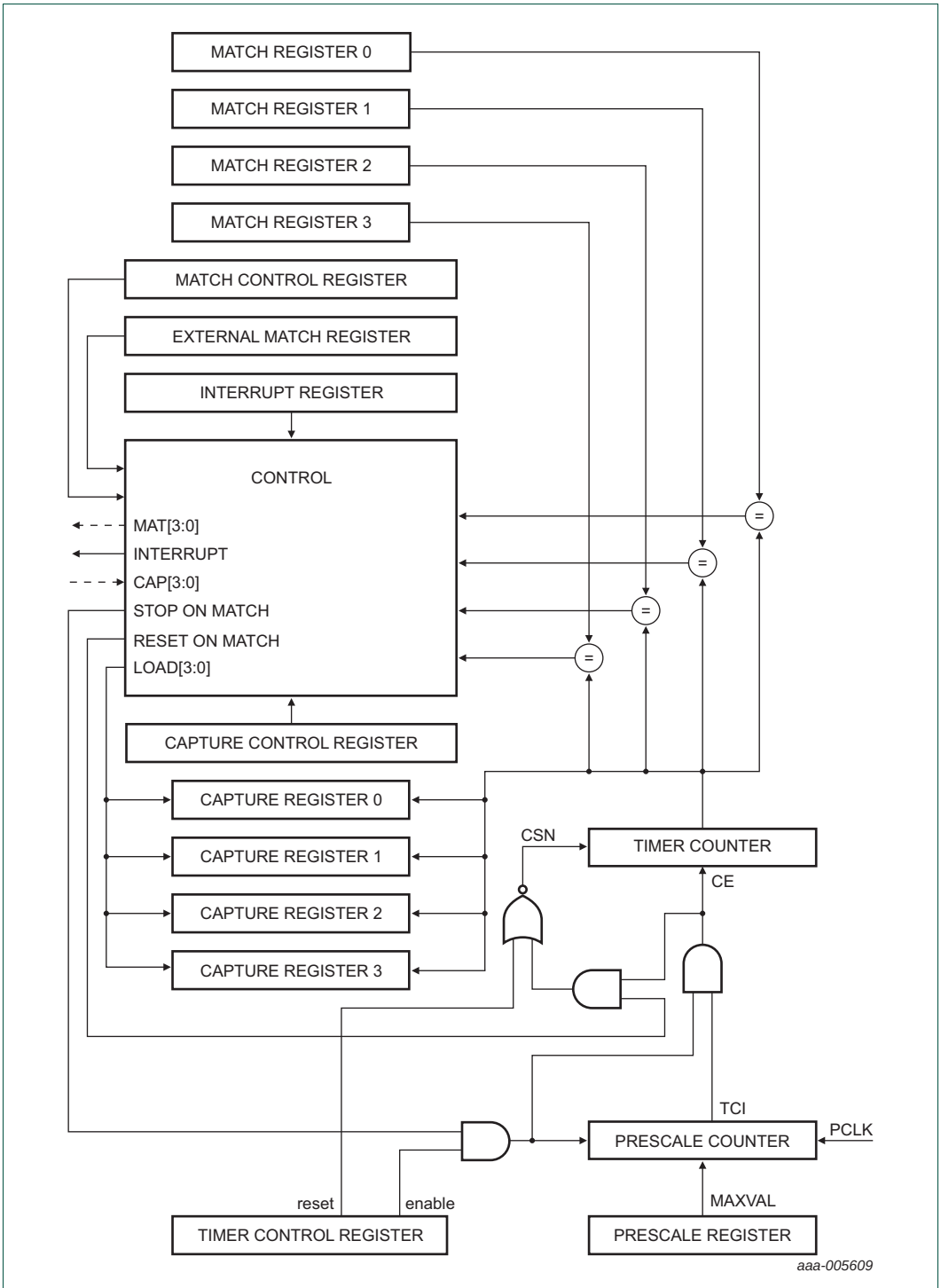


Fig 47. 32-bit counter/timer block diagram

13.1 How to read this chapter

The system tick timer (SysTick timer) is part of the ARM Cortex-M0 core and is identical for all EM783 parts.

13.2 Features

- Time intervals of 10 milliseconds.
- Uses dedicated exception vector.
- Clocked internally by a dedicated system tick timer clock.

13.3 Description

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 all Cortex-M0 based devices.

Refer to the *Cortex-M0 User Guide* for details.

13.4 Operation

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. In order to generate recurring interrupts at a specific interval, the STRELOAD register must be initialized with the correct value for the desired interval. A default value is provided in the STCALIB register and may be changed by software. The default value gives a 10 millisecond interrupt rate if the CPU clock is set appropriately.

The block diagram of the SysTick timer is shown in Figure 48.

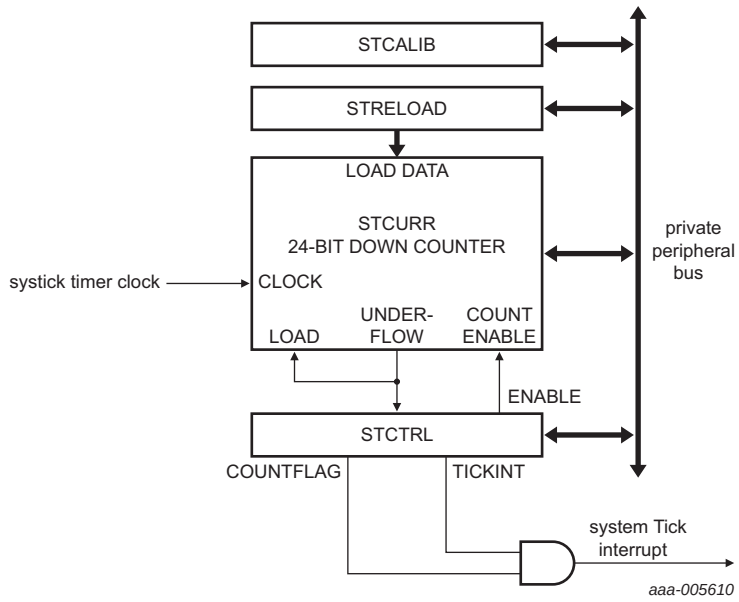


Fig 48. System tick timer block diagram

13.5 Register description

Table 167. SysTick timer register map (base address 0xE000 E000)

Name	Access	Address offset	Description	Reset value[1]
STCTRL	R/W	0x010	System Timer Control and status register	0x4
STRELOAD	R/W	0x014	System Timer Reload value register	0
STCURR	R/W	0x018	System Timer Current value register	0
STCALIB	R/W	0x01C	System Timer Calibration value register	0x4

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

13.5.1 System Timer Control and status register (STCTRL - 0xE000 E010)

The STCTRL register contains control information for the SysTick timer, and provides a status flag.

Table 168. System Timer Control and status register (STCTRL - 0xE000 E010) bit description

Bit	Symbol	Description	Reset value
0	ENABLE	System Tick counter enable. When 1, the counter is enabled. When 0, the counter is disabled.	0
1	TICKINT	System Tick interrupt enable. When 1, the System Tick interrupt is enabled. When 0, the System Tick interrupt is disabled. When enabled, the interrupt is generated when the System Tick counter counts down to 0.	0
2	-	Reserved	1

Table 168. System Timer Control and status register (STCTRL - 0xE000 E010) bit description *…continued*

Bit	Symbol	Description	Reset value
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	System Tick counter flag. This flag is set when the System Tick counter counts down to 0, and is cleared by reading this register.	0
31:17	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.5.2 System Timer Reload value register (STRELOAD - 0xE000 E014)

The STRELOAD 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 STCALIB register may be read and used as the value for STRELOAD if the CPU or external clock is running at the frequency intended for use with the STCALIB value.

Table 169. System Timer Reload value register (STRELOAD - 0xE000 E014) bit description

Bit	Symbol	Description	Reset value
23:0	RELOAD	This is the value that is loaded into the System Tick counter when it counts down to 0.	0
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

13.5.3 System Timer Current value register (STCURR - 0xE000 E018)

The STCURR register returns the current count from the System Tick counter when it is read by software.

Table 170. System Timer Current value register (STCURR - 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

13.5.4 System Timer Calibration value register (STCALIB - 0xE000 E01C)

For details, see Table 300.

Table 171. System Timer Calibration value register (STCALIB - 0xE000 E01C) bit description

Bit	Symbol	Value	Description	Reset value
23:0	TENMS		System tick timer calibration value	0x04
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 300.	0
31	NOREF		See Table 300.	0

14.1 How to read this chapter

The WWDT is identical for all parts.

14.2 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 before 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.
- Debug mode.

14.3 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 is 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, the 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 generates a watchdog event, allowing for system recovery.

14.4 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 WDTC register.
- Set the Watchdog timer operating mode in the WDMOD register.
- Set a value for the watchdog window time in the WDWINDOW register if windowed operation is desired.
- Set a value for the watchdog warning interrupt in the WDWARNINT register if a warning interrupt is desired.
- Enable the Watchdog by writing 0xAA followed by 0x55 to the WDFEED register.
- The Watchdog must be fed again before the Watchdog counter reaches zero in order to prevent a watchdog event. If a window value is programmed, the feed must also occur after the watchdog counter passes that value.

When the Watchdog Timer is configured so that a watchdog event causes a reset and the counter reaches zero, the CPU is 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 occurs when the counter matches the value defined by the WDWARNINT register.

14.5 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 3). The WDCLK is used for the watchdog timer counting and is derived from the wdt_clk in Figure 3. Either the IRC or the watchdog oscillator can be used as wdt_clk.

There is some synchronization logic between these two clock domains. When the WDMOD and WDTC registers are updated by APB operations, the new value takes effect in 3 WDCLK cycles on the logic in the WDCLK clock domain. When the watchdog timer is counting on WDCLK, the synchronization logic first locks the value of the counter on WDCLK and then synchronizes it with the PCLK for reading as the WDTV register by the CPU.

The watchdog oscillator can be powered down in the PDRUNCFG register (Section 3.4.26) if it is not used. The clock to the watchdog register block (PCLK) can be disabled in the SYSAHBCLKCTRL register (Section 3.4.15) for power savings.

14.6 Register description

The Watchdog Timer contains the registers shown in Table 172.

Table 172. Register overview: Watchdog timer (base address 0x4000 4000)

Name	Access	Address offset	Description	Reset Value[1]
WDMOD	R/W	0x000	Watchdog mode register. This register contains the basic mode and status of the Watchdog Timer.	0
WDTC	R/W	0x004	Watchdog timer constant register. This 24-bit register determines the time-out value.	0xFF
WDFEED	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
WDTV	RO	0x00C	Watchdog timer value register. This 24-bit register reads out the current value of the Watchdog timer.	0xFF
WDCLKSEL	R/W	0x010	Watchdog clock select register.	0
WDWARNINT	R/W	0x014	Watchdog Warning Interrupt compare value.	0
WDWINDOW	R/W	0x018	Watchdog Window compare value.	0xFF FFFF

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

14.6.1 Watchdog Mode register (WDMOD)

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 173. Watchdog Mode register (WDMOD - 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 time-out does not cause a chip reset.	
		1	A watchdog time-out causes 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 173. Watchdog Mode register (WDMOD - 0x4000 4000) bit description

Bit	Symbol	Value	Description	Reset Value
4	WDPROTECT		Watchdog update mode. Once this bit has been written with a 1, it cannot be rewritten with a 0.	0
		0	The watchdog reload value (WDTC) can be changed at any time.	
		1	The watchdog reload value (WDTC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW.	
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 that bit to a clock source that is disabled or powered down.	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 cannot be cleared by software. Both flags are cleared by an external reset or a Watchdog timer reset.

WDTOF The Watchdog time-out flag is set when the Watchdog times out, when a feed error occurs, or when WDPROTECT =1 and an attempt is made to write to the WDTC register. This flag is cleared by software writing a 0 to this bit.

WDINT The Watchdog interrupt flag is set when the Watchdog counter reaches the value specified by WDWARNINT. This flag is cleared when any reset occurs, and is cleared by software by writing a 0 to this bit.

A Watchdog reset or interrupt can occur when the watchdog is running and has an operating clock source. Any clock source works in sleep mode.

Table 174. 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 is generated but watchdog does not reset. When this mode is selected, the watchdog counter reaching the value specified by WDWARNINT sets the WDINT flag and the Watchdog interrupt request is 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 sets the WDINT flag, the Watchdog interrupt request is generated, and the watchdog counter reaching zero resets the microcontroller. A watchdog feed before reaching the value of WDWINDOW also causes a watchdog reset.

14.6.2 Watchdog Timer Constant register (WDTC)

The WDTC register determines the time-out value. Every time a feed sequence occurs, the value in the WDTC is loaded into the Watchdog timer. The WDTC resets to 0x00 00FF. Writing a value below 0xFF causes 0x00 00FF to be loaded into the WDTC. Thus the minimum time-out interval is $T_{WDCLK} \times 256 \times 4$.

If the WDPROTECT bit in WDMOD = 1, an attempt to change the value of WDTC before the watchdog counter is below the values of WDWARNINT and WDWINDOW causes a watchdog reset and set the WDTOF flag.

Table 175. Watchdog Timer Constant register (WDTC - 0x4000 4004) bit description

Bit	Symbol	Description	Reset Value
23:0	Count	Watchdog time-out interval.	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

14.6.3 Watchdog Feed register (WDFEED)

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 is generated during the second PCLK following an incorrect access to a Watchdog register during a feed sequence.

It is good practice 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 176. Watchdog Feed register (WDFEED - 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

14.6.4 Watchdog Timer Value register (WDTV)

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 read by the CPU.

Table 177. Watchdog Timer Value register (WDTV - 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

14.6.5 Watchdog Clock Select register (WDCLKSEL)

Table 178. Watchdog Clock Select register (WDCLKSEL - 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 1, writing to this register does not affect the CLKSEL bit, and therefore the clock source cannot be changed. Once the LOCK bit is set, it cannot be cleared.	0

14.6.6 Watchdog Timer Warning Interrupt register (WDWARNINT)

The WDWARNINT register determines the watchdog timer counter value that generates a watchdog interrupt. When the watchdog timer counter matches the value defined by WDWARNINT, an interrupt is 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 occurs at the same time as the watchdog event.

Table 179. Watchdog Timer Warning Interrupt register (WDWARNINT - 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

14.6.7 Watchdog Timer Window register (WDWINDOW)

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 WDTV value, so windowing is not in effect.

Table 180. Watchdog Timer Window register (WDWINDOW - 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

14.7 Block diagram

The block diagram of the Watchdog is shown below in the Figure 49. The synchronization logic (PCLK - WDCLK) is not shown in the block diagram.

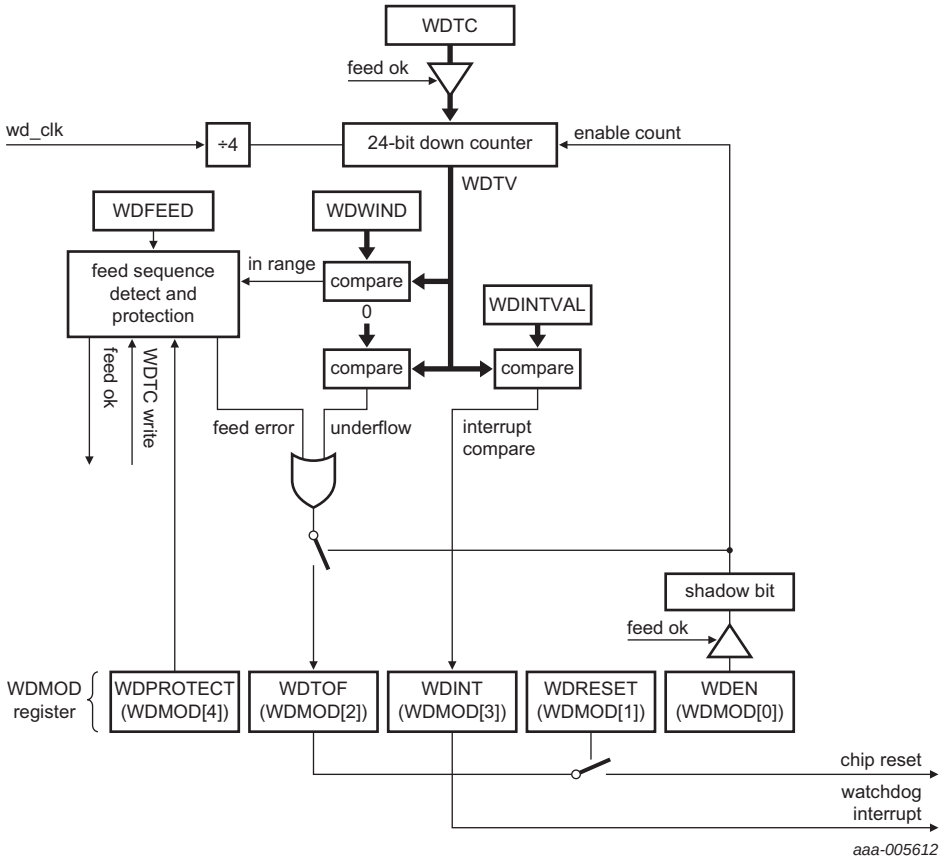
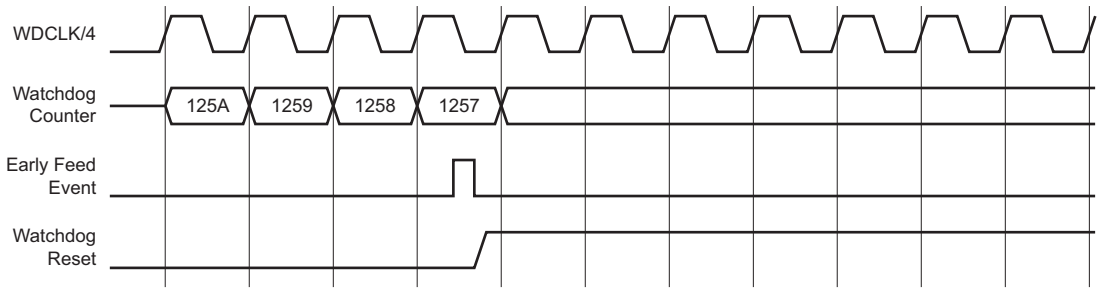


Fig 49. Watchdog block diagram

14.8 Watchdog timing examples

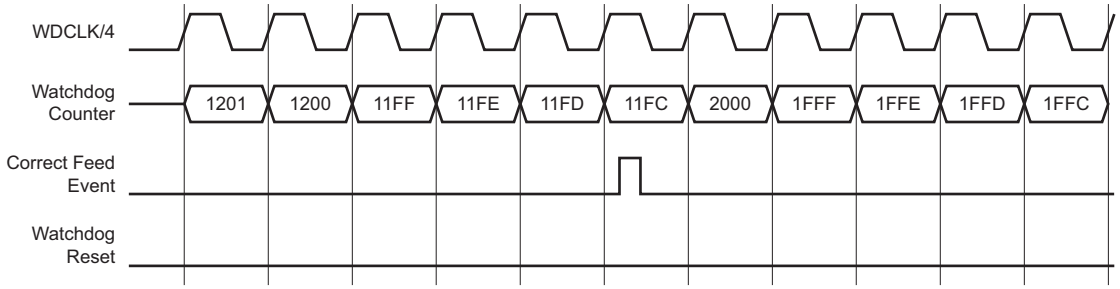
Figure 50, Figure 51 and Figure 52 illustrate several aspects of Watchdog Timer operation.



Conditions:
WDWINDOW= 0x1200
WDWARNINT= 0x3FF
WDTC= 0x2000

aaa-005613

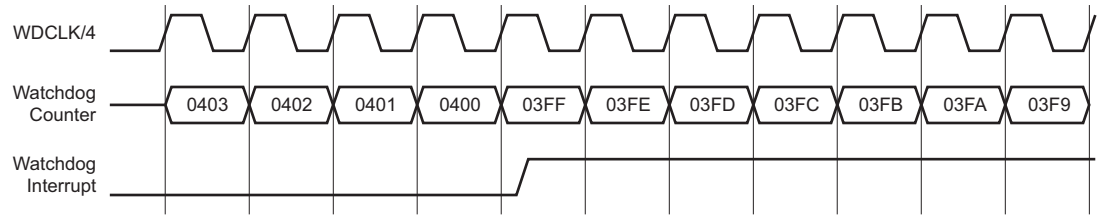
Fig 50. Early Watchdog Feed with Windowed Mode Enabled



Conditions:
WDWINDOW= 0x1200
WDWARNINT= 0x3FF
WDTC= 0x2000

aaa-005614

Fig 51. Correct Watchdog Feed with Windowed Mode Enabled



Conditions:
WDWINDOW= 0x1200
WDWARNINT= 0x3FF
WDTC= 0x2000

aaa-005615

Fig 52. Watchdog Warning Interrupt

15.1 Introduction

The Metrology Engine performs energy measurements for 45 mains periods. This results in at least one measurement update per second for mains frequencies above 45 Hz. The Metrology Engine processes 128 data samples per mains period. The theoretical bandwidth for the voltage and current inputs is 3.2 kHz for 50 Hz mains frequency and 3.84 kHz for 60 Hz mains frequency. This means that for both 50 Hz and 60 Hz mains, over 60 harmonics can be measured by the Metrology Engine.

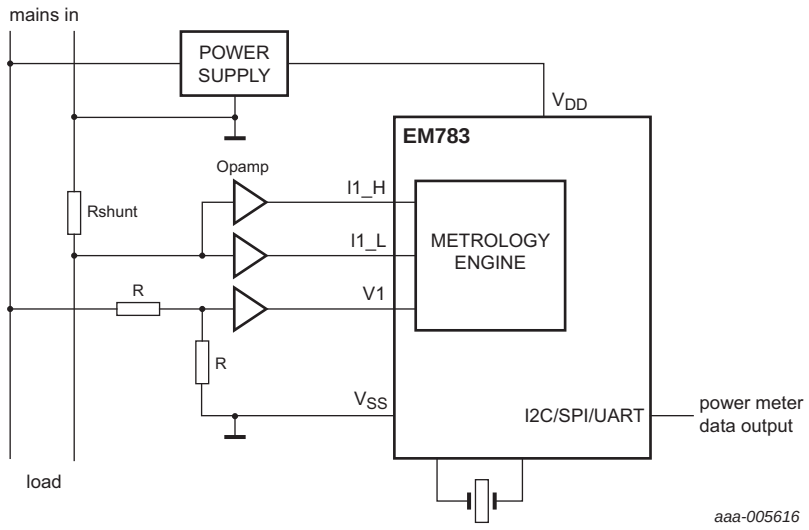


Fig 53. EM783-SC with external measurement circuitry connected to Metrology Engine

Figure 53 shows the EM783-SC with an example of the external measurement circuitry connected to the Metrology Engine. In this case, the current measurements are done using a resistor Rshunt and two current measurement channels I1_H and I1_L. Voltage is measured using a resistor voltage divider and the V1 channel.

The opamps in the measurement circuit amplify or attenuate the mains current and voltage signals to the level required by the Metrology Engine. The opamps also apply a DC-bias in order to enable the measurement of AC signals. Finally, the opamps can also be used as low-pass filters to limit the input signal bandwidth and by doing so to prevent aliasing.

15.2 Features

- Metrology Engine for Energy Measurements
- Outputs RMS voltage (V), RMS current (I), active power (P), apparent power (S), non-active power (Q), power factor (PF), fundamental reactive power (Q1), fundamental apparent power (S1), fundamental power factor (PF1), non-fundamental

apparent power (SN) and current total harmonic distortion (THDI)), frequency (F), measurement duration (D) and temperature (T) in accordance with *IEEE Std 1459-2010*.

- Active power measurement accuracy better than 1 % according to *IEC 62053-21*.
- Minimum configuration: One voltage input and one current input.
- Optional second current input for increased dynamic range.
- Measurement range from V_{SS} to V_{DD}. Do not exceed the V_{DD} voltage level.
- Automatic mains frequency tracking reducing measurement ripple.
- Temperature measurement output for temperature compensation by application software.
- Input signal bandwidth sufficient to process over 60 harmonics.

15.3 Pin description

Table 181 gives a brief summary of the Metrology related pins of EM783. Refer to Table 72 for pin description of all other variants.

Table 181. Metrology Engine pin description

Pin Name	Type	Description
Ix_H	Input	**Analog Input.** The Metrology Engine measures the high-gain current on this input. An external circuit must convert the AC measurement current to the input voltage range. **Remark:** The maximum input voltage must not exceed V_{DD}.
Ix_L / Ix	Input	**Analog Input.** The Metrology Engine measures the low-gain current on this input. An external circuit must convert the AC measurement current to the input voltage range. **Remark:** The maximum input voltage must not exceed V_{DD}.
Vx	Input	**Analog Input.** The Metrology Engine measures the voltage on this input. An external circuit must convert the AC measurement voltage to this voltage range. **Remark:** The maximum input voltage must not exceed V_{DD}.
VBIAS	Input	**Analog Input.** Bias voltage. **Remark:** The maximum input voltage must not exceed V_{DD}.
V_{DD}	Input	Reference voltage for the Metrology Engine.

15.4 External analog front-end design guidelines

The energy measurement system requires one current sensor per phase, however the output of the same current sensor may be connected to up to two EM783 inputs with different gain paths depending upon the variant selected. The EM783-SC/SP/MC3 uses Ix_H (high-gain) and Ix_L (low-gain) inputs for each current sensor, whereas EM783-TP/MC6 uses only single current input, Ix, for each current sensor. The metrology engine combines low-gain and high-gain paths to provide wide dynamic range of 1000:1. For maximum dynamic range the DC bias of the AFE should be $0.5 \times V_{DD}$

Note: For EM783-SP the I2_L and I2_H are used for neutral current sensing.

15.4.1 High-gain current inputs (I1_H, I2_H and I3_H)

The I1_H, I2_H, I3_H are the high-gain path current inputs for EM783-SC/SP/MC3 depending on the selected variant.

The output of the current sensor used for low-gain path is also provided at I1_H/I2_H/I3_H inputs through a high-gain path. The AFE gain of this path should be ideally 32 (minimum 30) times the gain for matching low-gain path. However the absolute value of gain is not very critical because the calibration procedure available in API will calibrate for actual gain applied.

15.4.2 Low-gain current inputs (I1_L, I2_L, I3_L, I1, I2, I3, I4, I5 and I6)

The I1_L, I2_L, I3_L are the low-gain path current inputs for EM783-SC/SP/MC3 whereas I1, I2, I3, I4, I5, I6 are the only current inputs for EM783-TP/MC6.

The AFE in this path should be such that the input to metrology engine is limited between $0.05 \times V_{DD}$ and $0.95 \times V_{DD}$ for upper and lower peaks of maximum input current.

15.4.3 Voltage inputs (V1, V2 and V3)

The V1 input pin is the input for voltage measurements in EM783-SC/SP/MC3/MC6 variants. The EM783-TP variant additionally uses inputs V2 and V3 for voltage measurement of other phases of three-phase mains supply. This section describes the AFE design guidelines for V1. A similar approach can also be used for V2 and V3.

The analog front end external to the EM783 must maintain the voltage at the V1 pin between V_{SS} and V_{DD}. EM783 supports full range up to V_{DD} for all voltage inputs, however the analog front-end components used externally may not be very linear close to V_{DD} and/or V_{SS}. Therefore it is recommended that for overall system linearity the analog front end should be designed so that the upper and lower peaks of maximum input voltage map closely to $0.05 \times V_{DD}$ and $0.95 \times V_{DD}$, respectively.

15.5 Output definitions

The outputs of the Metrology Engine are described in this Chapter. The definitions used in this section are according to *IEEE Std 1459-2010*.

15.5.1 Output for sinusoidal and non-sinusoidal voltage and sinusoidal and non-sinusoidal current

The following parameters are directly measured by the Metrology Engine. Other, derived parameters such as kWh Energy registers can be calculated by the application on the microcontroller core using this data.

Voltage: V — The root-mean-square voltage V in Volt (V) is measured according to:

(4)

$$V = \sqrt{\frac{1}{kT}\int_{\tau}^{\tau+kT} v^2 dt}$$

In this formula, k is the index of the internal voltage samples v inside the Metrology Engine and T is the sample period of the Metrology Engine. The product kT = 1 second.

Current: I — The root-mean-square current in Ampere (A) is measured according to:

(5)

$$I = \sqrt{\frac{1}{kT}\int_{\tau}^{\tau+kT} i^2 dt}$$

In this formula i represents the current samples inside the Metrology Engine.

Apparent Power: S — The apparent power S in Volt-Ampere (VA) is measured as the product of the RMS voltage and RMS current:

(6)

$$S = VI$$

Active Power: P — The active power P in Watt (W) is measured according to:

(7)

$$P = \frac{1}{kT}\int_{\tau}^{\tau+kT} vidt$$

Non-active Power: Q — The non-active power Q in Volt-Ampere-reactive (var) is measured according to:

(8)

$$Q = \sqrt{S^2 - P^2}$$

Power Factor: PF — The dimensionless parameter power factor PF is measured as the ratio between active power and apparent power.

(9)

$$PF = \frac{P}{S}$$

15.5.2 Additional output for sinusoidal voltage and sinusoidal and non-sinusoidal current

The Metrology Engine output described in this chapter provides additional information for sinusoidal voltages and sinusoidal or non-sinusoidal currents.

According to *IEEE Std 1459-2010* these calculations are valid if the voltage waveform has significantly lower distortion than the current waveform. In this case the fundamental active power P1 approximately equals the active power P described above.

Fundamental Reactive Power: Q1 — The fundamental reactive power in Volt-Ampere-reactive (var) is measured according to:

(10)

$$Q_1 = \frac{\omega}{kT} \int_{\tau}^{\tau+kT} i_1 \left[\int v_1 dt \right] dt$$

In this formula, k is the index of the fundamental voltage samples v1 and fundamental current samples i1 inside the Metrology Engine and T is the sample period of the Metrology Engine.

Fundamental Apparent Power: S1 — The fundamental apparent power S1 in Volt-Ampere (VA) is measured according to:

(11)

$$S_1 = \sqrt{P_1^2 + Q_1^2}$$

Fundamental Power Factor: PF1 — The dimensionless parameter fundamental power factor PF1 is measured as the ratio between fundamental active power P1 and fundamental apparent power S1:

(12)

$$PF_1 = \frac{P_1}{S_1}$$

Nonfundamental Apparent Power: SN — The nonfundamental apparent power SN in Volt-Ampere (VA) is measured according to:

(13)

$$S_N = \sqrt{S^2 - S_1^2}$$

Current Total Harmonic Distortion: THDI — This dimensionless parameter is measured using the approximation formula below, which is accurate within 1% for voltages with THDV < 0.05 and for currents with THDI > 0.4. For values outside this range it's an indicative approximation.

(14)

$$THD_I = \frac{S_N}{S_1}$$

Mains frequency: F — The mains frequency F in Hz is calculated using the duration of 45 mains voltage periods.

(15)

$$F = \frac{45}{T_{45periods}}$$

15.5.3 Additional output for temperature compensation and power integration

Temperature: T — The microcontroller temperature T in degrees Celsius (°C) is measured to support temperature compensation of the Metrology Engine measurements. The resolution of the temperature measurement is better than ±3 °C over the full temperature range (−40 °C to +85 °C).

Measurement duration: MD — The measurement duration MD in AHB clock cycles of the last output, this value can be used to integrate output parameters of the Metrology Engine. Examples are the integration of active power P (W) into active energy (kWh) or apparent power S (VA) into apparent energy (kVAh).

15.6 Operation

This section explains the configuration data and calibration procedure for the Metrology Engine. Details on the Metrology Engine interface are described in Chapter 16.

15.6.1 Configuration data

Table 182 summarizes the Metrology Engine configuration parameters for EM783-SC. For other variants similar parameters are used for additional current and voltage channels.

Table 182. Metrology Engine configuration parameters

Member	Description
V1pp	Voltage measurement input range (in Volts peak-peak) for V1 channel
I1_Hpp	Current measurement input range (in Ampere peak-peak) for I1_H channel
I1_Lpp	Current measurement input range (in Ampere peak-peak) for I1_L channel
DeltaPhi1H	Phase error correction angle (in radian) for I1_H channel
DeltaPhi1L	Phase error correction angle (in radian) for I1_L channel

15.6.1.1 Voltage measurement input range V1pp

The parameter V1pp depends on transfer of the analog input circuit in front of the Metrology Engine and it sets the peak-to-peak range for the voltage channel (in Volts).

Examples: For 110 V mains this value will be at least $2\sqrt{2} \times 110 = 311$ V and for 230 V mains at least 650 V. In practice these values need to be larger, because margins for mains voltage fluctuations and for component tolerances need to be accounted for in the input circuit design.

15.6.1.2 Current measurement input range I1_Hpp

The parameter I1_Hpp for the I1_H input depends on transfer of the analog input circuit in front of the Metrology Engine and it sets the peak-to-peak range for the first current channel (in Ampere).

Example: For a single current-channel 100 A meter this value will be at least $2\sqrt{2} \times 100 = 282.8$ A. For a two current-channel 100 A meter this range will cover the range from 0 to 100/32 A and the corresponding value for I1_Hpp will be at least $2\sqrt{2} \times 100/32 = 8.839$ A. In practice the above values need to be larger, because margins for component tolerances need to be accounted for in the input circuit design.

15.6.1.3 Current measurement input range I1_Lpp

The parameter I1_Lpp for the I1_L input depends on the transfer of the analog input circuit in front of the Metrology Engine and it sets the peak-to-peak range for the second current channel (in Ampere). The input circuit for this channel should be designed in such a way that this range is approximately $32 \times$ I1_Hpp for an optimized accuracy curve.

Example: For a two current-channel 100 A meter this range will cover the range from 100/32 A to 100 A and the corresponding value for I1_Hpp will be at least $2\sqrt{2} \times 100 = 282.8$ A In practice this value needs to be larger, because margins for component tolerances need to be accounted for in the input circuit design.

Remark: If there is no second current measurement channel present, this value must be set to zero.

15.6.1.4 Phase error correction angle DeltaPhi1H

Due to component differences, a phase error can occur between the V1 and I1_H channels. This error needs to be corrected for accurate measurements.

The parameter DeltaPhi1H contains the correction angle (in radian) to compensate a phase error between the V1 channel and the I1_H current channel.

15.6.1.5 Phase error correction angle DeltaPhi1L

Due to component differences, a phase error can occur between the V1 and I1_L channels (if this channel is used). This error needs to be corrected for accurate measurements.

The parameter DeltaPhi1L contains the correction angle (in radian) to compensate a phase error between the V1 channel and the I1_L current channel.

If there is no second current measurement channel present, this value can be set to zero.

15.6.2 Calibration procedure

The Metrology Engine can be calibrated by changing the configuration parameters for an individual meter.

Before calibration, make sure to set the right values for AHBClkFrequency and mains frequency Fmains with the initialization of the Metrology Engine.

The Calibration Measurements Setup for EM783-SC is shown in Figure 54.

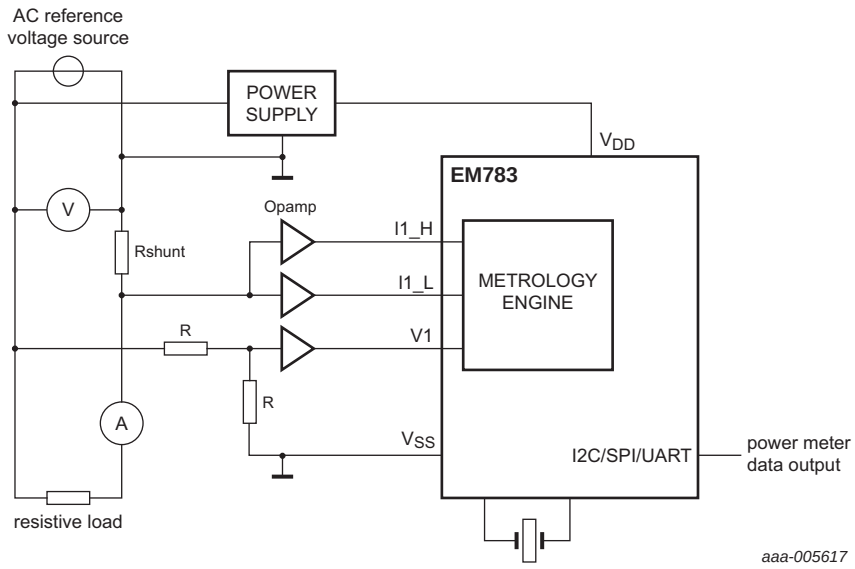


Fig 54. EM783-SC calibration measurements setup

15.6.2.1 V1pp calibration

The voltage input measurement range V1pp can be calibrated as follows:

- Calculate the initial value for V1pp with the transfer of the analog input circuit in front of the Metrology Engine and the input range of the V1 input of the Energy IC (V_{SS} to V_{DD}). Set the parameter V1pp to this initial value.
- Connect an AC reference voltage source to the meter with a reference voltage (Vsource) and frequency (that is, 110V/60Hz or 230V/50Hz).
- Measure the output value for voltage V. The calibrated value for V1pp can then be calculated as:

$$V1pp = V1pp, initialvalue \cdot \left(\frac{V_{source}}{V} \right) \quad (16)$$

- Set this new value as V1pp

The calibration can be verified by checking the measured voltage V again. This should now equal Vsource.

15.6.2.2 I1_Hpp calibration

The current input measurement range I1_Hpp can be calibrated as follows:

- Calculate the initial value for I1_Hpp with the transfer of the analog input circuit in front of the Metrology Engine and the input range of the I1_H input of the Energy IC (V_{SS} to V_{DD}). Set the parameter I1_Hpp to this initial value.
- Connect an AC reference voltage source to the meter with a reference voltage and frequency (that is,110V/60Hz or 230V/50Hz).

- Connect a reference load with a known current Iref1. **Remark**: Make sure the current is in the range for I1_Hpp.
- Measure the output value for current I. The calibrated value for I1_Hpp can then be calculated as:

$$I1\_Hpp = I1\_Hpp, initialvalue \cdot \left(\frac{I_{ref\,1}}{I} \right) \tag{17}$$

- Set this new value as I1_Hpp.

The calibration can be verified by checking the measured current I again. This should now equal Iref1.

15.6.2.3 I1_Lpp calibration

The current input measurement range I1_Lpp can be calibrated as follows:

- Calculate the initial value for I1_Lpp with the transfer of the analog input circuit in front of the Metrology Engine and the input range of the I1_L input of the Energy IC (V_{SS} to V_{DD}). Set the parameter I1_Lpp to this initial value.
- Connect an AC reference voltage source to the meter with a reference voltage and frequency (that is, 110V/60Hz or 230V/50Hz).
- Connect a reference load with a known current Iref2. **Remark**: Ensure the current is in the range for I1_Lpp.
- Measure the output value for current I. The calibrated value for I1_Lpp can then be calculated as:

$$I1\_Lpp = I1\_Lpp, initialvalue \cdot \left(\frac{I_{ref\,2}}{I} \right) \tag{18}$$

- Set this new value as I1_Lpp.

The calibration can be verified by checking the measured current I again. This should now equal Iref2.

15.6.2.4 DeltaPhi1H calibration

The phase error correction angle DeltaPhi1H can be calibrated as follows:

- Connect an AC reference voltage source to the meter with the correct voltage and frequency (that is, 110V/60Hz or 230V/50Hz).
- Set DeltaPhi1H to zero and apply a resistive load (that has no reactive power) to the meter. **Remark**: Ensure the current is in the range for I1_Hpp.
- Measure the output values for fundamental Reactive Power Q1 and active power P. The correction value DeltaPhi1H can then be calculated as:

$$DeltaPhi1H = \arctan\left(\frac{Q_1}{P} \right) \tag{19}$$

- Set this calculated value as DeltaPhi1H.

The calibration can be verified by measuring the fundamental reactive power Q1 for this resistive load again. The fundamental reactive power Q1 should now be reduced towards 0 var for the resistive load.

15.6.2.5 DeltaPhi1L calibration

The phase error correction angle DeltaPhi1L can be calibrated as follows:

- Connect an AC reference voltage source to the meter with the correct voltage and frequency (that is, 110V/60Hz or 230V/50Hz).
- Set DeltaPhi1L to zero and apply a resistive load (that has no reactive power) to the meter. **Remark**: Ensure the current is in the range for I1_Lpp.
- Measure the output values for fundamental Reactive Power Q1 and active power P. The correction value DeltaPhi1L can then be calculated as:

$$DeltaPhi1L = \arctan\left(\frac{Q_1}{P}\right) \tag{20}$$

- Set this calculated value as DeltaPhi1L.

The calibration can be verified by measuring the fundamental reactive power Q1 for this resistive load again. The fundamental reactive power should now be reduced towards 0 var for the resistive load.

16.1 Metrology driver interface

The metrology library Application Programming Interface (API) allows the user application to perform the following operations:

1. Metrology engine initialization
2. Metrology engine interrupt handling
3. Metrology engine calibration
4. Metrology engine start and stop
5. Metrology engine measurement data read.

16.1.1 Metrology engine initialization

This interface configures the metrology engine with the appropriate settings based on the given system clock frequency and the mains frequency.

16.1.2 Metrology engine interrupt handling

The user application must add the metrology engine interrupt handler to the interrupt vector table. The metrology library interrupt handler takes appropriate action according the measured data and the configured ranges.

16.1.3 Metrology engine calibration

The calibration interface allows the user application to configure the calibration parameters for the voltage and current inputs.

For the SC variant the parameters are:

- V1pp for the V1 input
- I1_Hpp for the I1_H input
- I1_Lpp for the I1_L input
- DeltaPhi1H for the I1_H input
- DeltaPhi1L for the I1_L input

For details of the range of parameters for the other variant, please refer to the EM783 API document.

16.1.4 Metrology engine start and stop

These interfaces are used to start and stop the metrology engine operations. These APIs allow the metrology engine to be stopped before loading the new calibration parameters and then restarting the metrology engine.

16.1.5 Metrology measurement data

The metrology engine measures and reports the following metrology parameters:

Table 183. Metrology result class structure

Member	Description
V	RMS voltage (in Volts)
I	RMS current (in Ampere)
P	Active power (in Watts)
Q1	Fundamental reactive power (in var)
S	Apparent power (in VA)
S1	Fundamental apparent power (in VA)
PF	Power factor (no dimension, 0.00 to 1.00)
PF1	Fundamental power factor (no dimension, 0.00 to 1.00)
SN	Nonfundamental apparent power (in VA)
Q	Nonactive power (in var)
THDI	Total harmonic distortion of the current (no dimension)
F	Mains frequency (in Hz)
D	Measurement duration (in AHB clock cycles)
T	Temperature (in degrees C)

16.2 Metrology Engine interface

For more details on the metrology engine interface, the data structures and the API call sequence, refer to the *EM783 Application Programming Interface* document.

17.1 How to read this chapter

The DAC block is identical for all EM783 parts.

17.2 Basic configuration

The DAC is configured using the following registers:

1. Pins: The DAC output pin is configured in the IOCON register block (Table 69).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 21 (Table 19). Power to the DAC is controlled through the PDRUNCFG register (Table 30).

17.3 Features

- 10-bit digital to analog converter
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable speed as a function of power
- Maximum update rate of 1 MHz
- Conversion can be triggered by a selected edge on an internal or external signal
- Interrupt option when an edge triggers a conversion

17.4 Pin description

Table 184 describes the DAC output. This output is driven out onto a pin selected as described in Section 6.3.1 “I/O configuration registers”.

Table 184. D/A pin description

Pin	Type	Description
AOUT	Output	**Analog Output.** After the selected settling time after a conversion starts, the voltage on this signal (with respect to V_{SS}) is VALUE × (V_{DD}/1024).

17.5 Register description

Table 185. Register overview: DAC (base address 0x4002 4000)

Name	Access	Address offset	Description	Reset value
CR	R/W	0x000	D/A control register	0

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

17.5.1 D/A Control Register (CR)

This read/write register includes the digital value to be converted to analog, and a bit that trades off performance versus power. Bits 5:0 are reserved for future, higher-resolution D/A converters.

Table 186: D/A Converter Register (CR - address 0x4002 4000) bit description

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after a conversion begins, the voltage on the AOUT pin (with respect to V_{SS}) is VALUE × (V_{DD}/1024).	0
16	BIAS		Settling time. The settling times noted in the description of the BIAS bit are valid for a capacitance load on the AOUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time.	0
		0	The settling time of the DAC is 1 μs max[1], and the maximum current is 700 μA. This allows a maximum update rate of 1 MHz[1].	
		1	The settling time of the DAC is 2.5 μs[1] and the maximum current is 350 μA. This allows a maximum update rate of 400 kHz[1].	
19:17	TRIG		The value written to this field determines whether conversion begins immediately after this register is written, or whether conversion is delayed until a selected event occurs.	000
		0x0	Conversion begins when this register is written, and AOUT begins to change to the new voltage immediately. For all other values in this field, AOUT remains at its previous voltage until the selected event has occurred.	
		0x1	Reserved	
		0x2	Conversion is triggered by the selected edge(s) on ATRG0.	
		0x3 -0x7	Reserved	
20	-		Reserved	NA
22:21	EDGESEL		For non-zero values of TRIG, this field selects when the conversion is triggered:	00
		0x0	Falling edges	
		0x1	Rising edges	
		0x2	Both edges	
		0x3	Both edges	
23	TRIGERD		If the TRIG field (shown earlier) is non-zero, this bit is set when a conversion is triggered, and is cleared by any write to this register.	0
31:24	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] These times/frequencies are guaranteed if $V_{DD} \geq 2.6$ V.

17.5.2 Power-Down Control

Power control of the DAC is implemented in the System Control block. See Table 30.

17.6 Operation

17.6.1 Hardware-triggered conversion

If the TRIG field is non-zero, the D/A converter starts a conversion when a transition occurs on a selected pin or timer match signal.

17.6.2 Interrupts

An interrupt request is asserted to the interrupt controller when the TRIGERD bit in the DACR is 1. Software can use the Interrupt Enable bit that corresponds to the DAC in the interrupt controller to control whether this results in an interrupt. Software can write to the DACR to clear the request.

17.7 Block diagram

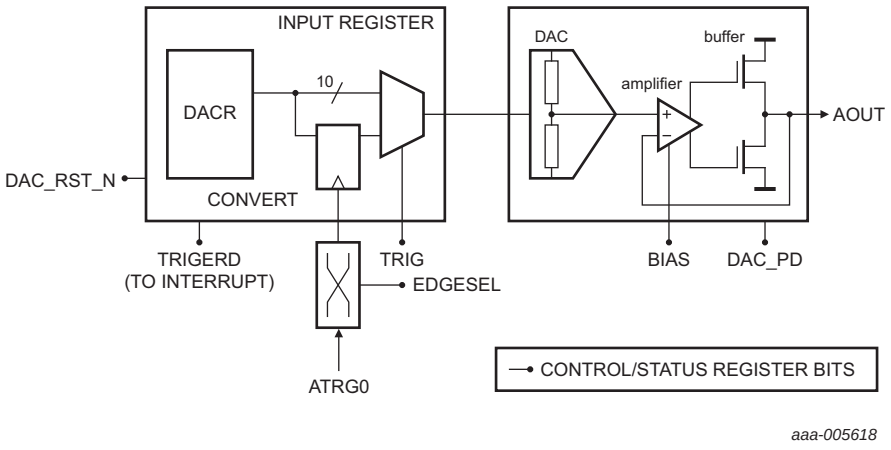


Fig 55. DAC block diagram

18.1 Bootloader

The bootloader controls initial operation after reset and also provides the means to accomplish programming of the flash memory. This could be initial programming of a blank device, erasure and reprogramming of a previously programmed device, or programming of the flash memory by the application program in a running system.

Remark: SRAM location 0x1000 0000 to 0x1000 0050 is not used by the bootloader and the memory content in this area is retained during reset. SRAM memory is not retained when the part is powered down.

18.2 Features

- In-System Programming: In-System programming (ISP) programs or reprograms the on-chip flash memory, using the bootloader software and UART serial port. This can be done when the part resides in the end-user board.
- In-Application Programming: In-Application programming (IAP) performs erase and write operation on the on-chip flash and EEPROM memories, as directed by the end-user application code.
- Flash access times can be configured through a register in the flash controller block.
- 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 Table 223.

18.3 Applications

The bootloader provides both In-System and In-Application programming interfaces for programming the on-chip flash and EEPROM memories.

18.4 Description

The bootloader code is executed every time the part is powered on or reset. The loader can execute the ISP command handler or the user application code. A LOW level after reset on the P0_1 pin is considered as an external hardware request to start the ISP command handler. Assuming that power supply pins are on their nominal levels when the rising edge on $\overline{\text{RESET}}$ pin is generated, it may take up to 3 ms before P0_1 is sampled and the decision on whether to continue with user code or ISP handler is made. If P0_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 (P0_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.

Pin P0_1 that is used as hardware request for ISP requires no special attention. Since P0_1 is in pull-up mode after reset, the default operation is to check for valid user code.

18.4.1 Memory map after any reset

The boot block is 16 kB in size. The boot block 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, that is, the bottom 512 bytes of the boot block are also visible in the memory region starting from the address 0x0000 0000.

18.4.1.1 Criterion for Valid User Code

Criterion for valid user code: The reserved Cortex-M0 exception vector location 7 (offset 0x 0000 001C in the vector table) should contain the two's complement of the checksum of table entries 0 to 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 serial port 0. The host should send a '?' (0x3F) as a synchronization character and wait for a response. The host side serial port settings should be 8 data bits, 1 stop bit and no parity. The auto-baud routine measures the bit time of the received synchronization character in terms of its own frequency and programs the baud rate generator of the serial port. It also sends an ASCII string ("Synchronized<CR><LF>") to the Host. In response to this, the host should send the same string ("Synchronized<CR><LF>"). The auto-baud routine looks at the received characters to verify synchronization. If synchronization is verified, then "OK<CR><LF>" string is sent to the host. The host should respond by sending the crystal frequency (in kHz) at which the part is running. For example, if the part is running at 10 MHz, the response from the host should be "10000<CR><LF>". "OK<CR><LF>" string is sent to the host after receiving the crystal frequency. If synchronization is not verified then the auto-baud routine waits again for a synchronization character. For auto-baud to work correctly in case of user invoked ISP, the CCLK frequency should be greater than or equal to 10 MHz.

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 18.6 "ISP commands" on page 217.

18.4.2 Communication protocol

All ISP commands should be sent as single ASCII strings. Strings should be terminated with Carriage Return (CR) and/or Line Feed (LF) control characters. Extra <CR> and <LF> characters are ignored. All ISP responses are sent as <CR><LF> terminated ASCII strings. Data is sent and received in UU-encoded format.

18.4.2.1 Connections

On the EM783, the RXD function on P0_12 and the TXD function on P0_13 are used for ISP communication.

18.4.2.2 ISP command format

“Command Parameter_0 Parameter_1 ... Parameter_n<CR><LF>” “Data” (Data only for Write commands).

18.4.2.3 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).

18.4.2.4 ISP data format

The data stream is in UU-encode 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 checksum after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) that is, it can hold 45 data bytes. The receiver should compare it with the checksum of the received bytes. If the checksum matches then the receiver should respond with “OK<CR><LF>” to continue further transmission. If the checksum does not match, the receiver should respond with “RESEND<CR><LF>”. In response the sender should retransmit the bytes.

A description of UU-encoding is available at: http://en.wikipedia.org/wiki/Uuencoding.

18.4.2.5 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.

18.4.2.6 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.

18.4.2.7 Interrupts during ISP

The boot block interrupt vectors located in the boot block of the flash are active after any reset.

18.4.2.8 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. Before making any IAP call, either disable the interrupts or ensure that the user interrupt vectors are active in RAM and that the interrupt handlers reside in RAM. The IAP code does not use or disable interrupts.

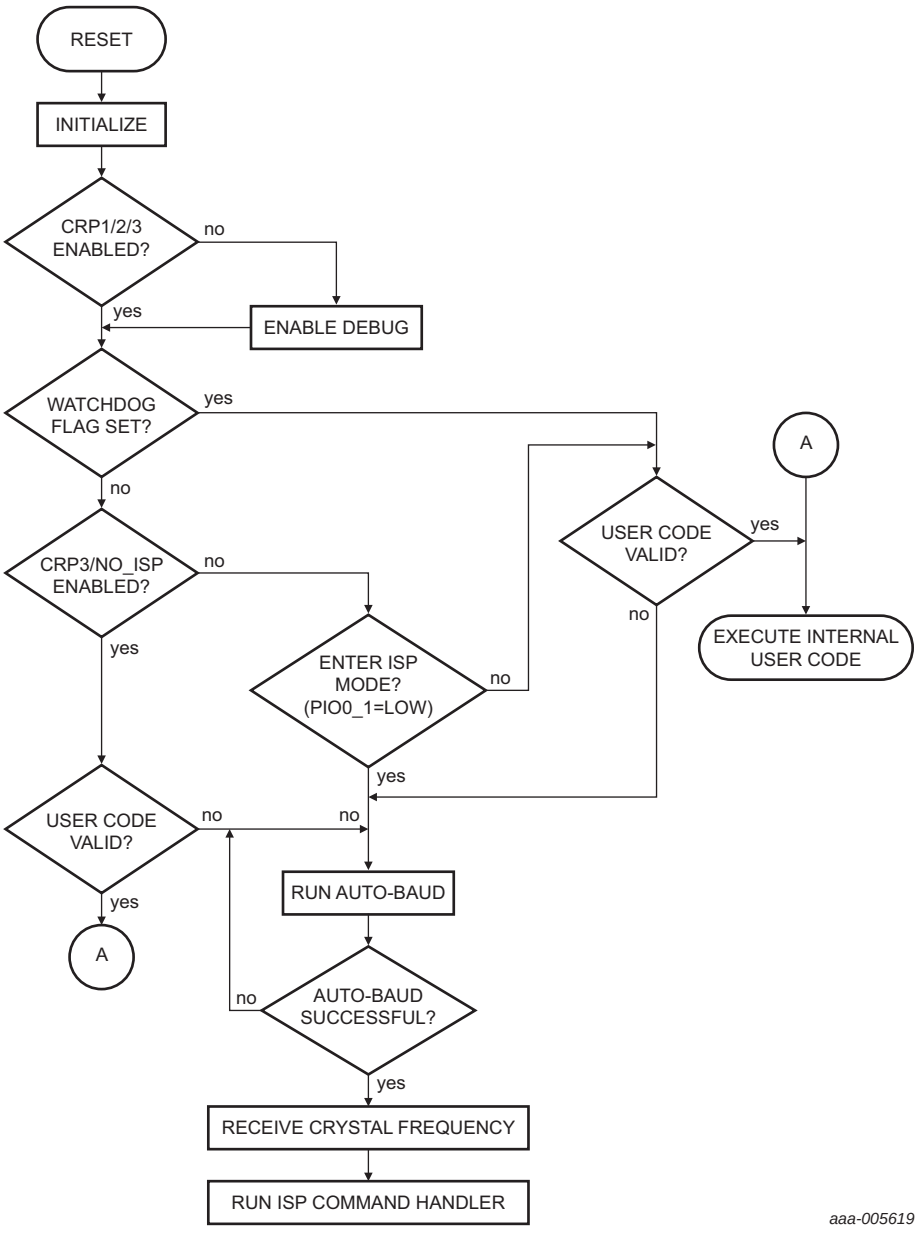
18.4.2.9 RAM used by ISP command handler

ISP commands use on-chip RAM from 0x1000 017C to 0x1000 025B. This area could be used, but the contents may be lost upon reset. Flash programming commands use the top 32 bytes of on-chip RAM. The stack is located at RAM top − 32. The maximum stack usage is 256 bytes and it extends downwards.

18.4.2.10 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 it extends downwards.

18.4.3 Boot process flowchart



(1) For details on handling the crystal frequency, see Section 18.7.8 “Reinvoke ISP” on page 229

Fig 56. Boot process flowchart

18.4.4 Sector numbers

Some IAP and ISP commands operate on “sectors” and specify sector numbers. Table 187 shows the correspondence between sector numbers and memory addresses for EM783 devices.

Table 187. Sectors in an EM783 device

Sector number	Sector size	Address range
0	4 kB	0x0000 0000 - 0x0000 0FFF
1	4 kB	0x0000 1000 - 0x0000 1FFF
2	4 kB	0x0000 2000 - 0x0000 2FFF
3	4 kB	0x0000 3000 - 0x0000 3FFF
4	4 kB	0x0000 4000 - 0x0000 4FFF
5	4 kB	0x0000 5000 - 0x0000 5FFF
6	4 kB	0x0000 6000 - 0x0000 6FFF
7	4 kB	0x0000 7000 - 0x0000 7FFF

18.5 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 188. Code Read Protection options

Name	Pattern programmed in 0x0000 02FC	Description
NO_ISP	0x4E69 7370	Prevents sampling of pin P0_1 for entering ISP mode. P0_1 is available for other uses.
CRP1	0x12345678	Debugging via the Serial Wire 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 below 0x1000 0200 is disabled. • Copy RAM to flash command cannot 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. • Blank check command reports blank sectors only. Offset and value of non-blank sectors are always reported as 0. This mode is useful when CRP is required and flash field updates are needed but all sectors cannot 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	Debugging via the Serial Wire pins is disabled. The following ISP commands are disabled: • Read Memory • Write to RAM • Go • Copy RAM to flash • Compare • Blank check command reports blank sectors only. Offset and value of non-blank sectors are always reported as 0. When CRP2 is enabled, the ISP erase command only allows erasure of all user sectors.
CRP3	0x43218765	Debugging via the Serial Wire pins is disabled. ISP entry by pulling P0_1 LOW is disabled if a valid user code is present in flash sector 0. This mode effectively disables ISP override using P0_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 UART. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Table 189. Code Read Protection hardware/software interaction

CRP option	User Code Valid	P0_1 pin at reset	SW debug enabled	Device enters ISP mode	partial flash update in ISP mode
No	No	x	Yes	Yes	Yes
No	Yes	High	Yes	No	NA
No	Yes	Low	Yes	Yes	Yes
CRP1	Yes	High	No	No	NA

Table 189. Code Read Protection hardware/software interaction *...continued*

CRP option	User Code Valid	P0_1 pin at reset	SW debug enabled	Device enters ISP mode	partial flash update in ISP mode
CRP1	Yes	Low	No	Yes	Yes
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 190. ISP commands allowed by 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)	yes; reports blank sectors only	yes; reports blank sectors only	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.

18.5.1 ISP entry protection

In addition to the three CRP modes, the user can prevent the sampling of pin P0_1 for entering ISP mode and thereby release pin P0_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.

18.6 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 the received ISP command has been 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 191. ISP command summary

ISP Command	Usage	Described in
Unlock	U <Unlock Code>	Table 192
Set Baud Rate	B <Baud Rate> <stop bit>	Table 193
Echo	A <setting>	Table 194
Write to RAM	W <start address> <number of bytes>	Table 195
Read Memory	R <address> <number of bytes>	Table 196
Prepare sector(s) for write operation	P <start sector number> <end sector number>	Table 197
Copy RAM to flash	C <Flash address> <RAM address> <number of bytes>	Table 198
Go	G <address> <Mode>	Table 199
Erase sector(s)	E <start sector number> <end sector number>	Table 200
Blank check sector(s)	I <start sector number> <end sector number>	Table 201
Read Part ID	J	Table 202
Read Boot code version	K	Table 204
Compare	M <address1> <address2> <number of bytes>	Table 205
ReadUID	N	Table 206

18.6.1 Unlock <Unlock code>

Table 192. 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.

18.6.2 Set Baud Rate <Baud Rate> <stop bit>

Table 193. ISP Set Baud Rate command

Command	B
Input	Baud Rate: 9600 | 19200 | 38400 | 57600 | 115200 | 230400 Stop bit: 1 | 2
Return Code	CMD_SUCCESS | INVALID_BAUD_RATE | INVALID_STOP_BIT | PARAM_ERROR
Description	This command is used to change the baud rate. The new baud rate is effective after the command handler sends the CMD_SUCCESS return code.
Example	"B 57600 1<CR><LF>" sets the serial port to baud rate 57600 bps and 1 stop bit.

18.6.3 Echo <setting>

Table 194. 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.

18.6.4 Write to RAM <start address> <# bytes>

The host should send the data only after receiving the CMD_SUCCESS return code. The host should send the checksum 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), that is, it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines, then the checksum should be of the actual number of bytes sent. The ISP command handler compares it with the checksum of the received bytes. If the checksum matches, the ISP command handler responds with “OK<CR><LF>” to continue further transmission. If the checksum does not match, the ISP command handler responds with “RESEND<CR><LF>”. In response the host should retransmit the bytes.

Table 195. 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.

18.6.5 Read Memory <address> <no. of bytes>

The data stream is followed by the command success return code. The checksum 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), that is, it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines, then the checksum is of actual number of bytes sent. The host should compare it with the checksum of the received bytes. If the checksum matches then the host should respond with “OK<CR><LF>” to continue further transmission. If the checksum does not match, then the host should respond with “RESEND<CR><LF>”. In response the ISP command handler sends the data again.

Table 196. 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.

18.6.6 Prepare sector(s) for write operation <start sector number> <end sector number>

This command makes flash write/erase operation a two-step process.

Table 197. 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 cannot 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.

18.6.7 Copy RAM to flash <Flash address> <RAM address> <no of bytes>

Table 198. 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.
Example	“C 0 268467504 512<CR><LF>” copies 512 bytes from the RAM address 0x1000 0800 to the flash address 0.

18.6.8 Go <address> <mode>

Table 199. 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 (not allowed).
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.

18.6.9 Erase sector(s) <start sector number> <end sector number>

Table 200. 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 cannot 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.

18.6.10 Blank check sector(s) <sector number> <end sector number>

Table 201. 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 remapped to flash boot block.**
Example	"I 2 3<CR><LF>" blank checks the flash sectors 2 and 3.

18.6.11 Read Part Identification number

Table 202. ISP Read Part Identification command

Command	J
Input	None.
Return Code	CMD_SUCCESS followed by part identification number in ASCII (see Table 203).
Description	This command is used to read the part identification number.

Table 203. EM783 part identification numbers

Device	Hex coding
EM783-SC	t.b.d.
EM783-SP	t.b.d.
EM783-TP	t.b.d.
EM783-MC3	t.b.d.
EM783-MC6	t.b.d.
EM783-MC7	t.b.d.

18.6.12 Read Boot code version number

Table 204. ISP Read Boot Code version 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.

18.6.13 Compare <address1> <address2> <# bytes>

Table 205. 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.
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 remapped 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.

18.6.14 ReadUID

Table 206. ReadUID command

Command	N
Input	None
Return Code	CMD_SUCCESS followed by four 32-bit words of E-sort test information in ASCII format. The word sent at the lowest address is sent first.
Description	This command is used to read the unique ID.

18.6.15 ISP Return Codes

Table 207. 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.

Table 207. ISP Return Codes Summary *…continued*

Return Code	Mnemonic	Description
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid or end sector number is greater than start sector number.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data not equal.
11	BUSY	Flash programming hardware interface is busy.
12	PARAM_ERROR	Insufficient number of parameters or invalid parameter.
13	ADDR_ERROR	Address is not on word boundary.
14	ADDR_NOT_MAPPED	Address is not mapped in the memory map. Count value is taken in to consideration where applicable.
15	CMD_LOCKED	Command is locked.
16	INVALID_CODE	Unlock code is invalid.
17	INVALID_BAUD_RATE	Invalid baud rate setting.
18	INVALID_STOP_BIT	Invalid stop bit setting.
19	CODE_READ_PROTECTION_ENABLED	Code read protection enabled.

18.7 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. 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 large enough to hold all the results in case if number of results are more than number of parameters. Parameter passing is illustrated in the Figure 57. 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 is a change to the 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);
```

As per the ARM specification (*ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05*) up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively. Additional parameters are passed on the stack. Up to 4 parameters can be returned in the r0, r1, r2 and r3 registers respectively. Additional parameters are returned indirectly via memory. Some of the IAP calls require more than 4 parameters. If the ARM suggested scheme is used for the parameter passing/returning, then it might create problems due to difference in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such risk.

The flash memory is not accessible during a write or erase operation. IAP commands, which result 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 use this space if IAP flash programming is permitted in the application.

Table 208. IAP Command Summary

IAP Command	Command Code	Described in
Prepare sector(s) for write operation	50_{10}	Table 209
Copy RAM to flash	51_{10}	Table 210
Erase sector(s)	52_{10}	Table 211
Blank check sector(s)	53_{10}	Table 212
Read Part ID	54_{10}	Table 213
Read Boot code version	55_{10}	Table 214
Compare	56_{10}	Table 215
Reinvoke ISP	57_{10}	Table 216
Read UID	58_{10}	Table 217
EEPROM Write	61_{10}	Table 218
EEPROM Read	62_{10}	Table 219

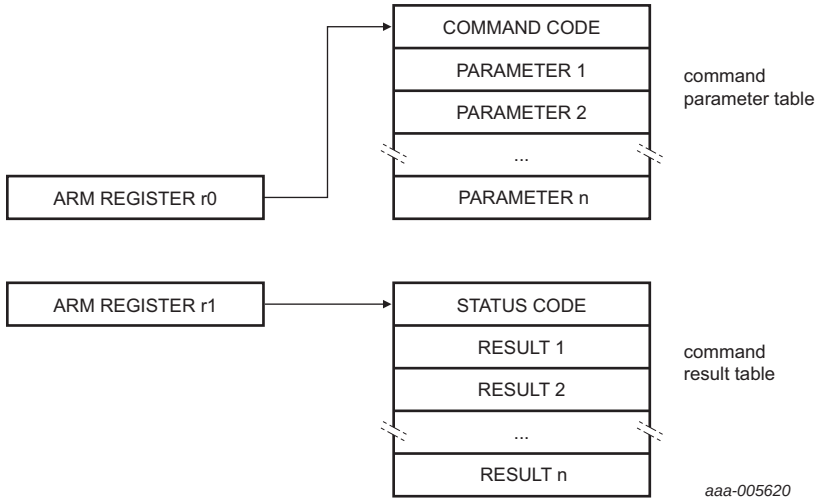


Fig 57. IAP parameter passing

18.7.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two-step process.

Table 209. IAP Prepare sector(s) for write operation command

Command	Prepare sector(s) for write operation
Input	**Command code: 50_{10}** **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 cannot be prepared by this command. To prepare a single sector use the same “Start” and “End” sector numbers.

18.7.2 Copy RAM to flash

Table 210. IAP Copy RAM to flash command

Command	Copy RAM to flash
Input	**Command code: 51_{10}** **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 cannot be written by this command.

18.7.3 Erase Sector(s)

Table 211. IAP Erase Sector(s) command

Command	Erase Sector(s)
Input	**Command code: 52_{10}** **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 cannot be erased by this command. To erase a single sector use the same “Start” and “End” sector numbers.

18.7.4 Blank check sector(s)

Table 212. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	**Command code: 53_{10}** **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.

18.7.5 Read Part Identification number

Table 213. IAP Read Part Identification command

Command	Read part identification number
Input	**Command code: 54_{10}** **Parameters:** None
Return Code	CMD_SUCCESS |
Result	**Result0:** Part Identification Number.
Description	This command is used to read the part identification number.

18.7.6 Read Boot code version number

Table 214. IAP Read Boot Code version command

Command	Read boot code version number
Input	**Command code: 55_{10}** **Parameters:** None
Return Code	CMD_SUCCESS |
Result	**Result0:** 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.

18.7.7 Compare <address1> <address2> <# bytes>

Table 215. IAP Compare command

Command	Compare
Input	**Command code: 56_{10}** **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 remapped to RAM.**

18.7.8 Reinvoke ISP

Table 216. Reinvoke ISP

Command	Compare
Input	**Command code: 57_{10}**
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, and resets the FDR (see Table 92). This command may be used when a valid user program is present in the internal flash memory and the P0_1 pin is not accessible to force the ISP mode.

18.7.9 Read UID

Table 217. IAP Read UID command

Command	Compare
Input	**Command code: 58_{10}**
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.

18.7.10 Write EEPROM

Table 218. IAP Write EEPROM command

Command	Compare
Input	**Command code: 61_{10}** **Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be written. **Param3:** System Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED
Result	none
Description	Data is copied from the RAM address to the EEPROM address. **Remark:** The top 64 bytes of the 4 kB EEPROM memory are reserved and cannot be written to.

18.7.11 Read EEPROM

Table 219. IAP Read EEPROM command

Command	Compare
Input	**Command code: 62_{10}** **Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be read. **Param3:** System Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED
Result	none
Description	Data is copied from the EEPROM address to the RAM address.

18.7.12 IAP Status Codes

Table 220. 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 220. IAP Status Codes Summary *...continued*

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.

18.8 Debug notes

18.8.1 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 221. 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

18.8.2 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.

18.9 Flash controller registers

Table 222. 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 223
FMSSTART	R/W	0x020	Signature start address register	0	Table 224
FMSSTOP	R/W	0x024	Signature stop-address register	0	Table 225
FMSW0	R	0x02C	Word 0 [31:0]	-	Table 226
FMSW1	R	0x030	Word 1 [63:32]	-	Table 227
FMSW2	R	0x034	Word 2 [95:64]	-	Table 228
FMSW3	R	0x038	Word 3 [127:96]	-	Table 229
EEMSSTART	R/W	0x9C	EEPROM BIST start address register	0x0	Table 230
EEMSSTOP	R/W	0xA0	EEPROM BIST stop address register	0x0	Table 231

Table 222. Register overview: FMC (base address 0x4003 C000) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
EEMSSIG	R	0xA4	EEPROM 24-bit BIST signature register	0x0	Table 232
FMSTAT	R	0xFE0	Signature generation status register	0	Table 233
FMSTATCLR	W	0xFE8	Signature generation status clear register	-	Table 234

18.9.1 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 EM783 flash memory.

Table 223. 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
		0x1	1 system clock flash access time (for system clock frequencies of up to 20 MHz).	
		0x2	2 system clocks flash access time (for system clock frequencies of up to 40 MHz).	
		0x3	3 system clocks flash access time (for system clock frequencies of up to 48 MHz).	
		0x4	Reserved.	
31:2	-	-	Reserved. **User software must not change the value of these bits. Bits 31:2 must be written back exactly as read**.	-

18.9.2 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 (for example, during programming).

The address range for generating a signature must be aligned on flash-word boundaries, that is, 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 (for example, internal RAM) can be executed during signature generation. This can include interrupt services, if the interrupt vector table is remapped to memory other than the flash memory. The code that initiates signature generation should also be placed outside of the flash memory.

18.9.3 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 224 and Table 225 show the bit assignments in the FMSSTART and FMSSTOP registers respectively.

Table 224. 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 225. 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

18.9.4 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 therefore saves time and code space. The method for generating the signature is described in Section 18.9.10.

Table 229 show bit assignment of the FMSW0 and FMSW1, FMSW2, FMSW3 registers respectively.

Table 226. 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 227. 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 228. 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 229. 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).	-

18.9.5 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 230. 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	-

18.9.6 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 231. 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 starts the BIST. This bit is self-clearing.	0x0	R/W

18.9.7 EEPROM signature register

The EEPROM BIST signature register returns the signatures as produced by the embedded signature generators.

Table 232. 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

18.9.8 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 233. 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

18.9.9 Flash module status clear register

The FMSTATCLR register is used to clear the signature generation completion flag.

Table 234. 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

18.9.10 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 58.

```
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
```

aaa-006647

Fig 58. Algorithm for generating a 128-bit signature

19.1 How to read this chapter

The integer division routines are available on all EM783 parts.

19.2 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.

19.3 General description

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);
} EM_ROM_DIV_STRUCT;
```

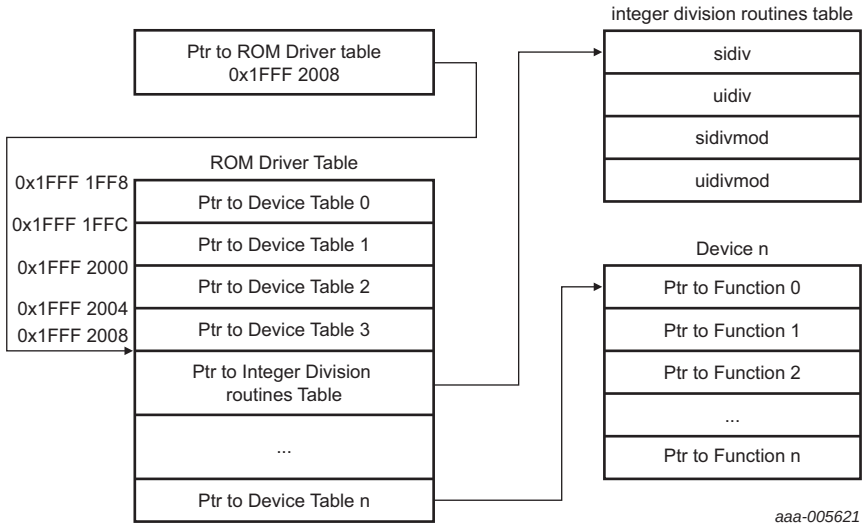


Fig 59. Integer division routines pointer structure

19.4 API description

19.4.1 Examples

19.4.1.1 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 EM_ROM_DIV_STUCT * pROMDiv;
     const unsigned p_dev4;
     const unsigned p_dev5;
     const unsigned p_dev6;
ROM ** rom = (ROM **) 0x1FFF1FF8;
     pDivROM = (*rom)->pROMDiv;

} ROM;
```

19.4.1.2 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) */
```

19.4.1.3 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) */
```

20.1 How to read this chapter

The I2C-bus ROM driver routines are available on all EM783 parts.

20.2 Features

- Simple I2C drivers to send and receive data on the I2C-bus.
- Polled and interrupt-driven receive and transmit functions for Master and Slave modes.

20.3 General description

The drivers are callable for use by any application program to send or receive data on the I2C bus. The purpose of these drivers is to make it quick and easy for the application programmer to produce working projects using the I2C interface.

The ROM routines allow user to operate the I2C interface as a Master or a Slave. The software routines do not implement arbitration to make a Master switch to a Slave mode in the midst of a transmission.

Although multi-master arbitration is not implemented in these I2C drivers, it is possible to use them in a system design with more than one master. If the flag returned from the driver indicates that the message was not successful due to loss of arbitration, the application just resends the message.

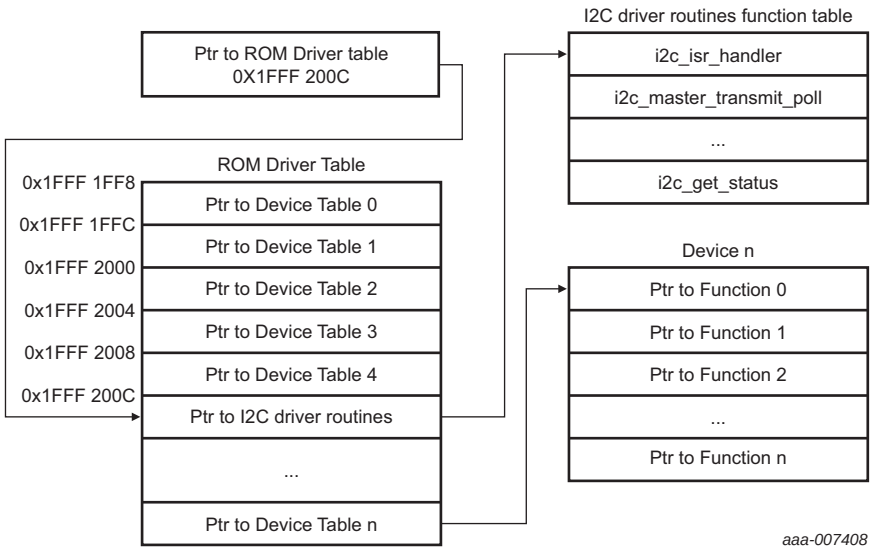


Fig 60. I2C-bus driver routines pointer structure

20.4 API description

Table 235. ISR handler

Routine	ISR handler
Prototype	void i2c_isr_handler(I2C_HANDLE_T*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area.
Return	None.
Description	I2C ROM Driver interrupt service routine. This function must be called from the I2C ISR when using I2C Rom Driver interrupt mode.

Table 236. I2C Master Transmit Polling

Routine	I2C Master Transmit Polling
Prototype	ErrorCode_t i2c_master_transmit_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Transmits bytes in the send buffer to a slave. The slave address with the R/W bit =0 is expected in the first byte of the send buffer. STOP condition is sent at end unless stop_flag =0. When the task is completed, the function returns to the line after the call.

Table 237. I2C Master Receive Polling

Routine	I2C Master Receive Polling
Prototype	ErrorCode_t i2c_master_receive_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Receives bytes from slave and put into receive buffer. The slave address with the R/W bit =0 is expected in the first byte of the send buffer. After the task is finished, the slave address with the R/W bit =1 is in the first byte of the receive buffer. STOP condition is sent at end unless stop_flag =0. When the task is completed, the function returns to the line after the call.

Table 238. I2C Master Transmit and Receive Polling

Routine	I2C Master Transmit and Receive Polling
Prototype	ErrorCode_t i2c_master_tx_rx_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)

Table 238. I2C Master Transmit and Receive Polling *...continued*

Routine	I2C Master Transmit and Receive Polling
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	First, transmit bytes in the send buffer to a slave and secondly, receives bytes from slave and store it in the receive buffer. The slave address with the R/W bit =0 is expected in the first byte of the send buffer. After the task is finished, the slave address with the R/W bit =1 is in the first byte of the receive buffer. STOP condition is sent at end unless stop_flag =0. When the task is completed, the function returns to the line after the call.

Table 239. I2C Master Transmit Interrupt

Routine	I2C Master Transmit Interrupt
Prototype	ErrorCode_t i2c_master_transmit_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Transmits bytes in the send buffer to a slave. The slave address with the R/W bit =0 is expected in the first byte of the send buffer. STOP condition is sent at end unless stop_flag =0. Program control will be returned immediately and task will be completed on an interrupt-driven basis. When task is completed, the callback function is called.

Table 240. I2C Master Receive Interrupt

Routine	I2C Master Receive Interrupt
Prototype	ErrorCode_t i2c_master_receive_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Receives bytes from slave and put into receive buffer. After the task is finished, the slave address with the R/W bit =1 is in the first byte of the receive buffer. STOP condition is sent at end unless stop_flag =0. Program control will be returned immediately and task will be completed on an interrupt-driven basis. When task is completed, the callback function is called.

Table 241. I2C Master Transmit Receive Interrupt

Routine	I2C Master Transmit Receive Interrupt
Prototype	ErrorCode_t i2c_master_tx_rx_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	First, transmits bytes in the send buffer to a slave and secondly, receives bytes from slave and store it in the receive buffer. The slave address with the R/W bit =0 is expected in the first byte of the send buffer. After the task is finished, the slave address with the R/W bit =1 is in the first byte of the receive buffer. STOP condition is sent at end unless stop_flag =0. Program control will be returned immediately and task will be completed on an interrupt-driven basis. When task is completed, the callback function is called.

Table 242. I2C Slave Receive Polling

Routine	I2C Slave Receive Polling
Prototype	ErrorCode_t i2c_slave_receive_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Receives data from master. When the task is completed, the function returns to the line after the call.

Table 243. I2C Slave Transmit Polling

Routine	I2C Slave Transmit Polling
Prototype	ErrorCode_t i2c_slave_transmit_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Sends data bytes back to master. When the task is completed, the function returns to the line after the call.

Table 244. I2C Slave Receive Interrupt

Routine	I2C Slave Receive Interrupt
Prototype	ErrorCode_t i2c_slave_receive_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)

Table 244. I2C Slave Receive Interrupt *…continued*

Routine	I2C Slave Receive Interrupt
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Receives data from master. Program control will be returned immediately and task will be completed on an interrupt-driven basis. When task is completed, the callback function is called.

Table 245. I2C Slave Transmit Interrupt

Routine	I2C Slave Transmit Interrupt
Prototype	ErrorCode_t i2c_slave_transmit_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. I2C_PARAM - Pointer to the I2C PARAM struct. I2C_RESULT - Pointer to the I2C RESULT struct.
Return	ErrorCode.
Description	Sends data to the Master. Program control will be returned immediately and task will be completed on an interrupt-driven basis. When task is completed, the callback function is called.

Table 246. I2C Set Slave Address

Routine	I2C Set Slave Address
Prototype	ErrorCode_t i2c_set_slave_addr(I2C_HANDLE_T*, slave_addr_0_3, slave_mask_0_3)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. Slave_addr_0_3 - unint32 variable. 7-bit slave address . Slave_mask_0_3 - unint32 variable. Slave address mask.
Return	ErrorCode.
Description	Sets the slave address and associated mask. The set_slave_addr() function supports four 7-bit slave addresses and masks.

Table 247. I2C Get Memory Size

Routine	I2C Get Memory Size
Prototype	uint32_t i2c_get_mem_size(void)
Input parameter	None.
Return	uint32.
Description	Returns the number of bytes in SRAM needed by the I2C driver.

Table 248. I2C Setup

Routine	I2C Setup
Prototype	I2C_HANDLE_T* i2c_setup(i2c_base_addr, *start_of_ram)

Table 248. I2C Setup *...continued*

Routine	I2C Setup
Input parameter	I2C_base addr - unint32 variable. Base address for I2C peripherals. Start_of_ram - unint32 pointer. Pointer to allocated SRAM.
Return	I2C_Handle.
Description	Returns a handle to the allocated SRAM area.

Table 249. I2C Set Bit Rate

Routine	I2C Set Bit Rate
Prototype	ErrorCode_t i2c_set_bitrate(I2C_HANDLE_T*, P_clk_in_hz, bitrate_in_bps)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area. P_clk_in_hz - unint32 variable. The Peripheral Clock in Hz. Bitrate_in_bps - unint32 variable. Requested I2C operating frequency in Hz.
Return	ErrorCode.
Description	Configures the I2C duty-cycle registers (SCLH and SCLL).

Table 250. I2C Get Firmware Version

Routine	I2C Get Firmware Version
Prototype	uint32_t i2c_get_firmware_version(void)
Input parameter	None.
Return	I2C ROM Driver version number.
Description	Returns the version number. The firmware version is an unsigned 32-bit number.

Table 251. I2C Get Status

Routine	I2C Get Status
Prototype	I2C_MODE_T i2c_get_status(I2C_HANDLE_T*)
Input parameter	I2C_HANDLE_T - Handle to the allocated SRAM area.
Return	Status code.
Description	Returns status code. The status code indicates the state of the I2C bus. Refer to I2C Status Code Table.

Table 252. Error codes

Error Code	Description	Comment
0	Successful completion	Function was completed successfully.
1	General error	-
0x0006 0001	ERR_I2C_NAK	Correspond to I2C status code 0x20, 0x30, 0x48, 0x88, 0x98 in the I2C Status register[1].
0x0006 0002	ERR_I2C_BUFFER_OVERFLOW	Receive buffer overflow.
0x0006 0003	ERR_I2C_BYTE_COUNT_ERR	Correspond to I2C status code 0xC0, 0xA0, 0xA8, 0xB8 in the I2C Status register[1].
0x0006 0004	ERR_I2C_LOSS_OF_ARBRITRATION	Correspond to I2C status code 0x38, 0x68, 0x78 in the I2C Status register[1].
0x0006 0005	ERR_I2C_SLAVE_NOT_ADDRESSED	Correspond to I2C status code 0xC8 in the I2C Status register[1].

Table 252. Error codes *...continued*

Error Code	Description	Comment
0x0006 0006	ERR_I2C_LOSS_OF_ARBRITRATION_NAK_BIT	Correspond to I2C status code 0x38 in the I2C Status register[1].
0x0006 0007	ERR_I2C_GENERAL_FAILURE	Failure detected on I2C bus.
0x0006 0008	ERR_I2C_REGS_SET_TO_DEFAULT	I2C clock frequency could not be set. Default value of 0x04 is loaded into SCLH and SCLL.

[1] Error code corresponds to I2C status register in Table 119, Table 120, Table 123, Table 124.

Table 253. I2C Status code

Status code	Description
0	IDLE
1	MASTER_SEND
2	MASTER_RECEIVE
3	SLAVE_SEND
4	SLAVE_RECEIVE

20.4.1 I2C ROM driver variables

The I2C ROM driver requires specific variables to be declared and initialized for proper usage. Depending on the operating mode, some variables can be omitted.

20.4.1.1 I2C Handle

The I2C handle is a pointer allocated for the I2C ROM driver. The handle needs to be defined as an I2C handle TYPE:

```
typedef void* I2C_HANDLE_T
```

After the definition of the handle, the handle must be initialized with I2C base address and RAM reserved for the I2C ROM driver by making a call to the i2c_setup() function.

The callback function type must be defined if interrupts for the I2C ROM driver are used:

```
typedef void (*I2C_CALLBK_T) (uint32_t err_code, uint32_t n)
```

The callback function will be called by the I2C ROM driver upon completion of a task when interrupts are used.

20.4.2 PARAM and RESULT structure

The I2C ROM driver input parameters consist of two structures, a PARAM structure and a RESULT structure. The PARAM structure contains the parameters passed to the I2C ROM driver and the RESULT structure contains the results after the I2C ROM driver is called.

The PARAM structure is as follows:

```
typedef struct i2c_A { //parameters passed to ROM function

 uint32_t num_bytes_send ;
```

```
 uint32_t num_bytes_rec ;

 uint8_t *buffer_ptr_send ;

 uint8_t *buffer_ptr_rec ;

 I2C_CALLBK_T func_pt; // callback function pointer

 uint8_t stop_flag;

 uint8_t dummy[3] ;         // required for word alignment

 } I2C_PARAM ;
```

The RESULT structure is as follows:

```
typedef struct i2c_R {     // RESULTs struct--results are here when returned

 uint32_t n_bytes_sent ;

 uint32_t n_bytes_recd ;

 } I2C_RESULT ;
```

20.4.3 Error structure

The error code returned by the I2C ROM driver is an enum structure. The Error structure is as follows:

```
typedef enum

{

                EM783_OK=0, /**< enum value returned on Success */

                 ERROR,

 ERR_I2C_BASE = 0x00060000,

/*0x00060001*/ ERR_I2C_NAK=ERR_I2C_BASE+1,

/*0x00060002*/ ERR_I2C_BUFFER_OVERFLOW,

/*0x00060003*/ ERR_I2C_BYTE_COUNT_ERR,

/*0x00060004*/ ERR_I2C_LOSS_OF_ARBRITRATION,

/*0x00060005*/ ERR_I2C_SLAVE_NOT_ADDRESSED,

/*0x00060006*/ ERR_I2C_LOSS_OF_ARBRITRATION_NAK_BIT,

/*0x00060007*/ ERR_I2C_GENERAL_FAILURE,

/*0x00060008*/ ERR_I2C_REGS_SET_TO_DEFAULT

} ErrorCode_t;
```

20.4.4 I2C Mode

The i2c_get_status() function returns the current status of the I2C engine. The return codes can be defined as an enum structure:

```
typedef enum I2C_mode {

 IDLE,

 MASTER_SEND,

 MASTER_RECEIVE,

 SLAVE_SEND,

 SLAVE_RECEIVE

 } I2C_MODE_T ;
```

20.4.5 I2C ROM driver pointer

The I2C ROM driver resides in the address 0x1FFF1FF8. The address must be declared to allow access to the ROM driver:

```
#define ROM_DRIVERS_PTR ((ROM *)(*((unsigned int *)0x1FFF1FF8)))
```

20.4.6 API function table

A structure needs to be defined to allow access of the I2C ROM driver:

```
typedef struct I2CD_API {   // index of all the i2c driver functions

void (*i2c_isr_handler) (I2C_HANDLE_T* h_i2c) ; // ISR interrupt service request

// MASTER functions ***

ErrorCode_t (*i2c_master_transmit_poll)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,

                                                    I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_master_receive_poll)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,

                                                  I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_master_tx_rx_poll)(I2C_HANDLE_T* h_i2c,I2C_PARAM* ptp,

                                                I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_master_transmit_intr)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,
     I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_master_receive_intr)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,
     I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_master_tx_rx_intr)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp, I2C_RESULT*
     ptr ) ;

// SLAVE functions ***
```

```
ErrorCode_t (*i2c_slave_receive_poll)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp, I2C_RESULT*
     ptr ) ;

ErrorCode_t (*i2c_slave_transmit_poll)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,
     I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_slave_receive_intr)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp, I2C_RESULT*
     ptr ) ;

ErrorCode_t (*i2c_slave_transmit_intr)(I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp,
     I2C_RESULT* ptr ) ;

ErrorCode_t (*i2c_set_slave_addr)(I2C_HANDLE_T* h_i2c,

                                  uint32_t slave_addr_0_3, uint32_t slave_mask_0_3);

// OTHER functions

uint32_t (*i2c_get_mem_size)(void) ; //ramsize_in_bytes memory needed by I2C drivers

I2C_HANDLE_T*   (*i2c_setup)(uint32_t i2c_base_addr, uint32_t *start_of_ram ) ;

ErrorCode_t   (*i2c_set_bitrate)(I2C_HANDLE_T* h_i2c, uint32_t P_clk_in_hz,

                                 uint32_t bitrate_in_bps) ;

uint32_t (*i2c_get_firmware_version)() ;

I2C_MODE_T (*i2c_get_status)(I2C_HANDLE_T* h_i2c ) ;

} I2CD_API_T ;
```

20.4.7 I2C set-up

Before calling any setup functions in the I2C ROM, the application program is responsible for performing the following:

1. Enable the clock to the I2C peripheral.
2. Enable the two pins required for the SCL and SDA outputs of the I2C peripheral.
3. Allocate a RAM area for dedicated use of the I2C ROM Driver.

After the I2C block is configured, the I2C ROM driver variables must be set up:

1. Initialize pointer to the I2C API function table.
2. Declare the PARAM and RESULT struct.
3. Declare Error Code struct.
4. Declare the transmit and receive buffer.

If interrupts are used, then additional driver variables must be set up:

1. Declare the I2C_CALLBK_ T type.
2. Declare callback functions.
3. Declare I2C ROM Driver ISR within the I2C ISR.
4. Enable I2C interrupt.

20.4.8 I2C Master mode set-up

The I2C ROM Driver support polling and interrupts. In the master mode, 7-bit and 10-bit addressing are supported. The setup is as follows:

1. Allocate SRAM for the I2C ROM Driver by making a call to the i2c_get_mem_size() function.
2. Create the I2C handle by making a call to the i2c_setup() function.
3. Set the I2C operating frequency by making a call to the i2c_set_bitrate() function.

```
pI2cApi = ROM_DRIVERS_PTR->pI2CD; //setup I2C function table pointer

size_in_bytes = pI2cApi->i2c_get_mem_size();

i2c_handle = pI2cApi->i2c_setup(EM783_I2C_BASE, (uint32_t *)&I2C_Handle[0] );

error_code = pI2cApi->i2c_set_bitrate((I2C_HANDLE_T*)i2c_handle, PCLK_in_Hz,
     bps_in_hz);
```

20.4.9 I2C Slave mode set-up

The I2C ROM Driver support polling and interrupts in the slave mode. In the slave mode, only 7-bit addressing is supported. The set-up is as follows:

1. Allocate SRAM for the I2C ROM Driver by making a call to the i2c_get_mem_size() function.
2. Create the I2C handle by making a call to the i2c_setup() function.
3. Set the I2C operating frequency by making a call to the i2c_set_bitrate() function.
4. Set the slave address by making a call to the i2c_set_slave_addr() function.

The I2C ROM driver allows setting up to 4 slave addresses and 4 address masks as well as possibly enabling the General Call address.

The four slave address bytes are packed into the 4 byte variable. Slave address byte 0 is the least significant byte and Slave address byte 3 is the most significant byte. The Slave address mask bytes are ordered the same way as in the other 32 bit variable. When in slave receive mode, all of these addresses (or groups if masks are used) will be monitored for a match. If the General Call bit (least significant bit of any of the four slave address bytes) is set, then the General Call address of 0x00 is also monitored.

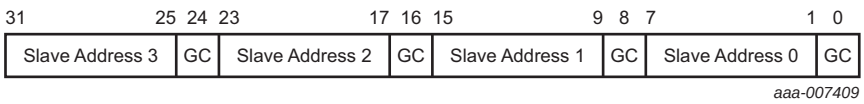


Fig 61. I2C slave mode set-up address packing

```
pI2cApi = ROM_DRIVERS_PTR->pI2CD; //setup I2C function table pointer

size_in_bytes = pI2cApi->i2c_get_mem_size();

i2c_handle = pI2cApi->i2c_setup(EM783_I2C_BASE, (uint32_t *)&I2C_Handle[0] );
```

```
error_code = pI2cApi->i2c_set_bitrate((I2C_HANDLE_T*)i2c_handle, PCLK_in_Hz,
     bps_in_hz);

error_code = pI2cApi->i2c_set_slave_addr((I2C_HANDLE_T*)i2c_handle, slave_addr,
     slave_addr_mask) ;
```

20.4.10 I2C Master Transmit/Receive

The Master mode drivers give the user the choice of either polled (wait for the message to finish) or interrupt driven routines (non-blocking). Polled routines are recommended for testing purposes or very simple I2C applications. These routines allow the Master to send to Slaves with 7-bit or 10-bit addresses.

The following routines are polled routines:

```
err_code i2c_master_transmit_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_master_receive_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_master_tx_rx_poll (I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)
```

The following routines are interrupt driven routines:

```
err_code i2c_master_transmit_intr(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_master_receive_intr(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_master_tx_rx_intr(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)
```

Where:

- err_code is the return state of the function. An “0” indicates success. All non-zero indicates an error. Refer to Error Table.
- I2C_PARM* is a structure with parameters passed to the function. Refer to Section 20.4.2.
- I2C_RESULT* is a containing the results after the function executes.

To initiate a master mode write/read the I2C_PARAM has to be setup. The I2C_PARAM is a structure with various variables needed by the I2C ROM Driver to operate correctly. The structure contains the following:

- Number of bytes to be transmitted.
- Number of bytes to be receive.
- Pointer to the transmit buffer.
- Pointer to the receive buffer.
- Pointer to callback function.
- Stop flag.

The RESULT structure contains the results after the function executes. The structure contains the following:

- Number of bytes transmitted.
- Number of bytes received.

Remark: The number of bytes transmitted will be updated for i2c_master_transmit_intr() and i2c_master_transmit_poll(). The number of bytes received will only be updated on i2c_master_receive_poll(), i2c_master_receive_intr(), i2c_master_tx_rx_poll(), and i2c_master_tx_rx_intr().

In all the master mode routines, the transmit buffer's first byte must be the slave address with the R/W bit set to "0". To enable a master read, the receive buffer's first byte must be the slave address with the R/W bit set to "1".

The following conditions must be fulfilled to use the I2C driver routines in master mode:

- For 7-bit addressing, the first byte of the send buffer must have the slave address in the most significant 7 bits and the least significant (R/W) bit = 0. Example: Slave address 0x53, first byte is 0xA6.
- For 7-bit addressing, the first byte of the receive buffer must have the slave address in the most significant 7 bits and the least significant (R/W) bit = 1. Example: Slave Addr 0x53, first byte 0xA7.
- For 10-bit address, the first byte of the transmit buffer must have the slave address most significant 2 bits with the (R/W) bit =0. The second byte must contain the remaining 8-bit of the slave address.
- For 10-bit address, the first byte of the receive buffer must have the slave address most significant 2 bits with the (R/W) bit =1. The second byte must contain the remaining 8-bit of the slave address.
- The number of bytes to be transmitted should include the first byte of the buffer which is the slave address byte. Example: 2 data bytes + 7-bit slave addr = 3.
- The application program must enable I2C interrupts. When I2C interrupt occurs, the i2c_isr_handler function must be called from the application program.

When using the interrupt function calls, the callback functions must be define. Upon the completion of a read/write as specified by the PARAM structure, the callback functions will be invoked.

20.4.11 The I2C Slave Mode Transmit/Receive

In slave mode, polled routines are intended for testing purposes. It is up to the user to decide whether to use the polled or interrupt driven mode. While operating the Slave driver in polled mode can be useful for program development and debugging, most applications will need the interrupt-driven versions of Slave Receive and Transmit in the final software.

The following routines are polled routines:

```
err_code i2c_slave_receive_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_slave_receive_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_slave_transmit_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)
```

The following routines are interrupt driven routines:

```
err_code i2c_slave_receive_intr(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)

err_code i2c_slave_transmit_intr(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*)
```

Where:

- err_code is the return state of the function. An 0 indicates success. All non-zero indicates an error. Refer to the Error Code Table.
- I2C_PARM is a structure with parameters passed to the function. Section 20.4.2.
- I2C_RESULT is a containing the results after the function executes. Section 20.4.2.

To initiate a master-mode write/read the I2C_PARAM has to be setup. The I2C_PARAM is a structure with various variables needed by the I2C ROM Driver to operate correctly. The structure contains the following:

- Number of bytes to be transmitted.
- Number of bytes to be received.
- Pointer to the transmit buffer.
- Pointer to the receive buffer.
- Pointer to callback function.
- Stop flag.

The RESULT structure contains the results after the function executes. The structure contains the following:

- Number of bytes transmitted.
- Number of bytes received.

Remark: The number of bytes transmitted is updated only for i2c_slave_send_poll() and i2c_slave_send_intr(). The number of bytes received is updated only for i2c_slave_receive_poll() and i2c_slave_receive_intr().

To initiate a slave mode communication, the receive function is called. This can be either the polling or interrupt driven function, i2c_slave_receive_poll() or i2c_slave_receive_intr(), respectively. The receive buffer should be as large or larger than any data or command that will be received. If the amount of data exceed the receive buffer size, an error code will be returned.

In slave-receive mode, the driver receives data until one of the following are true:

- Address matching set in the set_slave_addr() function with the R/W bit set to 1
- STOP or repeated START is received
- An error condition is detected

When using the interrupt function calls, the callback functions must be define. Upon the completion of a read/write as specified by the PARAM structure, the callback functions will be invoked.

21.1 How to read this chapter

The power profiles are available on all EM783 parts.

21.2 Features

- Power Management services
- Clocking services

21.3 General description

The API calls to the ROM are performed by executing functions which are pointed by a pointer within the ROM Driver Table. Figure 62 shows the pointer structure used to call the Power Profiles API.

Remark: Disable all interrupts before making calls to the power profile API. You can re-enable the interrupts after the power profile API calls have completed.

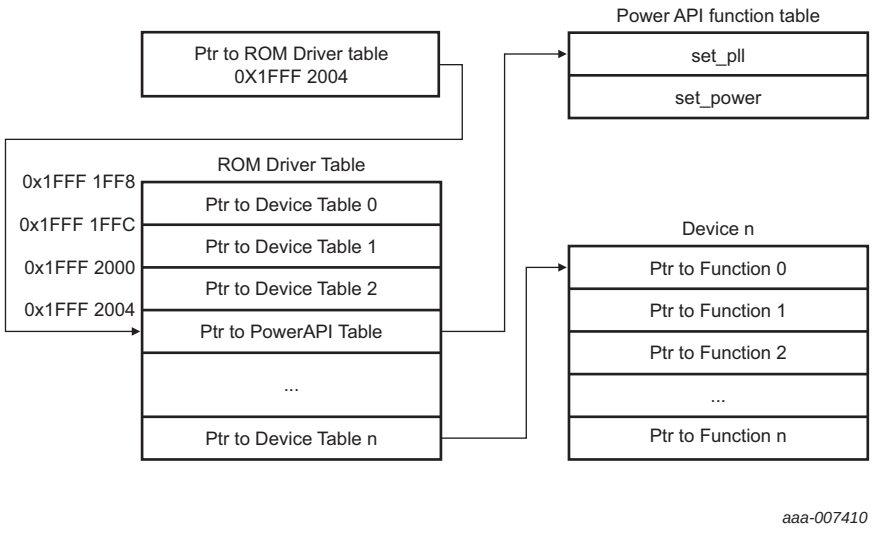


Fig 62. Power profiles pointer structure

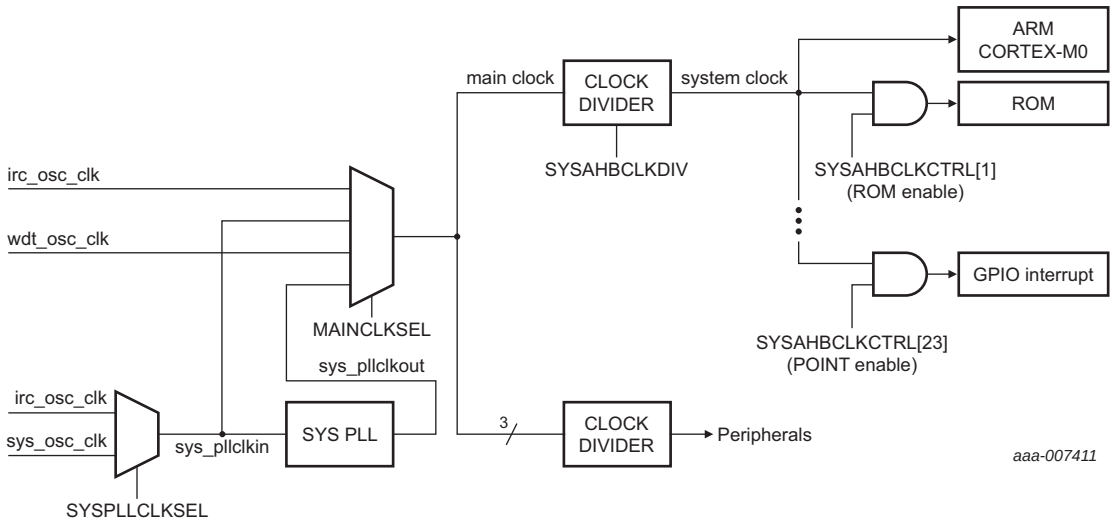


Fig 63. Clock configuration for power API use

21.4 Definitions

The following elements must 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 unsigned p_dev1;
     const unsigned p_dev2;
     const unsigned p_dev3;
     const PWRD *pPWRD;
     const EM783_ROM_DIV_STUCT * pROMDiv;
     const unsigned p_dev4;
     const unsigned p_dev5;
     const unsigned p_dev6;

} ROM;
ROM ** rom = (ROM **) 0x1FFF1FF8;
unsigned int command[4], result[2];
```

21.5 Clocking routine

21.5.1 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 14), the main clock source must be set to the input clock to the system PLL (Table 16) and the system/AHB clock divider must be set to 1 (Table 18).

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 peripheral clocks in the device.

Table 254. 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 63. For more details see Figure 3.

21.5.1.1 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 48 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.

21.5.1.2 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*.

21.5.1.3 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.

21.5.1.4 Code examples

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

21.5.1.4.1 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 48 MHz. Therefore *set_pll* returns PLL_INVALID_FREQ in *result[0]* and 12000 in *result[1]* without changing the PLL settings.

21.5.1.4.2 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.

21.5.1.4.3 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.

21.5.1.4.4 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.

21.5.1.4.5 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.

21.5.1.4.6 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.

21.6 Power routine

21.6.1 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 18 and Figure 63). 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.

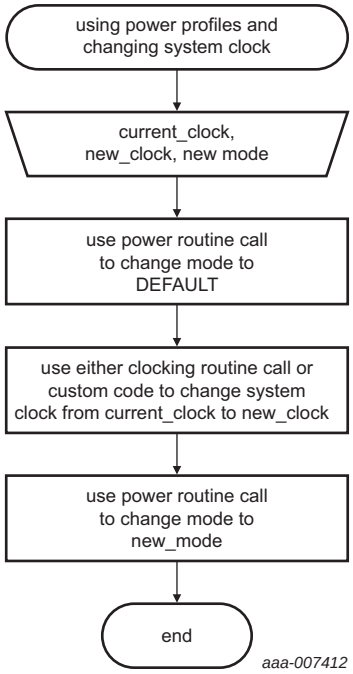


Fig 64. Power profiles usage

Table 255. 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 63. For more details see Figure 3.

21.6.1.1 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 48 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.

21.6.1.2 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.

21.6.1.3 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 48 MHz inclusive.

21.6.1.4 Code examples

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

21.6.1.4.1 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 48 MHz maximum, *set_power* returns PWR_INVALID_FREQ in *result[0]* without changing anything in the existing power setup.

21.6.1.4.2 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.

22.1 Features

- Supports ARM Serial Wire Debug mode.
- The debug functionality is identical for all EM783 parts.
- Direct debug access to all memories, registers, and peripherals.
- No target resources are required for the debugging session.
- Four breakpoints.
- Two data watchpoints that can also be used as triggers.

22.2 General description

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.

22.3 Pin description

Table 256 indicates 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. To maximize I/O flexibility, each of these signals can be on either of two pins. See Table 68 and Table 70 for pinning information. For debugging, exactly one pin must be selected as SWCLK and exactly one as SWDIO.

Table 256. Serial Wire Debug pin description

Pin Name	Type	Description
SWCLK	Input	**Serial Wire Clock.** This pin is the clock for debug logic when in the Serial Wire Debug mode (SWDCLK). For the SWD function, a pull-up resistor is recommended on this pin.
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 EM783. For the SWD function, a pull-up resistor is recommended on this pin. **Note**: for SP/TP/MC3/MC6 variants user should keep an option to use pin 11 or pin 25 for SWDIO.

22.4 Functional description

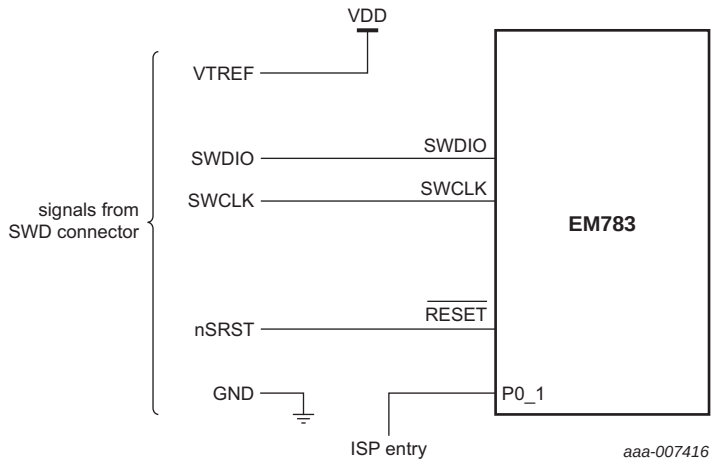
22.4.1 Systick timer in debug mode

During a debugging session, the System Tick Timer is automatically stopped whenever the CPU is stopped. Other peripherals are not affected.

22.4.2 Debug connections for SWD

For debugging purposes, it is useful to provide access to the ISP entry pin P0_1. This pin can be used to recover the part from configurations which would disable the SWD port

such as improper PLL configuration or re-configuration of SWD pins as metrology inputs. 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 65. Connecting the SWD pins to a standard SWD connector

23.1 Introduction

The following material is using the ARM *Cortex-M0 User Guide*. The following minor changes have been made regarding the specific implementation of the Cortex-M0 for the EM783:

- Deep-sleep mode not implemented.

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.

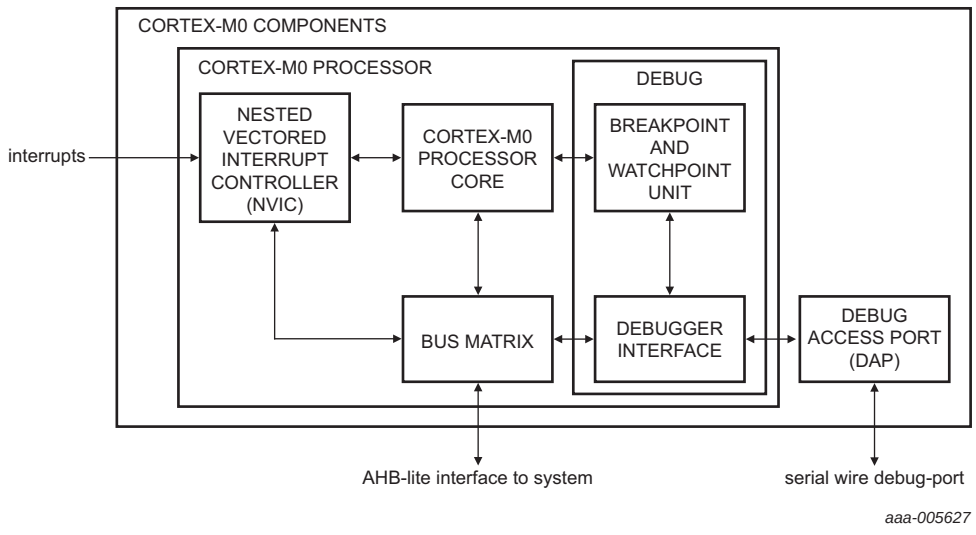


Fig 66. Cortex-M0 implementation

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:

- 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 that 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 257. 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:

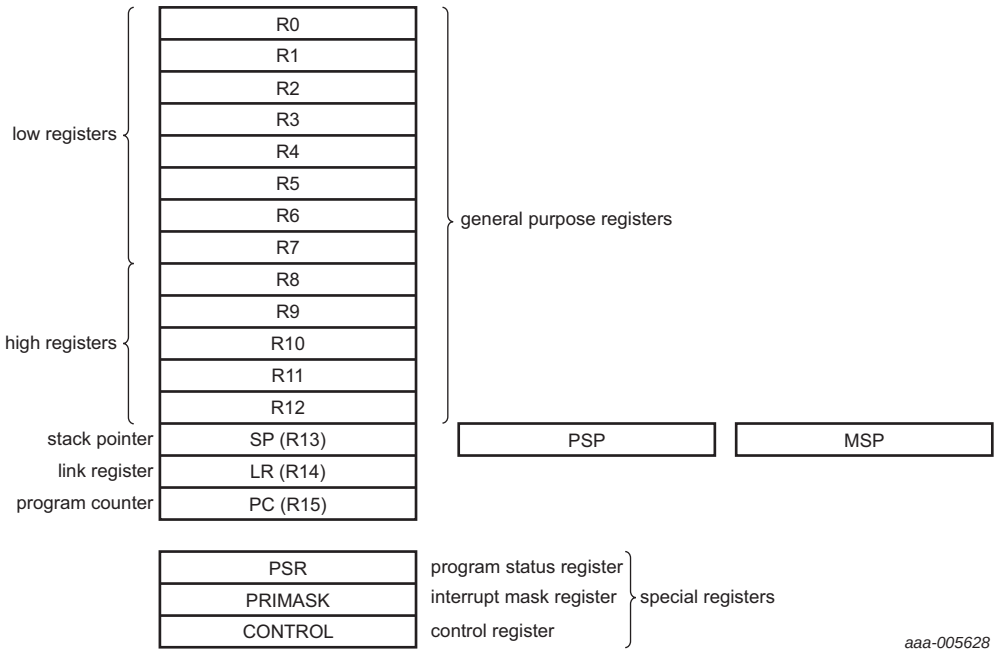


Fig 67. Processor core register set

Table 258. 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 23–259
APSR	RW	Unknown	Table 23–260
IPSR	RO	0x00000000	Table 261
EPSR	RO	Unknown[2]	Table 23–262
PRIMASK	RW	0x00000000	Table 23–263
CONTROL	RW	0x00000000	Table 23–264

[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 bit fields in the 32-bit PSR. The PSR bit assignments are:

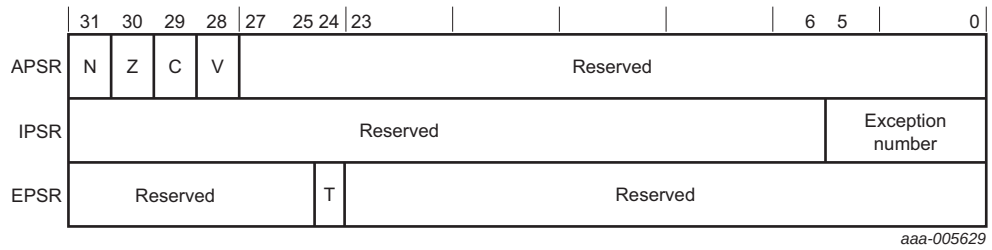


Fig 68. APSR, IPSR, EPSR register bit assignments

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 259. 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 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 23–258 for its attributes. The bit assignments are:

Table 260. 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 23–258 for its attributes. The bit assignments are:

Table 261. 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 23–258 for the EPSR attributes. The bit assignments are:

Table 262. 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 23–258 for its attributes. The bit assignments are:

Table 263. 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 23–258 for its attributes. The bit assignments are:

Table 264. 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 4 GB of addressable memory. The memory map is shown in Figure 2 on page 9.

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 reorder 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 that 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 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 reordering 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	-	<	<	<

aaa-005630

Fig 69. 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 265. Memory access behavior

Address range	Memory region	Memory type[1]	XN[1]	Description
0x00000000-0x0FFFFFFF	Code	Normal	-	Executable region for program code. You can also put data here.
0x10000000-0x3FFFFFFF	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:

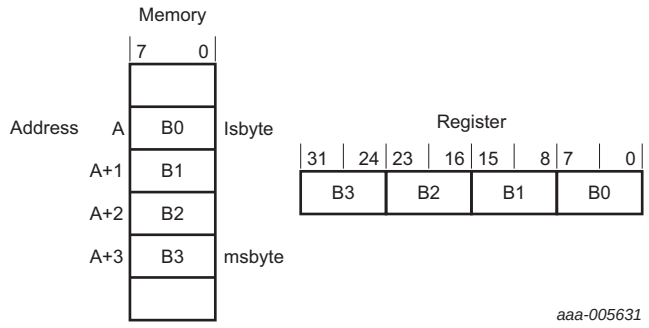


Fig 70. Little endian format

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 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 266. 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 266. Properties of different exception types *…continued*

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 23–261.

[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 23–266 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 23–71 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.

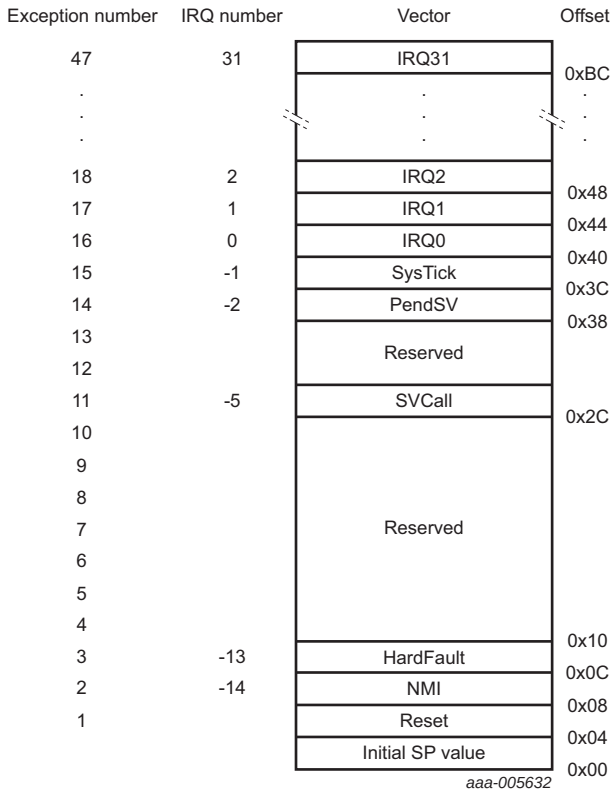


Fig 71. Vector table

The vector table is fixed at address `0x00000000`.

23.3.3.5 Exception priorities

As Table 23–266 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.

Remark: Configurable priority values are in the range **0-3**. 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 that 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 as a **stack frame**. The stack frame contains the following information:

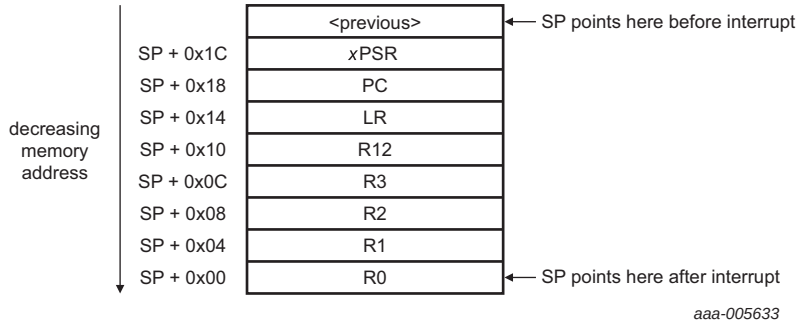


Fig 72. Exception entry stack contents

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 23–267 shows.

Table 267. 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 mode. The deep-sleep mode is not implemented on the EM783.

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 wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back 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 EM783.

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 Wakeup from sleep mode

The conditions for the processor to wakeup depend on the mechanism that caused it to enter sleep mode.

23.3.5.2.1 Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1. 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 Wakeup from WFE

Remark: The WFE instruction is not implemented on the EM783.

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 268 lists the supported instructions.

Remark: In Table 268

- 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 268. 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
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

Table 268. Cortex-M0 instructions *...continued*

Mnemonic	Operands	Brief description	Flags	Reference
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
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, it may be necessary 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 269. 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)
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 270. Intrinsic functions for accessing special registers via MRS/MSR

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 to 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 subsections 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 and into the right hand 32 -*n* bits of the result. It also copies the original bit[31] of the register into the leftmost *n* bits of the result. See Figure 73.

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*.

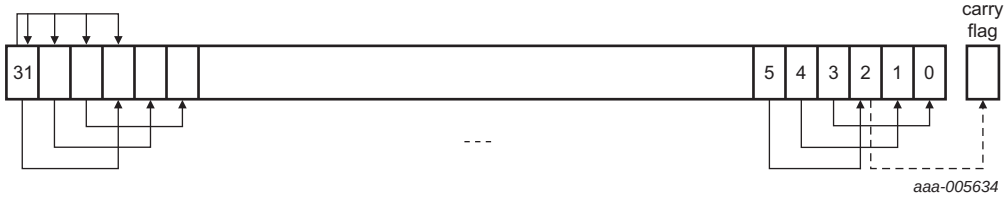


Fig 73. ASR #3

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 leftmost *n* bits of the result to 0. See Figure 74.

If the value is regarded as an unsigned integer, use the LSR operation to divide the value in the register *Rm* by 2^n.

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.

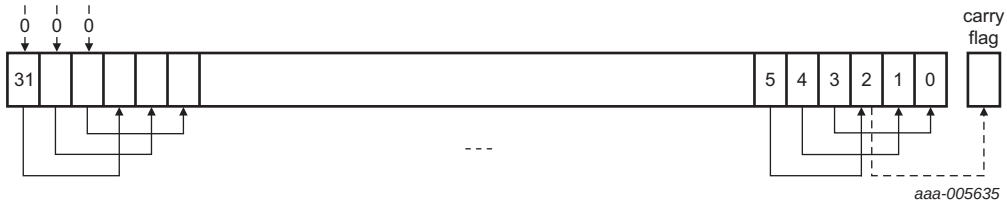


Fig 74. LSR #3

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 rightmost *n* bits of the result to 0. See Figure 75.

If the value is regarded as an unsigned integer or a two's complement signed integer, use the LSL operation to multiply the value in the register *Rm* by 2^n. 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.

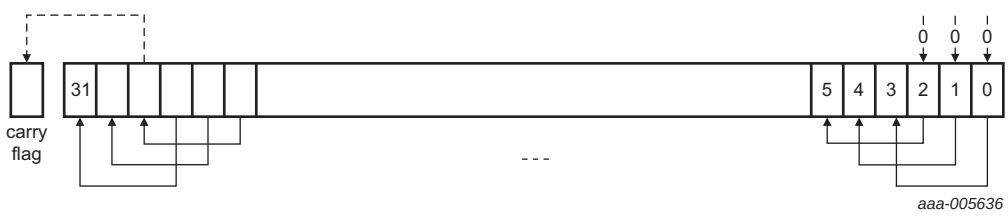


Fig 75. LSL #3

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 rightmost *n* bits of the register into the leftmost *n* bits of the result. See Figure 76.

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.

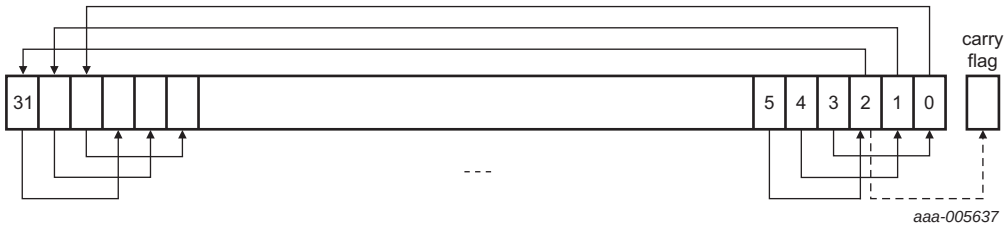


Fig 76. ROR #3

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 great, 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 23.4.3.6.1. 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. Table 271 shows the condition codes to use.

Table 271 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 271. 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
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 272 shows the memory access instructions:

Table 272. 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, 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

LDRSH  R1, [R2, R3]         ; 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 273 shows the data processing instructions:

Table 273. 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 274 lists the legal combinations of register specifiers and immediate values that can be used with each instruction.

Table 274. 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 do 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 275 shows the branch and control instructions:

Table 275. Branch and control instructions

Mnemonic	Brief description	See
B{cc}	Branch {conditionally}	Section 23–23.4.6.1

Table 275. Branch and control instructions

Mnemonic	Brief description	See
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 276 shows the ranges for the various branch instructions.

Table 276. 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 an 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 277 shows the remaining Cortex-M0 instructions:

Table 277. 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 completion.

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

Remark: The SEV instruction is not implemented on the EM783.

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 EM783.

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 an NOP operation.

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 an 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 278. Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System Control Block	Table 23–287
0xE000E010-0xE000E01F	System timer	Table 23–296
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 23–279
0xE000ED00-0xE000ED3F	System Control Block	Table 23–287
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 23–279

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 279. 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 280. CMSIS access to 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 266 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 279 for the register attributes.

The bit assignments are according to the IRQ numbers in Table 36.

Table 281. 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 23–279 for the register attributes.

The bit assignments are according to the IRQ numbers in Table 36.

Table 282. 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 23–279 for the register attributes.

The bit assignments are according to the IRQ numbers in Table 36.

Table 283. 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 23–279 for the register attributes.

The bit assignments are according to the IRQ numbers in Table 36.

Table 284. 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 a priority field for each interrupt. These registers are only word-accessible. See the register summary in Table 23–279 for their attributes. Each register holds four priority fields as shown, and are arranged according to the IRQ numbers in Table 36.

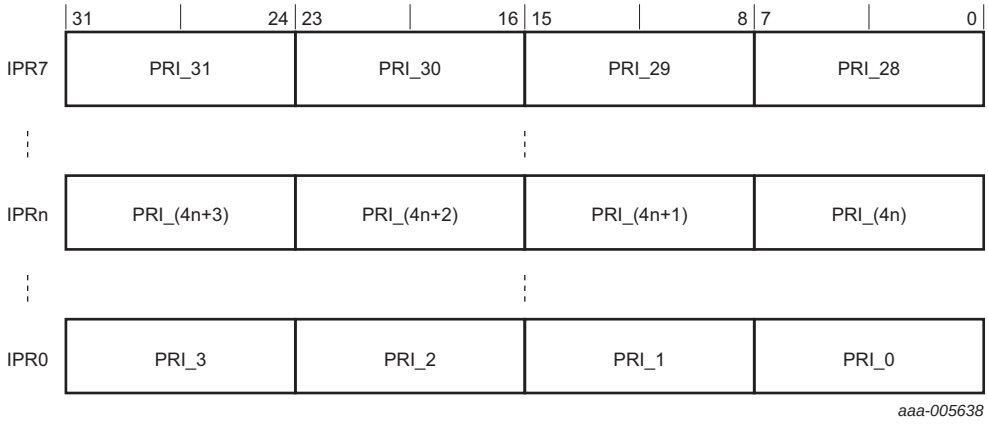


Fig 77. IPR register

Table 285. IPR bit assignments

Bits	Name	Function
[31:24]	Priority, byte offset 3	Each priority field holds a priority value 0-3 in bit [7:6]. The lower the value, the greater the priority of the corresponding interrupt. Bits [5:0] of each field read as zero and are ignored on 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** = **M** 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 that 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 that 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 286. 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 23–266 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 287. 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

Table 287. Summary of the SCB registers

Address	Name	Type	Reset value	Description
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 CMSIS mapping of 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 288. 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: 0x0 = Revision 0
[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: 0x0 = Patch 0

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 23–287 for the ICSR attributes. The bit assignments are:

Table 289. 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 289. ICSR bit assignments *...continued*

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 23–261.

[1] This is the same value as IPSR bits[5:0], see Table 23–261.

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/Reset Control Register

The AIRCR provides endian status for data accesses and reset control of the system. See the register summary in Table 23–287 and Table 23–290 for its attributes.

To write to this register, write `0x05FA` to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

Table 290. 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 290. AIRCR bit assignments *...continued*

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 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 23–287 for its attributes. The bit assignments are:

Table 291. 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 (Deep-sleep mode is not supported on the EM783): 0 = sleep 1 = deep sleep (not supported on the EM783; do not use this setting)
[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 (Deep-sleep is not supported on the EM783). 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 23–287 for the CCR attributes.

The bit assignments are:

Table 292. 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 23–287 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 23–266 for more information.

The system fault handlers, and the priority field and register for each handler are:

Table 293. 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 294. 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 295. 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 296. 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 297. 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 297. SYST_CSR bit assignments *...continued*

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 23–296 for its attributes. The bit assignments are:

Table 298. 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 23–296 for its attributes. The bit assignments are:

Table 299. 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 23–296 for its attributes. The bit assignments are:

Table 300. 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 that 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. If this clock signal is stopped for low power mode, the SysTick counter stops.

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 301. 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 301. Cortex M0- instruction summary *…continued*

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 register	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 301. Cortex M0- instruction summary *…continued*

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 halfword	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 that 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 spent waiting for an interrupt or event.

[6] Executes as NOP.

24.1 Abbreviations

Table 302. Abbreviations

Acronym	Description
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
BOD	BrownOut Detection
DAC	Digital-to-Analog Converter
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First In, First Out
GPIO	General Purpose Input/Output
I^2C or IIC	Inter-Integrated Circuit bus
I/O or IO	Input/Output
IAP	In-Application Programming
ISP	In-System Programming
ISR	Interrupt Service Routine
LFO	Low Frequency Oscillator
NVIC	Nested Vectored Interrupt Controller
PIO	Parallel Input/Output
PLL	Phase-Locked Loop
POR	Power-On Reset
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface
SSI	Serial Synchronous Interface
SWD	Serial Wire Debug
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
WWDOsc	Windowed WatchDog Oscillator
WWDT	Windowed WatchDog Timer

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

Table 1. Ordering information 5
Table 2. Ordering options 5
Table 3. Pin summary. 10
Table 4. Register overview: system control block (base address 0x4004 8000) 12
Table 5. System memory remap register (SYSMEMREMAP, address 0x4004 8000) bit description 13
Table 6. Peripheral reset control register (PRESETCTRL, address 0x4004 8004) bit description 13
Table 7. System PLL control register (SYSPLLCTRL, address 0x4004 8008) bit description 14
Table 8. System PLL status register (SYSPLLSTAT, address 0x4004 800C) bit description 14
Table 9. System oscillator control register (SYSOSCCTRL, address 0x4004 8020) bit description. 15
Table 10. Watchdog oscillator control register (WDTOSCCTRL, address 0x4004 8024) bit description 15
Table 11. High-frequency oscillator control register (IRCCTRL, address 0x4004 8028) bit description 16
Table 12. LF oscillator control register (LFOSCCTRL, address 0x4004 802C)) bit description 17
Table 13. System reset status register (SYSRSTSTAT, address 0x4004 8030) bit description 17
Table 14. System PLL clock source select register (SYSPLLCLKSEL, address 0x4004 8040) bit description 18
Table 15. System PLL clock source update enable register (SYSPLLCLKUEN, address 0x4004 8044) bit description 18
Table 16. Main clock source select register (MAINCLKSEL, address 0x4004 8070) bit description. 19
Table 17. Main clock source update enable register (MAINCLKUEN, address 0x4004 8074) bit description 19
Table 18. System clock divider register (SYSAHBCLKDIV, address 0x4004 8078) bit description. 19
Table 19. System clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description 20
Table 20. SSP0 clock divider register (SSP0CLKDIV, address 0x4004 8094) bit description. 21
Table 21. UART clock divider register (UARTCLKDIV, address 0x4004 8098) bit description. 22
Table 22. CLKOUT clock source select register (CLKOUTSEL, address 0x4004 80E0) bit description 22
Table 23. CLKOUT clock source update enable register (CLKOUTUEN, address 0x4004 80E4) bit description 22
Table 24. CLKOUT clock divider registers (CLKOUTDIV, address 0x4004 80E8) bit description 23
Table 25. POR captured PIO status register 0 (PIOPORCAP0, address 0x4004 8100) bit description 23
Table 26. BrownOut Detect register (BODR, address 0x4004 8150) bit description 23
Table 27. System tick timer calibration register (SYSTCKCAL, address 0x4004 8158) bit description 24
Table 28. NMI source selection register (NMISRC, address 0x4004 8174) bit description 24
Table 29. Pin interrupt select registers (PINTSEL0 to 7, address 0x4004 8178 to 0x4004 8194) bit description 24
Table 30. Power configuration register (PDRUNCFG, address 0x4004 8238) bit description 25
Table 31. Device ID register (DEVICE_ID, address 0x4004 83F4) bit description 26
Table 32. Device configuration register (DEVICECFG, address 0x4004 83FC) bit description 26
Table 33. EM783 power and clock control registers 30
Table 34. PLL operating modes. 32
Table 35. PLL frequency parameters. 32
Table 36. Connection of interrupt sources to the Vectored Interrupt Controller 34
Table 37. Register overview: GPIO pin interrupts (base address: 0x4004 C000) 38
Table 38. Register overview: GPIO GROUP0 interrupt (base address: 0x4005 C000) 38
Table 39. Register overview: GPIO GROUP1 interrupt (base address: 0x4006 0000) 38
Table 40. Register overview: GPIO port (base address: 0x5000 0000) 38
Table 41. Pin interrupt mode register (ISEL, address 0x4004 C000) bit description 39
Table 42. Pin interrupt level (rising edge) interrupt enable register (IENR, address 0x4004 C004) bit description 39
Table 43. Pin interrupt level (rising edge) interrupt set register (SIENR, address 0x4004 C008) bit description 40
Table 44. Pin interrupt level (rising edge interrupt) clear register (CIENR, address 0x4004 C00C) bit description 40
Table 45. Pin interrupt active level (falling edge) interrupt enable register (IENF, address 0x4004 C010) bit description 41
Table 46. Pin interrupt active level (falling edge interrupt) set register (SIENF, address 0x4004 C014) bit description 41
Table 47. Pin interrupt active level (falling edge) interrupt clear register (CIENF, address 0x4004 C018) bit description 42
Table 48. Pin interrupt rising edge register (RISE, address 0x4004 C01C) bit description 42
Table 49. Pin interrupt falling edge register (FALL, address 0x4004 C020) bit description 42
Table 50. Pin interrupt status register (IST address 0x4004 C024) bit description 43

Table 51. GPIO grouped interrupt control register (CTRL, addresses 0x4005 C000 (GROUP0 INT) and 0x4006 0000 (GROUP1 INT)) bit description . . 43
Table 52. GPIO grouped interrupt port 0 polarity registers (PORT_POL0, addresses 0x4005 C020 (GROUP0 INT) and 0x4006 0020 (GROUP1 INT)) bit description 44
Table 53. GPIO grouped interrupt port 0 enable registers (PORT_ENA0, addresses 0x4005 C040 (GROUP0 INT) and 0x4006 0040 (GROUP1 INT)) bit description 44
Table 54. GPIO port 0 byte pin registers (B0 to B26, addresses 0x5000 0000 to 0x5000 001A) bit description 44
Table 55. GPIO port 0 word pin registers (W0 to W31, addresses 0x5000 1000 to 0x5000 1068) bit description 45
Table 56. GPIO direction port 0 register (DIR0, address 0x5000 2000) bit description 45
Table 57. GPIO mask port 0 register (MASK0, address 0x5000 2080) bit description 45
Table 58. GPIO port 0 pin register (PIN0, address 0x5000 2100) bit description 45
Table 59. GPIO masked port 0 pin register (MPIN0, address 0x5000 2180) bit description 46
Table 60. GPIO set port 0 register (SET0, address 0x5000 2200) bit description 46
Table 61. GPIO clear port 0 register (CLR0, address 0x5000 2280) bit description 46
Table 62. GPIO toggle port 0 register (NOT0, address 0x5000 2300) bit description 46
Table 63. Pin interrupt registers for edge- and level-sensitive pins 48
Table 64. Register overview: I/O configuration (base address 0x4004 4000) 53
Table 65. Type D IOCON registers bit description 54
Table 66. Type D IOCON registers: FUNC values and pin functions 55
Table 67. Type I IOCON registers bit description 56
Table 68. Type I IOCON registers: FUNC values and pin functions 56
Table 69. Type A IOCON registers bit description 56
Table 70. Type A IOCON registers: FUNC values and pin functions 57
Table 71. Pin multiplexing 58
Table 72. EM783 pin description 61
Table 73. USART pin description 67
Table 74. Register overview: USART (base address: 0x4000 8000) 68
Table 75. USART Receiver Buffer Register when DLAB = 0 (RBR - address 0x4000 8000) bit description . 69
Table 76. USART Transmitter Holding Register when DLAB = 0 (THR - address 0x4000 8000) bit description 69
Table 77. USART Divisor Latch LSB Register when DLAB = 1 (DLL - address 0x4000 8000) bit description 70
Table 78. USART Divisor Latch MSB Register (DLM - address 0x4000 8004 when DLAB = 1) bit description 70
Table 79. USART Interrupt Enable Register when DLAB = 0 (IER - address 0x4000 8004) bit description . . 70
Table 80. USART Interrupt Identification Register (IIR - address 0x4004 8008, RO) bit description . . . 71
Table 81. USART Interrupt Handling 72
Table 82. USART FIFO Control Register (FCR - address 0x4000 8008, Write Only) bit description 73
Table 83. USART Modem Control Register (MCR - address 0x4000 8010) bit description 74
Table 84. Modem status interrupt generation 76
Table 85. USART Line Control Register (LCR - address 0x4000 800C) bit description 76
Table 86. USART Line Status Register (LSR - address 0x4000 8014, Read Only) bit description 77
Table 87: USART Modem Status Register (MSR - address 0x4000 8018) bit description 79
Table 88. USART Scratch Pad Register (SCR - address 0x4000 801C) bit description 80
Table 89. Auto-baud Control Register (ACR - address 0x4000 8020) bit description 80
Table 90: IrDA Control Register (ICR - 0x4000 8024) bit description 83
Table 91: IrDA Pulse Width 84
Table 92. USART Fractional Divider Register (FDR - address 0x4000 8028) bit description 85
Table 93. Fractional Divider setting look-up table 86
Table 94. USART Oversampling Register (OSR - address 0x4000 802C) bit description 88
Table 95. USART Transmit Enable Register (TER - address 0x4000 8030) bit description 88
Table 96. Smart Card Interface Control register (SCICTRL - address 0x4000 8048) bit description 89
Table 97. USART RS485 Control register (RS485CTRL - address 0x4000 804C) bit description 90
Table 98. USART RS-485 Address Match register (RS485ADRMATCH - address 0x4000 8050) bit description 91
Table 99. USART RS-485 Delay value register (RS485DLY - address 0x4000 80454) bit description 91
Table 100. USART Synchronous mode control register (SYNCCTRL - address 0x4000 8058) bit description 93
Table 101. I^2C-bus pin description 99
Table 102. Register overview: I^2C (base address 0x4000 0000) 99
Table 103. I^2C Control Set register (CONSET - address 0x4000 0000) bit description 100
Table 104. I^2C Status register (STAT - 0x4000 0004) bit description 102
Table 105. I^2C Data register (DAT - 0x4000 0008) bit description 102
Table 106. I^2C Slave Address register 0 (ADR0- 0x4000 000C) bit description 102

Table 107. I2C SCL HIGH Duty Cycle register (SCLH - address 0x4000 0010) bit description 103
Table 108. I2C SCL Low duty cycle register (SCLL - 0x4000 0014) bit description 103
Table 109. SCLL + SCLH values for selected I2C clock values 103
Table 110. I2C Control Clear register (CONCLR - 0x4000 0018) bit description 104
Table 111. I2C Monitor mode control register (MMCTRL - 0x4000 001C) bit description 104
Table 112. I2C Slave Address registers (ADR[1, 2, 3]- 0x4000 00[20, 24, 28]) bit description 106
Table 113. I2C Data buffer register (DATA_BUFFER - 0x4000 002C) bit description 106
Table 114. I2C Mask registers (MASK[0, 1, 2, 3] - 0x4000 00[30, 34, 38, 3C]) bit description . . . 107
Table 115. CONSET used to configure Master mode . . . 112
Table 116. CONSET used to configure Slave mode 113
Table 117. Abbreviations used to describe I2C operation 115
Table 118. CON for Master Transmitter mode 115
Table 119. Master Transmitter mode 116
Table 120. Master Receiver mode 119
Table 121. ADR usage in Slave Receiver mode 121
Table 122. CON for Slave Receiver mode. 121
Table 123. Slave Receiver mode 121
Table 124. Slave Transmitter mode 125
Table 125. Miscellaneous States 127
Table 126. SSP pin descriptions 139
Table 127. Register overview: SPI0 (base address 0x4004 0000) 140
Table 128: SPI/SSP Control Register 0 (CR0 - address 0x4004 0000 (SSP0)) bit description 140
Table 129: SPI/SSP Control Register 1 (CR1 - address 0x4004 0004 (SSP0)) bit description 141
Table 130: SPI/SSP Data Register (DR - address 0x4004 0008 (SSP0)) bit description 142
Table 131: SPI/SSP Status Register (SR - address 0x4004 000C (SSP0)) bit description 142
Table 132: SPI/SSP Clock Prescale Register (CPSR - address 0x4004 0010 (SSP0)) bit description 143
Table 133: SPI/SSP Interrupt Mask Set/Clear register (IMSC - address 0x4004 0014 (SSP0)) bit description 143
Table 134: SPI/SSP Raw Interrupt Status register (RIS - address 0x4004 0018 (SSP0)) bit description 144
Table 135: SPI/SSP Masked Interrupt Status register (MIS - address 0x4004 001C (SSP0)) bit description 144
Table 136: SPI/SSP interrupt Clear Register (ICR - address 0x4004 0020 (SSP0)) bit description 144
Table 137. Counter/timer pin description 154
Table 138. Register overview: 16-bit counter/timer CT16B0 (base address 0x4000 C000) 154
Table 139. Interrupt Register (IR - address 0x4000 C000) bit description 156
Table 140. Timer Control Register (TCR, address 0x4000 C004) bit description 156
Table 141: Timer counter registers (TC, address 0x4000 C008) bit description 156
Table 142: Prescale registers (PR, address 0x4000 C00C) bit description 157
Table 143: Prescale counter registers (PC, address 0x4000 C010) bit description 157
Table 144. Match Control Register (MCR, address 0x4000 C014) bit description 157
Table 145: Match registers (MR0 to 3, addresses 0x4000 C018 to 24) bit description 158
Table 146. Capture Control Register (CCR, address 0x4000 C028) bit description 159
Table 147. Capture registers (CR0 to 3, addresses 0x4000 C02C to 0x4000 C034) bit description 160
Table 148. External Match Register (EMR, address 0x4000 C03C) bit description 161
Table 149. External match control 162
Table 150. Count Control Register (CTCR, address 0x4000 C070) bit description 163
Table 151. PWM Control Register (PWMC, address 0x4000 C074) bit description 164
Table 152. Counter/timer pin description 169
Table 153. Register overview: 32-bit counter/timer CT32B0 (base address 0x4001 4000) 169
Table 154: Interrupt Register (IR, address 0x4001 4000 (CT32B0) and IR, address 0x4001 8000) bit description 171
Table 155. Timer Control Register (TCR, address 0x4001 4004 (CT32B0)) bit description 171
Table 156: Timer counter registers (TC, address 0x4001 4008 (CT32B0)) bit description 171
Table 157: Prescale registers (PR, address 0x4001 400C (CT32B0)) bit description 172
Table 158: Prescale registers (PC, address 0x4001 4010 (CT32B0)) bit description 172
Table 159: Match Control Register (MCR, address 0x4001 4014 (CT32B0)) bit description 172
Table 160: Match registers (MR0 to 3, addresses 0x4001 4018 to 24 (CT32B0)) bit description 173
Table 161. Capture Control Register (CCR, address 0x4001 4028 (CT32B0)) bit description 174
Table 162: Capture registers (CR0 to 3, addresses 0x4001 402C to 0x4001 4038 (CT16B0)) bit description 175
Table 163: External Match Register (EMR, address 0x4001 403C (CT32B0)) bit description 176
Table 164. External match control 177
Table 165: Count Control Register (CTCR, address 0x4001 4070 (CT32B0)) bit description 178
Table 166. PWM Control Register (PWMC, 0x4001 4074 (CT32B0)) bit description 179
Table 167. SysTick timer register map (base address 0xE000 E000) 184
Table 168. System Timer Control and status register (STCTRL - 0xE000 E010) bit description 184
Table 169. System Timer Reload value register (STRELOAD - 0xE000 E014) bit description 185
Table 170. System Timer Current value register (STCURR - 0xE000 E018) bit description 185

Table 171. System Timer Calibration value register (STCALIB - 0xE000 E01C) bit description . . . 186
Table 172. Register overview: Watchdog timer (base address 0x4000 4000) 189
Table 173. Watchdog Mode register (WDMOD - 0x4000 4000) bit description 189
Table 174. Watchdog operating modes selection 190
Table 175. Watchdog Timer Constant register (WDTC - 0x4000 4004) bit description 191
Table 176. Watchdog Feed register (WDFEED - 0x4000 4008) bit description 191
Table 177. Watchdog Timer Value register (WDTV - 0x4000 400C) bit description. 191
Table 178. Watchdog Clock Select register (WDCLKSEL - 0x4000 4010) bit description 192
Table 179. Watchdog Timer Warning Interrupt register (WDWARNINT - 0x4000 4014) bit description 192
Table 180. Watchdog Timer Window register (WDWINDOW - 0x4000 4018) bit description 192
Table 181. Metrology Engine pin description. 196
Table 182. Metrology Engine configuration parameters . 200
Table 183. Metrology result class structure 206
Table 184. D/A pin description. 207
Table 185. Register overview: DAC (base address 0x4002 4000) 207
Table 186: D/A Converter Register (CR - address 0x4002 4000) bit description 208
Table 187. Sectors in an EM783 device 214
Table 188. Code Read Protection options. 215
Table 189. Code Read Protection hardware/software interaction 215
Table 190. ISP commands allowed by CRP levels 216
Table 191. ISP command summary 217
Table 192. ISP Unlock command 217
Table 193. ISP Set Baud Rate command 218
Table 194. ISP Echo command 218
Table 195. ISP Write to RAM command 219
Table 196. ISP Read Memory command 219
Table 197. ISP Prepare sector(s) for write operation command 220
Table 198. ISP Copy command 220
Table 199. ISP Go command 221
Table 200. ISP Erase sector command 221
Table 201. ISP Blank check sector command 222
Table 202. ISP Read Part Identification command 222
Table 203. EM783 part identification numbers 222
Table 204. ISP Read Boot Code version command 222
Table 205. ISP Compare command 223
Table 206. ReadUID command 223
Table 207. ISP Return Codes Summary 223
Table 208. IAP Command Summary 225
Table 209. IAP Prepare sector(s) for write operation command 226
Table 210. IAP Copy RAM to flash command 227
Table 211. IAP Erase Sector(s) command 227
Table 212. IAP Blank check sector(s) command 228
Table 213. IAP Read Part Identification command 228
Table 214. IAP Read Boot Code version command 228
Table 215. IAP Compare command 229
Table 216. Reinvoke ISP 229
Table 217. IAP Read UID command 229
Table 218. IAP Write EEPROM command 230
Table 219. IAP Read EEPROM command 230
Table 220. IAP Status Codes Summary 230
Table 221. Memory mapping in debug mode 231
Table 222. Register overview: FMC (base address 0x4003 C000) 231
Table 223. Flash configuration register (FLASHCFG, address 0x4003 C010) bit description 232
Table 224. Flash module signature start register (FMSSTART - 0x4003 C020) bit description . 233
Table 225. Flash module signature stop register (FMSSTOP - 0x4003 C024) bit description 233
Table 226. FMSW0 register (FMSW0, address: 0x4003 C02C) bit description 233
Table 227. FMSW1 register (FMSW1, address: 0x4003 C030) bit description 234
Table 228. FMSW2 register (FMSW2, address: 0x4003 C034) bit description 234
Table 229. FMSW3 register (FMSW3, address: 0x4003 40C8) bit description 234
Table 230. EEPROM BIST start address register (EEMSSTART - address 0x4003 C09C) bit description 234
Table 231. EEPROM BIST stop address register (EEMSSTOP - address 0x4003 C0A0) bit description 235
Table 232. EEPROM BIST signature register (EEMSSIG - address 0x4003 C0A4) bit description 235
Table 233. Flash module status register (FMSTAT - 0x4003 CFE0) bit description 236
Table 234. Flash module status clear register (FMSTATCLR - 0x0x4003 CFE8) bit description 236
Table 235. ISR handler 242
Table 236. I2C Master Transmit Polling 242
Table 237. I2C Master Receive Polling 242
Table 238. I2C Master Transmit and Receive Polling . . 242
Table 239. I2C Master Transmit Interrupt 243
Table 240. I2C Master Receive Interrupt 243
Table 241. I2C Master Transmit Receive Interrupt 244
Table 242. I2C Slave Receive Polling 244
Table 243. I2C Slave Transmit Polling 244
Table 244. I2C Slave Receive Interrupt 244
Table 245. I2C Slave Transmit Interrupt 245
Table 246. I2C Set Slave Address 245
Table 247. I2C Get Memory Size 245
Table 248. I2C Setup 245
Table 249. I2C Set Bit Rate 246
Table 250. I2C Get Firmware Version 246
Table 251. I2C Get Status 246
Table 252. Error codes 246
Table 253. I2C Status code 247
Table 254. set_pll routine 257
Table 255. set_power routine 261
Table 256. Serial Wire Debug pin description. 264
Table 257. Summary of processor mode and stack use

options 268
Table 258. Core register set summary. 269
Table 259. PSR register combinations 270
Table 260. APSR bit assignments 271
Table 261. IPSR bit assignments. 271
Table 262. EPSR bit assignments 272
Table 263. PRIMASK register bit assignments 272
Table 264. CONTROL register bit assignments 273
Table 265. Memory access behavior 276
Table 266. Properties of different exception types 278
Table 267. Exception return behavior 283
Table 268. Cortex-M0 instructions 286
Table 269. CMSIS intrinsic functions to generate some Cortex-M0 instructions 288
Table 270. Intrinsic functions for accessing special registers via MRS/MSR 288
Table 271. Condition code suffixes 293
Table 272. Access instructions 293
Table 273. Data processing instructions 299
Table 274. ADC, ADD, RSB, SBC and SUB operand restrictions 301
Table 275. Branch and control instructions 308
Table 276. Branch ranges 309
Table 277. Miscellaneous instructions. 310
Table 278. Core peripheral register regions 317
Table 279. NVIC register summary 318
Table 280. CMSIS access to NVIC functions 318
Table 281. ISER bit assignments. 318
Table 282. ICER bit assignments 319
Table 283. ISPR bit assignments. 319
Table 284. ICPR bit assignments 320
Table 285. IPR bit assignments 320
Table 286. CMSIS functions for NVIC control 322
Table 287. Summary of the SCB registers 322
Table 288. CPUID register bit assignments. 323
Table 289. ICSR bit assignments 324
Table 290. AIRCR bit assignments 325
Table 291. SCR bit assignments 326
Table 292. CCR bit assignments 327
Table 293. System fault handler priority fields. 327
Table 294. SHPR2 register bit assignments 327
Table 295. SHPR3 register bit assignments 328
Table 296. System timer registers summary 328
Table 297. SYST_CSR bit assignments 328
Table 298. SYST_RVR bit assignments 329
Table 299. SYST_CVR bit assignments 329
Table 300. SYST_CALIB register bit assignments 330
Table 301. Cortex M0- instruction summary 330
Table 302. Abbreviations 334

24.4 Figures

Fig 1. EM783 block diagram 6
Fig 2. EM783 memory map 9
Fig 3. EM783 CGU block diagram 11
Fig 4. UVLO reset behaviour 27
Fig 5. Typical start-up time with respect to the internal reset 28
Fig 6. Typical start-up time with respect to the external reset 29
Fig 7. System PLL block diagram 31
Fig 8. Standard I/O pin configuration 51
Fig 9. Reset pad configuration 53
Fig 10. Pin configuration HVQFN 33 package 58
Fig 11. Auto-RTS Functional Timing 75
Fig 12. Auto-CTS Functional Timing 76
Fig 13. Auto-baud a) mode 0 and b) mode 1 waveform . 83
Fig 14. Algorithm for setting USART dividers 86
Fig 15. USART block diagram 96
Fig 16. I^2C-bus configuration 98
Fig 17. I^2C serial interface block diagram 108
Fig 18. Arbitration procedure 110
Fig 19. Serial clock synchronization. 110
Fig 20. Format in the Master Transmitter mode. 112
Fig 21. Format of Master Receiver mode 113
Fig 22. A Master Receiver switches to Master Transmitter after sending Repeated START 113
Fig 23. Format of Slave Receiver mode 114
Fig 24. Format of Slave Transmitter mode 114
Fig 25. Format and states in Master Transmitter mode . 118
Fig 26. Format and states in Master Receiver mode . . . 120
Fig 27. Format and states in Slave Receiver mode 124
Fig 28. Format and states in Slave Transmitter mode . . 126
Fig 29. Simultaneous Repeated START conditions from two masters 128
Fig 30. Forced access to a busy I^2C-bus. 129
Fig 31. Recovering from a bus obstruction caused by a LOW level on SDA. 129
Fig 32. Texas Instruments Synchronous Serial Frame Format: (a) Single and (b) Continuous/back-to-back, Two Frames Transfer. 145
Fig 33. SPI frame format with CPOL=0 and CPHA=0; (a) Single and (b) Continuous transfer 146
Fig 34. SPI frame format with CPOL=0 and CPHA=1 . . 147
Fig 35. SPI frame format with CPO=1 and CPHA=0; (a) Single and (b) Continuous transfer 148
Fig 36. SPI Frame Format with CPOL=1 and CPHA=1 . 149
Fig 37. Microwire frame format (single transfer) 150
Fig 38. Microwire frame format (continuous transfers) . 150
Fig 39. Microwire frame format setup and hold details . 152
Fig 40. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register. 165
Fig 41. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled 165
Fig 42. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled 166
Fig 43. 16-bit counter/timer block diagram. 167
Fig 44. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register. 180
Fig 45. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled 180
Fig 46. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled 181
Fig 47. 32-bit counter/timer block diagram 182
Fig 48. System tick timer block diagram 184
Fig 49. Watchdog block diagram. 193
Fig 50. Early Watchdog Feed with Windowed Mode Enabled 194
Fig 51. Correct Watchdog Feed with Windowed Mode Enabled 194
Fig 52. Watchdog Warning Interrupt 194
Fig 53. EM783-SC with external measurement circuitry connected to Metrology Engine. 195
Fig 54. EM783-SC calibration measurements setup. . . 202
Fig 55. DAC block diagram 209
Fig 56. Boot process flowchart 213
Fig 57. IAP parameter passing 226
Fig 58. Algorithm for generating a 128-bit signature . . . 237
Fig 59. Integer division routines pointer structure 239
Fig 60. I2C-bus driver routines pointer structure. 241
Fig 61. I2C slave mode set-up address packing 251
Fig 62. Power profiles pointer structure 255
Fig 63. Clock configuration for power API use 256
Fig 64. Power profiles usage. 261
Fig 65. Connecting the SWD pins to a standard SWD connector 265
Fig 66. Cortex-M0 implementation 266
Fig 67. Processor core register set 269
Fig 68. APSR, IPSR, EPSR register bit assignments . 270
Fig 69. Memory ordering restrictions. 275
Fig 70. Little endian format 277
Fig 71. Vector table 280
Fig 72. Exception entry stack contents 282
Fig 73. ASR #3 289
Fig 74. LSR #3 290
Fig 75. LSL #3. 290
Fig 76. ROR #3 291
Fig 77. IPR register 320

24.5 Contents

Chapter 1: Introductory information

1.1 Introduction . . . 3
1.2 Features and benefits . . . 3
1.3 Applications . . . 5
1.4 Ordering information . . . 5
1.5 Block diagram . . . 6
1.6 ARM Cortex-M0 processor . . . 7

Chapter 2: EM783 Memory map

2.1 Memory map . . . 8

Chapter 3: EM783 System configuration

3.1 Introduction . . . 10
3.2 Pin description . . . 10
3.3 Clocking and Power Control . . . 10
3.3.1 Clocking . . . 10
3.3.2 Power Control . . . 11
3.4 Register description . . . 11
3.4.1 System memory remap register . . . 13
3.4.2 Peripheral reset control register . . . 13
3.4.3 System PLL control register . . . 14
3.4.4 System PLL status register . . . 14
3.4.5 System oscillator control register . . . 15
3.4.6 Watchdog oscillator control register . . . 15
3.4.7 High-frequency oscillator control register . . . 16
3.4.8 LF oscillator control register . . . 16
3.4.9 System reset status register . . . 17
3.4.10 System PLL clock source select register . . . 18
3.4.11 System PLL clock source update register . . . 18
3.4.12 Main clock source select register . . . 19
3.4.13 Main clock source update enable register . . . 19
3.4.14 System clock divider register . . . 19
3.4.15 System clock control register . . . 20
3.4.16 SSP0 clock divider register . . . 21
3.4.17 UART clock divider register . . . 22
3.4.18 CLKOUT clock source select register . . . 22
3.4.19 CLKOUT clock source update enable register 22
3.4.20 CLKOUT clock divider register . . . 23
3.4.21 POR captured PIO status register 0 . . . 23
3.4.22 BrownOut Detect register . . . 23
3.4.23 System tick counter calibration register . . . 24
3.4.24 NMI source selection register . . . 24
3.4.25 Pin interrupt select registers . . . 24
3.4.26 Power configuration register . . . 24
3.4.27 Device ID register . . . 26
3.4.28 Device Configuration register . . . 26
3.5 Reset . . . 26
3.5.1 Under-Voltage Lockout (UVLO) reset behavior 26
3.6 Brownout detection . . . 27
3.7 Start-up timing . . . 27
3.8 Power management . . . 29
3.8.1 Run mode . . . 30
3.8.2 Sleep mode . . . 30
3.9 System PLL functional description . . . 31
3.9.1 Lock detector . . . 31
3.9.2 Direct output mode . . . 31
3.9.3 Power-down control . . . 32
3.9.4 Operating modes . . . 32
3.9.5 Divider ratio programming . . . 32
Post divider . . . 32
Feedback divider . . . 32
Changing the divider values . . . 32
3.9.6 Frequency selection . . . 32
3.9.6.1 Mode 1 (Normal mode) . . . 33
3.9.6.2 Mode 3 (Power-down mode) . . . 33
3.10 Flash memory access . . . 33

Chapter 4: EM783 Interrupt controller

4.1 Introduction . . . 34
4.2 Features . . . 34
4.3 Interrupt sources . . . 34

Chapter 5: EM783 General Purpose I/O (GPIO)

5.1 Basic configuration . . . 36
5.2 Features . . . 36
5.2.1 GPIO pin interrupt features . . . 36
5.2.2 GPIO group interrupt features . . . 36
5.2.3 GPIO port features . . . 36
5.3 Introduction . . . 36
5.3.1 GPIO pin interrupts . . . 37
5.3.2 GPIO group interrupt . . . 37
5.3.3 GPIO port . . . 37
5.4 Register description . . . 37
5.4.1 GPIO pin interrupts register description . . . 39
5.4.1.1 Pin interrupt mode register . . . 39
5.4.1.2 Pin interrupt level (rising edge) interrupt enable register . . . 39

5.4.1.3 Pin interrupt level (rising edge) interrupt set register . . . 39
5.4.1.4 Pin interrupt level (rising edge interrupt) clear register . . . 40
5.4.1.5 Pin interrupt active level (falling edge) interrupt enable register . . . 40
5.4.1.6 Pin interrupt active level (falling edge) interrupt set register . . . 41
5.4.1.7 Pin interrupt active level (falling edge interrupt) clear register . . . 41
5.4.1.8 Pin interrupt rising edge register . . . 42
5.4.1.9 Pin interrupt falling edge register . . . 42
5.4.1.10 Pin interrupt status register . . . 43
5.4.2 GPIO GROUP0/GROUP1 interrupt register description . . . 43
5.4.2.1 Grouped interrupt control register . . . 43
5.4.2.2 GPIO grouped interrupt port polarity registers 43
5.4.2.3 GPIO grouped interrupt port enable registers. 44
5.4.3 GPIO port register description . . . 44
5.4.3.1 GPIO port byte pin registers . . . 44
5.4.3.2 GPIO port word pin registers . . . 44
5.4.3.3 GPIO port direction registers . . . 45
5.4.3.4 GPIO port mask registers . . . 45
5.4.3.5 GPIO port pin registers . . . 45
5.4.3.6 GPIO masked port pin registers . . . 46
5.4.3.7 GPIO port set registers . . . 46
5.4.3.8 GPIO port clear registers . . . 46
5.4.3.9 GPIO port toggle registers . . . 46
5.5 Functional description . . . 47
5.5.1 Reading pin state . . . 47
5.5.2 GPIO output . . . 47
5.5.3 Masked I/O . . . 48
5.5.4 GPIO Interrupts . . . 48
5.5.4.1 Pin interrupts . . . 48
5.5.4.2 Group interrupts . . . 49
5.5.5 Recommended practices . . . 49

Chapter 6: EM783 I/O configuration

6.1 Introduction . . . 50
6.2 General description . . . 50
6.2.1 Pin function . . . 51
6.2.2 Pin mode . . . 51
6.2.3 Hysteresis . . . 52
6.2.4 Input inversion . . . 52
6.2.5 Analog/digital mode . . . 52
6.2.6 I^2C mode . . . 52
6.2.7 Output slew rate . . . 52
6.2.8 Open-Drain Mode . . . 52
6.2.9 RESET (pin RESET/P0_0) . . . 53
6.3 Register description . . . 53
6.3.1 I/O configuration registers . . . 54
6.3.1.1 Type D IOCON registers . . . 54
6.3.1.2 Type I IOCON registers . . . 56
6.3.1.3 Type A IOCON registers . . . 56

Chapter 7: EM783 Pin configuration

7.1 Pin configuration . . . 58
7.2 Pin description . . . 58

Chapter 8: EM783 USART

8.1 How to read this chapter . . . 67
8.2 Basic configuration . . . 67
8.3 Features . . . 67
8.4 Pin description . . . 67
8.5 Register description . . . 67
8.5.1 USART Receiver Buffer Register (DLAB = 0, Read Only) . . . 69
8.5.2 USART Transmitter Holding Register (DLAB = 0, Write Only) . . . 69
8.5.3 USART Divisor Latch LSB and MSB Registers (DLAB = 1) . . . 69
8.5.4 USART Interrupt Enable Register (DLAB = 0) 70
8.5.5 USART Interrupt Identification Register . . . 71
8.5.6 USART FIFO Control Register . . . 73
8.5.7 USART Modem Control Register . . . 74
8.5.7.1 Auto-flow control . . . 75
8.5.7.1.1 Auto-RTS . . . 75
8.5.7.1.2 Auto-CTS . . . 75
8.5.8 USART Line Control Register . . . 76
8.5.9 USART Line Status Register (LSR - 0x4000 8014, Read Only) . . . 77
8.5.10 USART Modem Status Register . . . 79
8.5.11 USART Scratch Pad Register (SCR - 0x4000 801C) . . . 80
8.5.12 USART Auto-baud Control Register (ACR - 0x4000 8020) . . . 80
8.5.13 Auto-baud . . . 81
8.5.14 Auto-baud modes . . . 82
8.5.15 IrDA Control Register (ICR - 0x4000 8024) . . 83
8.5.16 USART Fractional Divider Register (FDR - 0x4000 8028) . . . 84
8.5.16.1 Baud rate calculation . . . 85
8.5.16.1.1 Example 1: UART_PCLK = 14.7456 MHz, BR = 9600 . . . 87
8.5.16.1.2 Example 2: UART_PCLK = 12.0 MHz, BR = 115200 . . . 87
8.5.17 USART Oversampling Register (OSR - 0x4000 802C) . . . 87
8.5.18 USART Transmit Enable Register . . . 88
8.5.19 Smart Card Interface Control register (SCICTRL - 0x4000 8048) . . . 89
8.5.19.1 Smart Card Connection . . . 89
8.5.19.2 Smart Card Setup . . . 89

8.5.20 USART RS485 Control register 90
8.5.21 USART RS-485 Address Match register 91
8.5.22 USART RS-485 Delay value register 91
8.5.23 RS-485/EIA-485 modes of operation 91
RS-485/EIA-485 Normal Multidrop Mode 92
RS-485/EIA-485 Auto Address Detection (AAD) mode 92
RS-485/EIA-485 Auto Direction Control. 92
RS485/EIA-485 driver delay time. 93
RS485/EIA-485 output inversion 93
8.5.24 USART Synchronous mode control register . 93
8.6 Architecture 95

Chapter 9: EM783 I2C-bus interface

9.1 How to read this chapter 97
9.2 Basic configuration 97
9.3 Features 97
9.4 Applications 97
9.5 General description 98
9.5.1 I²C Fast-mode Plus 98
9.6 Pin description 98
9.6.1 External Pull-ups. 99
9.7 Register description 99
9.7.1 I²C Control Set register (CONSET) 100
9.7.2 I²C Status register (STAT). 102
9.7.3 I²C Data register (DAT). 102
9.7.4 I²C Slave Address register 0 (ADR0) 102
9.7.5 I²C SCL HIGH and LOW duty cycle registers (SCLH - 0x4000 0010 and SCLL- 0x4000 0014) 103
9.7.5.1 Selecting the appropriate I²C data rate and duty cycle. 103
9.7.6 I²C Control Clear register (CONCLR). 103
9.7.7 I²C Monitor mode control register (MMCTRL) 104
9.7.7.1 Interrupt in Monitor mode 105
9.7.7.2 Loss of arbitration in Monitor mode 105
9.7.8 I²C Slave Address registers (ADR[1, 2, 3]). . 106
9.7.9 I²C Data buffer register (DATA_BUFFER) . . 106
9.7.10 I²C Mask registers (MASK[0, 1, 2, 3]). 106
9.8 Functional description 107
9.8.1 Input filters and output stages. 108
9.8.2 Address Registers, ADR0 to ADR3 109
9.8.3 Address masks, MASK0 to MASK3 109
9.8.4 Comparator. 109
9.8.5 Shift register, DAT. 109
9.8.6 Arbitration and synchronization logic 109
9.8.7 Serial clock generator. 110
9.8.8 Timing and control 111
9.8.9 Control register, CONSET and CONCLR . . . 111
9.8.10 Status decoder and status register 111
9.9 I²C operating modes 111
9.9.1 Master Transmitter mode 112
9.9.2 Master Receiver mode 113
9.9.3 Slave Receiver mode 113
9.9.4 Slave Transmitter mode 114
9.10 Details of I²C operating modes. 115
9.10.1 Master Transmitter mode 115
9.10.2 Master Receiver mode 118
9.10.3 Slave Receiver mode 121
9.10.4 Slave Transmitter mode 124
9.10.5 Miscellaneous states 127
9.10.5.1 STAT = 0xF8 127
9.10.5.2 STAT = 0x00 127
9.10.6 Some special cases. 127
9.10.6.1 Simultaneous Repeated START conditions from two masters 128
9.10.6.2 Data transfer after loss of arbitration 128
9.10.6.3 Forced access to the I²C-bus. 128
9.10.6.4 I²C-bus obstructed by a Low on SCL or SDA 129
9.10.6.5 Bus error 130
9.10.7 I²C state service routines. 130
9.10.8 Initialization 130
9.10.9 I²C interrupt service 130
9.10.10 The state service routines 130
9.10.11 Adapting state services to an application. . . 130
9.11 Software example 131
9.11.1 Initialization routine 131
9.11.2 Start Master Transmit function 131
9.11.3 Start Master Receive function 131
9.11.4 I²C interrupt routine 131
9.11.5 Non mode-specific states. 132
9.11.5.1 State: 0x00. 132
9.11.5.2 Master States. 132
9.11.5.3 State: 0x08. 132
9.11.5.4 State: 0x10. 132
9.11.6 Master Transmitter states 132
9.11.6.1 State: 0x18. 132
9.11.6.2 State: 0x20. 133
9.11.6.3 State: 0x28. 133
9.11.6.4 State: 0x30. 133
9.11.6.5 State: 0x38. 133
9.11.7 Master Receive states 134
9.11.7.1 State: 0x40. 134
9.11.7.2 State: 0x48. 134
9.11.7.3 State: 0x50. 134
9.11.7.4 State: 0x58. 134
9.11.8 Slave Receiver states 134
9.11.8.1 State: 0x60. 134
9.11.8.2 State: 0x68. 135
9.11.8.3 State: 0x70. 135
9.11.8.4 State: 0x78. 135
9.11.8.5 State: 0x80. 135
9.11.8.6 State: 0x88. 136
9.11.8.7 State: 0x90. 136
9.11.8.8 State: 0x98. 136
9.11.8.9 State: 0xA0. 136
9.11.9 Slave Transmitter states 136
9.11.9.1 State: 0xA8. 136
9.11.9.2 State: 0xB0. 137
9.11.9.3 State: 0xB8. 137

9.11.9.4 State: 0xC0 137
9.11.9.5 State: 0xC8 137

Chapter 10: EM783 SPI/SSP

10.1 Features 138
10.2 General description 138
10.3 Pin description 139
10.4 Clocking and power control 139
10.5 Register description 139
10.5.1 SPI/SSP Control Register 0 140
10.5.2 SPI/SSP0 Control Register 1 141
10.5.3 SPI/SSP Data Register 142
10.5.4 SPI/SSP Status Register 142
10.5.5 SPI/SSP Clock Prescale Register 142
10.5.6 SPI/SSP Interrupt Mask Set/Clear Register 143
10.5.7 SPI/SSP Raw Interrupt Status Register 143
10.5.8 SPI/SSP Masked Interrupt Status Register . 144
10.5.9 SPI/SSP Interrupt Clear Register 144
10.6 Functional description 145
10.6.1 Texas Instruments synchronous serial frame format 145
10.6.2 SPI frame format 145
10.6.2.1 Clock Polarity (CPOL) and Clock Phase (CPHA) control 146
10.6.2.2 SPI format with CPOL=0, CPHA=0 146
10.6.2.3 SPI format with CPOL=0, CPHA=1 147
10.6.2.4 SPI format with CPOL=1, CPHA=0 148
10.6.2.5 SPI format with CPOL=1, CPHA=1 149
10.6.3 National Semiconductor Microwire Format . 150
10.6.3.1 Setup and hold time requirements on CS with respect to SK in Microwire mode 151

Chapter 11: EM783 16-bit counter/timer (CT16B0)

11.1 Basic configuration 153
11.2 Features 153
11.3 Applications 153
11.4 Description 154
11.5 Pin description 154
11.6 Register description 154
11.6.1 Interrupt Registers (IR) 155
11.6.2 Timer Control Register (TCR) 156
11.6.3 Timer Counter registers (TC) 156
11.6.4 Prescale Register (PR) 156
11.6.5 Prescale Counter register (PC) 157
11.6.6 Match Control Register (MCR) 157
11.6.7 Match Registers (MR0/1/2/3) 158
11.6.8 Capture Control Register (CCR) 158
11.6.9 Capture Registers (CR0/1/2/3) 160
11.6.10 External Match Register (EMR) 160
11.6.11 Count Control Register (CTCR) 162
11.6.12 PWM Control register (PWMC) 163
11.6.13 Rules for single edge controlled PWMs 164
11.7 Example timer operation 165
11.8 Architecture 166

Chapter 12: EM783 32 bit counter/timer (CT32B0)

12.1 Basic configuration 168
12.2 Features 168
12.3 Applications 168
12.4 Description 169
12.5 Pin description 169
12.6 Register description 169
12.6.1 Interrupt Registers (IR) 170
12.6.2 Timer Control Register (TCR) 171
12.6.3 Timer Counter (TC) 171
12.6.4 Prescale Register (PR) 171
12.6.5 Prescale Counter Register (PC) 172
12.6.6 Match Control Register (MCR) 172
12.6.7 Match Registers (MR0/1/2/3) 173
12.6.8 Capture Control Register (CCR) 173
12.6.9 Capture Registers (CR) 175
12.6.10 External Match Register (EMR) 175
12.6.11 Count Control Register (CTCR) 177
12.6.12 PWM Control Register (PWMC) 178
12.6.13 Rules for single edge controlled PWMs 179
12.7 Example timer operation 180
12.8 Architecture 181

Chapter 13: EM783 System tick timer

13.1 How to read this chapter 183
13.2 Features 183
13.3 Description 183
13.4 Operation 183
13.5 Register description 184
13.5.1 System Timer Control and status register (STCTRL - 0xE000 E010) 184
13.5.2 System Timer Reload value register (STRELOAD - 0xE000 E014) 185
13.5.3 System Timer Current value register (STCURR - 0xE000 E018) 185
13.5.4 System Timer Calibration value register (STCALIB - 0xE000 E01C) 185

Chapter 14: EM783 Windowed WatchDog Timer (WWDT)

14.1 How to read this chapter 187
14.2 Features 187
14.3 Applications 187
14.4 Description 187
14.5 Clocking and power control 188
14.6 Register description 188
14.6.1 Watchdog Mode register (WDMOD) 189
14.6.2 Watchdog Timer Constant register (WDTC) . 190
14.6.3 Watchdog Feed register (WDFEED) 191
14.6.4 Watchdog Timer Value register (WDTV) 191
14.6.5 Watchdog Clock Select register (WDCLKSEL) 192
14.6.6 Watchdog Timer Warning Interrupt register (WDWARNINT) 192
14.6.7 Watchdog Timer Window register (WDWINDOW) 192
14.7 Block diagram 193
14.8 Watchdog timing examples 193

Chapter 15: EM783 Metrology engine

15.1 Introduction 195
15.2 Features 195
15.3 Pin description 196
15.4 External analog front-end design guidelines 196
15.4.1 High-gain current inputs (I1_H, I2_H and I3_H) 197
15.4.2 Low-gain current inputs (I1_L, I2_L, I3_L, I1, I2, I3, I4, I5 and I6) 197
15.4.3 Voltage inputs (V1, V2 and V3) 197
15.5 Output definitions 197
15.5.1 Output for sinusoidal and non-sinusoidal voltage and sinusoidal and non-sinusoidal current . . 197
15.5.2 Additional output for sinusoidal voltage and sinusoidal and non-sinusoidal current 199
15.5.3 Additional output for temperature compensation and power integration 200
15.6 Operation 200
15.6.1 Configuration data 200
15.6.1.1 Voltage measurement input range V1pp . . . 200
15.6.1.2 Current measurement input range I1_Hpp . 201
15.6.1.3 Current measurement input range I1_Lpp . . 201
15.6.1.4 Phase error correction angle DeltaPhi1H . . 201
15.6.1.5 Phase error correction angle DeltaPhi1L . . . 201
15.6.2 Calibration procedure 201
15.6.2.1 V1pp calibration 202
15.6.2.2 I1_Hpp calibration 202
15.6.2.3 I1_Lpp calibration 203
15.6.2.4 DeltaPhi1H calibration 203
15.6.2.5 DeltaPhi1L calibration 204

Chapter 16: EM783 Metrology engine interface

16.1 Metrology driver interface 205
16.1.1 Metrology engine initialization 205
16.1.2 Metrology engine interrupt handling 205
16.1.3 Metrology engine calibration 205
16.1.4 Metrology engine start and stop 205
16.1.5 Metrology measurement data 205
16.2 Metrology Engine interface 206

Chapter 17: EM783 Digital-to-Analog Converter (DAC)

17.1 How to read this chapter 207
17.2 Basic configuration 207
17.3 Features 207
17.4 Pin description 207
17.5 Register description 207
17.5.1 D/A Control Register (CR) 208
17.5.2 Power-Down Control 209
17.6 Operation 209
17.6.1 Hardware-triggered conversion 209
17.6.2 Interrupts 209
17.7 Block diagram 209

Chapter 18: EM783 Flash/EEPROM programming firmware

18.1 Bootloader 210
18.2 Features 210
18.3 Applications 210
18.4 Description 210
18.4.1 Memory map after any reset 211
18.4.1.1 Criterion for Valid User Code 211
18.4.2 Communication protocol 211
18.4.2.1 Connections 211
18.4.2.2 ISP command format 212
18.4.2.3 ISP response format 212
18.4.2.4 ISP data format 212
18.4.2.5 ISP flow control 212
18.4.2.6 ISP command abort 212
18.4.2.7 Interrupts during ISP 212
18.4.2.8 Interrupts during IAP 212
18.4.2.9 RAM used by ISP command handler 212
18.4.2.10 RAM used by IAP command handler 213
18.4.3 Boot process flowchart 213
18.4.4 Sector numbers 214

18.5 Code Read Protection (CRP) 214
18.5.1 ISP entry protection 216
18.6 ISP commands 217
18.6.1 Unlock <Unlock code> 217
18.6.2 Set Baud Rate <Baud Rate> <stop bit> 218
18.6.3 Echo <setting> 218
18.6.4 Write to RAM <start address> <# bytes> . . . 218
18.6.5 Read Memory <address> <no. of bytes> . . . 219
18.6.6 Prepare sector(s) for write operation <start sector number> <end sector number> 219
18.6.7 Copy RAM to flash <Flash address> <RAM address> <no of bytes> 220
18.6.8 Go <address> <mode> 221
18.6.9 Erase sector(s) <start sector number> <end sector number> 221
18.6.10 Blank check sector(s) <sector number> <end sector number> 222
18.6.11 Read Part Identification number 222
18.6.12 Read Boot code version number 222
18.6.13 Compare <address1> <address2> <# bytes> 223
18.6.14 ReadUID 223
18.6.15 ISP Return Codes 223
18.7 IAP commands 224
18.7.1 Prepare sector(s) for write operation 226
18.7.2 Copy RAM to flash 227
18.7.3 Erase Sector(s) 227
18.7.4 Blank check sector(s) 228
18.7.5 Read Part Identification number 228
18.7.6 Read Boot code version number 228
18.7.7 Compare <address1> <address2> <# bytes> 229
18.7.8 Reinvoke ISP 229
18.7.9 Read UID 229
18.7.10 Write EEPROM 230
18.7.11 Read EEPROM 230
18.7.12 IAP Status Codes 230
18.8 Debug notes 231
18.8.1 Comparing flash images 231
18.8.2 Serial Wire Debug (SWD) flash programming interface 231
18.9 Flash controller registers 231
18.9.1 Flash memory access register 232
18.9.2 Flash signature generation 232
18.9.3 Signature generation address and control registers 232
18.9.4 Signature generation result registers 233
18.9.5 EEPROM BIST start address register 234
18.9.6 EEPROM BIST stop address register 234
18.9.7 EEPROM signature register 235
18.9.8 Flash module status register 235
18.9.9 Flash module status clear register 236
18.9.10 Algorithm and procedure for signature generation 236
Signature generation 236
Content verification 236

Chapter 19: EM783 Integer division routines

19.1 How to read this chapter 238
19.2 Features 238
19.3 General description 238
19.4 API description 239
19.4.1 Examples 239
19.4.1.1 Initialization 239
19.4.1.2 Signed division 239
19.4.1.3 Unsigned division with remainder 240

Chapter 20: EM783 I2C-bus ROM driver routines

20.1 How to read this chapter 241
20.2 Features 241
20.3 General description 241
20.4 API description 242
20.4.1 I2C ROM driver variables 247
20.4.1.1 I2C Handle 247
20.4.2 PARAM and RESULT structure 247
20.4.3 Error structure 248
20.4.4 I2C Mode 249
20.4.5 I2C ROM driver pointer 249
20.4.6 API function table 249
20.4.7 I2C set-up 250
20.4.8 I2C Master mode set-up 251
20.4.9 I2C Slave mode set-up 251
20.4.10 I2C Master Transmit/Receive 252
20.4.11 The I2C Slave Mode Transmit/Receive 253

Chapter 21: EM783 Power profiles

21.1 How to read this chapter 255
21.2 Features 255
21.3 General description 255
21.4 Definitions 256
21.5 Clocking routine 256
21.5.1 set_pll 256
21.5.1.1 Param0: system PLL input frequency and Param1: expected system clock 257
21.5.1.2 Param2: mode 258
21.5.1.3 Param3: system PLL lock time-out 258
21.5.1.4 Code examples 258
21.5.1.4.1 Invalid frequency (device maximum clock rate exceeded) 258
21.5.1.4.2 Invalid frequency selection (system clock divider restrictions) 259
21.5.1.4.3 Exact solution cannot be found (PLL) 259
21.5.1.4.4 System clock less than or equal to the expected value 259

21.5.1.4.5 System clock greater than or equal to the expected value . . . 259
21.5.1.4.6 System clock approximately equal to the expected value . . . 260
21.6 Power routine . . . 260
21.6.1 set_power . . . 260
21.6.1.1 Param0: main clock . . . 262
21.6.1.2 Param1: mode . . . 262
21.6.1.3 Param2: system clock . . . 262
21.6.1.4 Code examples . . . 262
21.6.1.4.1 Invalid frequency (device maximum clock rate exceeded) . . . 262
21.6.1.4.2 An applicable power setup . . . 262

Chapter 22: EM783 Serial wire debug

22.1 Features . . . 264
22.2 General description . . . 264
22.3 Pin description . . . 264
22.4 Functional description . . . 264
22.4.1 Systick timer in debug mode . . . 264
22.4.2 Debug connections for SWD . . . 264

Chapter 23: Appendix: EM783 ARM Cortex-M0 Reference

23.1 Introduction . . . 266
23.2 About the Cortex-M0 processor and core peripherals . . . 266
23.2.1 System-level interface . . . 267
23.2.2 Integrated configurable debug . . . 267
23.2.3 Cortex-M0 processor features summary . . . 267
23.2.4 Cortex-M0 core peripherals . . . 267
23.3 Processor . . . 268
23.3.1 Programmers model . . . 268
23.3.1.1 Processor modes . . . 268
23.3.1.2 Stacks . . . 268
23.3.1.3 Core registers . . . 268
23.3.1.3.1 General-purpose registers . . . 269
23.3.1.3.2 Stack Pointer . . . 269
23.3.1.3.3 Link Register . . . 270
23.3.1.3.4 Program Counter . . . 270
23.3.1.3.5 Program Status Register . . . 270
23.3.1.3.6 Exception mask register . . . 272
23.3.1.3.7 CONTROL register . . . 273
23.3.1.4 Exceptions and interrupts . . . 273
23.3.1.5 Data types . . . 273
23.3.1.6 The Cortex Microcontroller Software Interface Standard . . . 274
23.3.2 Memory model . . . 274
23.3.2.1 Memory regions, types and attributes . . . 274
23.3.2.2 System ordering of memory accesses . . . 275
23.3.2.3 Behavior of memory accesses . . . 276
23.3.2.4 Software ordering of memory accesses . . . 276
23.3.2.5 Memory endianness . . . 277
23.3.2.5.1 Little endian format . . . 277
23.3.3 Exception model . . . 277
23.3.3.1 Exception states . . . 277
23.3.3.2 Exception types . . . 278
23.3.3.3 Exception handlers . . . 279
23.3.3.4 Vector table . . . 279
23.3.3.5 Exception priorities . . . 280
23.3.3.6 Exception entry and return . . . 281
23.3.3.6.1 Exception entry . . . 281
23.3.3.6.2 Exception return . . . 282
23.3.4 Fault handling . . . 283
23.3.4.1 Lockup . . . 283
23.3.5 Power management . . . 284
23.3.5.1 Entering sleep mode . . . 284
23.3.5.1.1 Wait for interrupt . . . 284
23.3.5.1.2 Wait for event . . . 284
23.3.5.1.3 Sleep-on-exit . . . 284
23.3.5.2 Wakeup from sleep mode . . . 285
23.3.5.2.1 Wakeup from WFI or sleep-on-exit . . . 285
23.3.5.2.2 Wakeup from WFE . . . 285
23.3.5.3 Power management programming hints . . . 285
23.4 Instruction set . . . 285
23.4.1 Instruction set summary . . . 285
23.4.2 Intrinsic functions . . . 287
23.4.3 About the instruction descriptions . . . 288
23.4.3.1 Operands . . . 288
23.4.3.2 Restrictions when using PC or SP . . . 289
23.4.3.3 Shift Operations . . . 289
23.4.3.3.1 ASR . . . 289
23.4.3.3.2 LSR . . . 290
23.4.3.3.3 LSL . . . 290
23.4.3.3.4 ROR . . . 291
23.4.3.4 Address alignment . . . 291
23.4.3.5 PC-relative expressions . . . 291
23.4.3.6 Conditional execution . . . 292
23.4.3.6.1 The condition flags . . . 292
23.4.3.6.2 Condition code suffixes . . . 293
23.4.4 Memory access instructions . . . 293
23.4.4.1 ADR . . . 293
23.4.4.1.1 Syntax . . . 293
23.4.4.1.2 Operation . . . 294
23.4.4.1.3 Restrictions . . . 294
23.4.4.1.4 Condition flags . . . 294
23.4.4.1.5 Examples . . . 294
23.4.4.2 LDR and STR, immediate offset . . . 294
23.4.4.2.1 Syntax . . . 294
23.4.4.2.2 Operation . . . 294
23.4.4.2.3 Restrictions . . . 295
23.4.4.2.4 Condition flags . . . 295
23.4.4.2.5 Examples . . . 295
23.4.4.3 LDR and STR, register offset . . . 295
23.4.4.3.1 Syntax . . . 295
23.4.4.3.2 Operation . . . 296
23.4.4.3.3 Restrictions . . . 296
23.4.4.3.4 Condition flags . . . 296

23.4.4.3.5 Examples 296
23.4.4.4 LDR, PC-relative 296
23.4.4.4.1 Syntax 296
23.4.4.4.2 Operation 296
23.4.4.4.3 Restrictions 296
23.4.4.4.4 Condition flags 296
23.4.4.4.5 Examples 297
23.4.4.5 LDM and STM 297
23.4.4.5.1 Syntax 297
23.4.4.5.2 Operation 297
23.4.4.5.3 Restrictions 297
23.4.4.5.4 Condition flags 298
23.4.4.5.5 Examples 298
23.4.4.5.6 Incorrect examples 298
23.4.4.6 PUSH and POP 298
23.4.4.6.1 Syntax 298
23.4.4.6.2 Operation 298
23.4.4.6.3 Restrictions 298
23.4.4.6.4 Condition flags 299
23.4.4.6.5 Examples 299
23.4.5 General data processing instructions 299
23.4.5.1 ADC, ADD, RSB, SBC, and SUB 300
23.4.5.1.1 Syntax 300
23.4.5.1.2 Operation 300
23.4.5.1.3 Restrictions 301
23.4.5.1.4 Examples 301
23.4.5.2 AND, ORR, EOR, and BIC 301
23.4.5.2.1 Syntax 302
23.4.5.2.2 Operation 302
23.4.5.2.3 Restrictions 302
23.4.5.2.4 Condition flags 302
23.4.5.2.5 Examples 302
23.4.5.3 ASR, LSL, LSR, and ROR 302
23.4.5.3.1 Syntax 302
23.4.5.3.2 Operation 303
23.4.5.3.3 Restrictions 303
23.4.5.3.4 Condition flags 303
23.4.5.3.5 Examples 303
23.4.5.4 CMP and CMN 303
23.4.5.4.1 Syntax 304
23.4.5.4.2 Operation 304
23.4.5.4.3 Restrictions 304
23.4.5.4.4 Condition flags 304
23.4.5.4.5 Examples 304
23.4.5.5 MOV and MVN 304
23.4.5.5.1 Syntax 304
23.4.5.5.2 Operation 305
23.4.5.5.3 Restrictions 305
23.4.5.5.4 Condition flags 305
23.4.5.5.5 Example 305
23.4.5.6 MULS 305
23.4.5.6.1 Syntax 305
23.4.5.6.2 Operation 306
23.4.5.6.3 Restrictions 306
23.4.5.6.4 Condition flags 306
23.4.5.6.5 Examples 306
23.4.5.7 REV, REV16, and REVSH 306
23.4.5.7.1 Syntax 306
23.4.5.7.2 Operation 306
23.4.5.7.3 Restrictions 307
23.4.5.7.4 Condition flags 307
23.4.5.7.5 Examples 307
23.4.5.8 SXT and UXT 307
23.4.5.8.1 Syntax 307
23.4.5.8.2 Operation 307
23.4.5.8.3 Restrictions 307
23.4.5.8.4 Condition flags 307
23.4.5.8.5 Examples 308
23.4.5.9 TST 308
23.4.5.9.1 Syntax 308
23.4.5.9.2 Operation 308
23.4.5.9.3 Restrictions 308
23.4.5.9.4 Condition flags 308
23.4.5.9.5 Examples 308
23.4.6 Branch and control instructions 308
23.4.6.1 B, BL, BX, and BLX 309
23.4.6.1.1 Syntax 309
23.4.6.1.2 Operation 309
23.4.6.1.3 Restrictions 309
23.4.6.1.4 Condition flags 310
23.4.6.1.5 Examples 310
23.4.7 Miscellaneous instructions 310
23.4.7.1 BKPT 310
23.4.7.1.1 Syntax 310
23.4.7.1.2 Operation 311
23.4.7.1.3 Restrictions 311
23.4.7.1.4 Condition flags 311
23.4.7.1.5 Examples 311
23.4.7.2 CPS 311
23.4.7.2.1 Syntax 311
23.4.7.2.2 Operation 311
23.4.7.2.3 Restrictions 311
23.4.7.2.4 Condition flags 311
23.4.7.2.5 Examples 311
23.4.7.3 DMB 311
23.4.7.3.1 Syntax 312
23.4.7.3.2 Operation 312
23.4.7.3.3 Restrictions 312
23.4.7.3.4 Condition flags 312
23.4.7.3.5 Examples 312
23.4.7.4 DSB 312
23.4.7.4.1 Syntax 312
23.4.7.4.2 Operation 312
23.4.7.4.3 Restrictions 312
23.4.7.4.4 Condition flags 312
23.4.7.4.5 Examples 312
23.4.7.5 ISB 312
23.4.7.5.1 Syntax 312
23.4.7.5.2 Operation 313
23.4.7.5.3 Restrictions 313
23.4.7.5.4 Condition flags 313
23.4.7.5.5 Examples 313
23.4.7.6 MRS 313
23.4.7.6.1 Syntax 313
23.4.7.6.2 Operation 313
23.4.7.6.3 Restrictions 313

23.4.7.6.4 Condition flags 313
23.4.7.6.5 Examples 313
23.4.7.7 MSR 313
23.4.7.7.1 Syntax. 313
23.4.7.7.2 Operation 314
23.4.7.7.3 Restrictions. 314
23.4.7.7.4 Condition flags 314
23.4.7.7.5 Examples 314
23.4.7.8 NOP 314
23.4.7.8.1 Syntax. 314
23.4.7.8.2 Operation 314
23.4.7.8.3 Restrictions. 314
23.4.7.8.4 Condition flags 314
23.4.7.8.5 Examples 314
23.4.7.9 SEV 314
23.4.7.9.1 Syntax. 315
23.4.7.9.2 Operation 315
23.4.7.9.3 Restrictions. 315
23.4.7.9.4 Condition flags 315
23.4.7.9.5 Examples 315
23.4.7.10 SVC 315
23.4.7.10.1 Syntax. 315
23.4.7.10.2 Operation 315
23.4.7.10.3 Restrictions. 315
23.4.7.10.4 Condition flags 315
23.4.7.10.5 Examples 315
23.4.7.11 WFE 315
23.4.7.11.1 Syntax. 316
23.4.7.11.2 Operation 316
23.4.7.11.3 Restrictions. 316
23.4.7.11.4 Condition flags 316
23.4.7.11.5 Examples 316
23.4.7.12 WFI 316
23.4.7.12.1 Syntax. 316
23.4.7.12.2 Operation 316
23.4.7.12.3 Restrictions. 317
23.4.7.12.4 Condition flags 317
23.4.7.12.5 Examples 317
23.5 Peripherals. 317
23.5.1 About the ARM Cortex-M0. 317
23.5.2 Nested Vectored Interrupt Controller 317
23.5.2.1 Accessing the Cortex-M0 NVIC registers using CMSIS 318
23.5.2.2 Interrupt Set-enable Register. 318
23.5.2.3 Interrupt Clear-enable Register 319
23.5.2.4 Interrupt Set-pending Register. 319
23.5.2.5 Interrupt Clear-pending Register 319
23.5.2.6 Interrupt Priority Registers 320
23.5.2.7 Level-sensitive and pulse interrupts. 321
23.5.2.7.1 Hardware and software control of interrupts 321
23.5.2.8 NVIC usage hints and tips 322
23.5.2.8.1 NVIC programming hints 322
23.5.3 System Control Block. 322
23.5.3.1 CMSIS mapping of Cortex-M0 SCB registers 323
23.5.3.2 CPUID Register 323
23.5.3.3 Interrupt Control and State Register 323
23.5.3.4 Application Interrupt/Reset Control Register 325
23.5.3.5 System Control Register 326
23.5.3.6 Configuration and Control Register 326
23.5.3.7 System Handler Priority Registers. 327
23.5.3.7.1 System Handler Priority Register 2 327
23.5.3.7.2 System Handler Priority Register 3 327
23.5.3.8 SCB usage hints and tips. 328
23.5.4 System timer, SysTick 328
23.5.4.1 SysTick Control and Status Register 328
23.5.4.2 SysTick Reload Value Register 329
23.5.4.2.1 Calculating the RELOAD value 329
23.5.4.3 SysTick Current Value Register 329
23.5.4.4 SysTick Calibration Value Register 329
23.5.4.5 SysTick usage hints and tips 330
23.6 Cortex-M0 instruction summary 330

Chapter 24: Supplementary information

24.1 Abbreviations. 334
24.2 Legal information. 335
24.2.1 Definitions. 335
24.2.2 Disclaimers 335
24.2.3 Trademarks. 335
24.3 Tables 336
24.4 Figures 341
24.5 Contents 342

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 14 November 2013
Document identifier: UM10586