UM10736

LPC15xx User manual

Rev. 1.1 — 3 March 2014 **User manual**

Document information

Info	Content
Keywords	LPC1500, LPC1500 User manual , LPC15xx UM, LPC15xx User manual, LPC1549, LPC1548, LPC1547, LPC1519, LPC1518, LPC1517
Abstract	LPC15xx User manual

Revision history

Rev	Date	Description
1.1	20140303	LPC15xx User manual
Modifications:		• Section 5.4.2 “Criterion for valid user code” corrected: If the signature is not valid, the part enumerates as USB MSC. Also see Figure 7. • Number of bits corrected in Table 228 “SCT event state mask registers 0 to 15 (EV[0:15]_STATE, addresses 0x1C01 8300 (EV0_STATE) to 0x1C01 8378 (EV15_STATE) (SCT0) and 0x1C01 C300 (EV0_STATE) to 0x1C01 C378 (EV15_STATE) (SCT1)) bit description”. • Figure 11 “Example: Connect function U0_RXD and U0_TXD to pins” corrected.
1	20140213	First LPC15xx User manual revision

Contact information

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

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

1.1 Introduction

The LPC15xx are ARM Cortex-M3 based microcontrollers for embedded applications featuring a rich peripheral set with very low power consumption. The ARM Cortex-M3 is a next generation core that offers system enhancements such as enhanced debug features and a higher level of support block integration.

The LPC15xx operate at CPU frequencies of up to 72 MHz. The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals. The ARM Cortex-M3 CPU also includes an internal prefetch unit that supports speculative branching.

The LPC15xx include up to 256 kB of flash memory, 32 kB of ROM, a 4 kB EEPROM, and up to 36 kB of SRAM. The peripheral compliment includes one full-speed USB 2.0 device, two SPI interfaces, three USARTs, one Fast-mode Plus I^2C-bus interface, one C_CAN module, PWM/timer subsystem with four configurable, multi-purpose State Configurable Timers (SCTimer/PWM) with input pre-processing unit, a Real-time clock module with independent power supply and a dedicated oscillator, two 12-channel/12-bit, 2 Msamples/sec ADCs, one 12-bit, 500 kSamples/sec DAC, four voltage comparators with internal voltage reference, and a temperature sensor. A DMA engine can service most peripherals.

For additional documentation, see Section 45.2 “References”.

1.2 Features

- System:
  - ARM Cortex-M3 processor, running at frequencies of up to 72 MHz.
  - ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
  - System tick timer.
  - Serial Wire Debug (SWD) with four breakpoints and two watchpoints.
  - Single-cycle multiplier supported.
  - Memory Protection Unit (MPU) included.
- Memory:
  - Up to 256 kB on-chip flash programming memory with 256 Byte page write and erase.
  - Up to 36 kB SRAM.
  - 4 kB EEPROM.

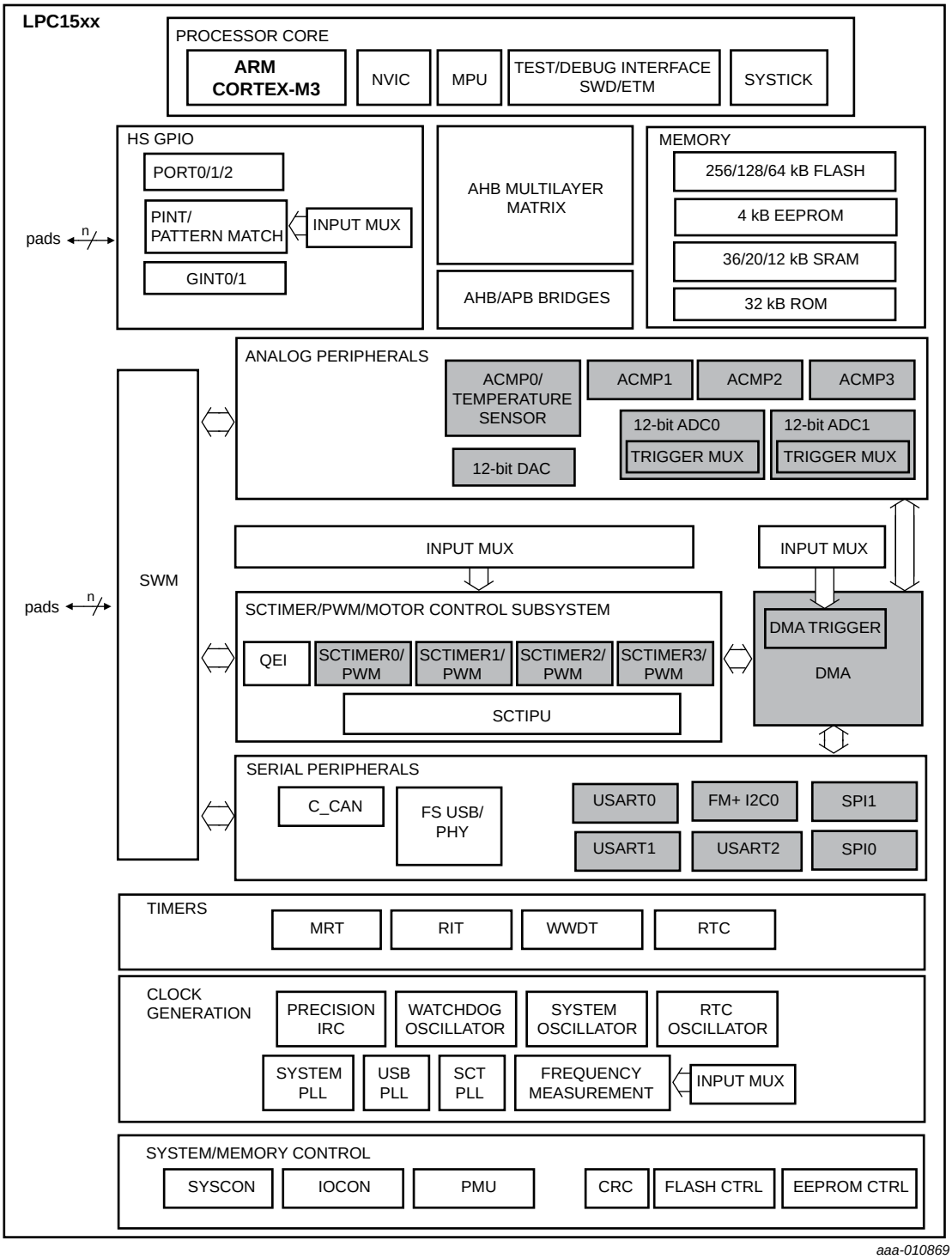
- ROM API support:
  - Boot loader with boot options from flash or external source via USART, C_CAN, or USB
  - USB drivers
  - ADC drivers
  - SPI drivers
  - USART drivers
  - I2C drivers
  - Power profiles and power mode configuration with low-power mode configuration option
  - DMA drivers
  - C_CAN drivers
  - Flash In-Application Programming (IAP) and In-System Programming (ISP).
- Digital peripherals:
  - Simple DMA engine with 18 channels and 20 programmable input triggers.
  - High-speed GPIO interface with up to 76 General-Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors, open-drain mode, input inverter, and programmable digital glitch filter.
  - GPIO interrupt generation capability with boolean pattern-matching feature on eight external inputs.
  - Two GPIO grouped port interrupts.
  - Switch matrix for flexible configuration of each I/O pin function.
  - CRC engine.
  - Quadrature Encoder Interface (QEI).
- Configurable PWM/timer/motor control subsystem:
  - Up to four 32-bit counter/timers or up to eight 16-bit counter/timers or combinations of 16-bit and 32-bit timers.
  - Up to 28 match outputs and 22 configurable capture inputs with input multiplexer.
  - Dither engine for improved average resolution of pulse edges.
  - Four State Configurable Timers (SCTimers) for highly flexible, event-driven timing and PWM applications.
  - SCT Input Pre-processor Unit (SCTIPU) for processing timer inputs and immediate handling of abort situations.
  - Integrated with ADC threshold compare interrupts, temperature sensor, and analog comparator outputs for motor control feedback using analog signals.
- Special-application and simple timers:
  - 24-bit, four-channel, multi-rate timer (MRT) for repetitive interrupt generation at up to four programmable, fixed rates.
  - Repetitive interrupt timer for general purpose use.
  - Windowed Watchdog timer (WWDT).

  - High-resolution 32-bit Real-time clock (RTC) with selectable 1 s or 1 ms time resolution running in the always-on power domain. RTC can be used for wake-up from all low power modes including Deep power-down.

- Analog peripherals:
  - Two 12-bit ADC with up to 12 input channels per ADC and with multiple internal and external trigger inputs and sample rates of up to 2 Msamples/s. Each ADC supports two independent conversion sequences. ADC conversion clock can be the system clock or an asynchronous clock derived from one of the three PLLs.
  - One 12-bit DAC.
  - Integrated temperature sensor and band gap internal reference voltage.
  - Four comparators with external and internal voltage references (ACMP0 to 3). Comparator outputs are internally connected to the SCTimer/PWMs and ADCs and externally to pins. Each comparator output contains a programmable glitch filter.
- Serial interfaces:
  - Three USART interfaces with DMA, RS-485 support, auto-baud, and with synchronous mode and 32 kHz mode for wake-up from Deep-sleep and Power-down modes. The USARTs share a fractional baud-rate generator.
  - Two SPI controllers.
  - One I^2C-bus interface supporting fast mode and Fast-mode Plus with data rates of up to 1Mbit/s and with multiple address recognition and monitor mode.
  - One C_CAN controller.
  - One USB 2.0 full-speed device controller with on-chip PHY.
- Clock generation:
  - 12 MHz internal RC oscillator trimmed to 1 % accuracy for $-25\ °C \leq T_{amb} \leq +85\ °C$ that can optionally be used as a system clock.
  - Crystal oscillator with an operating range of 1 MHz to 25 MHz.
  - Watchdog oscillator running at the fixed frequency of 503 kHz.
  - 32 kHz low-power RTC oscillator with 32 kHz, 1 kHz, and 1 Hz outputs.
  - System PLL allows CPU operation up to the maximum CPU rate without the need for a high-frequency crystal. May be run from the system oscillator or the internal RC oscillator.
  - Two additional PLLs for generating the USB and SCTimer/PWM clocks.
  - Clock output function with divider that can reflect the crystal oscillator, the main clock, the IRC, or the watchdog oscillator.
- Power control:
  - Integrated PMU (Power Management Unit) to minimize power consumption.
  - Reduced power modes: Sleep mode, Deep-sleep mode, Power-down mode, and Deep power-down mode.
  - APIs provided for optimizing power consumption in active and sleep modes and for configuring Deep-sleep, Power-down, and Deep power-down modes.
  - Wake-up from Deep-sleep and Power-down modes on activity on USB, USART, SPI, and I2C peripherals.

  - Wake-up from Sleep, Deep-sleep, Power-down, and Deep power-down modes from the RTC alarm or wake-up interrupts.
  - Timer-controlled self wake-up from Deep power-down mode using the RTC high-resolution/wake-up 1 kHz timer.
  - Power-On Reset (POR).
  - BrownOut Detect BOD).
- JTAG boundary scan modes supported.
- Unique device serial number for identification.
- Single power supply 2.4 V to 3.6 V.
- Temperature range -40 °C to +105 °C.
- Available as LQFP100, LQFP64, and LQFP48 packages.

1.3 Block diagram



Grey-shaded blocks show peripherals that can provide hardware triggers for DMA transfers or have DMA request lines.

Fig 1. Block diagram

1.4 Functional description

1.4.1 Architectural overview

The ARM Cortex-M3 includes three AHB-Lite buses, one system bus and the I-code and D-code buses. One bus is dedicated for instruction fetch (I-code), and one bus is dedicated for data access (D-code). The use of two core buses allows for simultaneous operations if concurrent operations target different devices.

A multi-layer AHB matrix connects the Cortex-M3 buses and other bus masters to peripherals in a flexible manner that optimizes performance by allowing peripherals on different slaves ports of the matrix to be accessed simultaneously by different bus masters. Details of the multilayer matrix connections are shown in Figure 2.

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

1.4.2 ARM Cortex-M3 processor

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

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

1.4.2.1 Cortex-M3 Configuration Options

The LPC15xx uses the r2p1 version of the Cortex-M3 CPU, which includes a number of configurable options, as noted below.

System options:

- The Nested Vectored Interrupt Controller (NVIC) is included.
- SYSTICK timer is included.
- Single-cycle multiplier supported.
- A Memory Protection Unit (MPU) is included.
- A ROM Table in included. The ROM Table provides addresses of debug components to external debug systems.

Debug related options:

- Serial Wire Debug is included. Serial Wire Debug allows debug operations using only two wires, simple trace functions can be added with a third wire.

- The Data Watchpoint and Trace (DWT) unit is included. The DWT allows data address or data value matches to be trace information or trigger other events. The DWT includes four comparators and counters for certain internal events.
- An Instrumentation Trace Macrocell (ITM) is included. Software can write to the ITM in order to send messages to the trace port.
- The Trace Port Interface Unit (TPIU) is included. The TPIU encodes and provides trace information to the outside world using the serial wire output pin function.

1.4.3 PWM/timer/motor control subsystem

The SCTs (State Configurable Timers) and the analog peripherals support multiple ways of interconnecting their inputs and outputs and of interfacing to the pins and the DMA controller. Using the highly flexible and programmable connection scheme makes it easy to configure various subsystems for motor control and complex timing and tracking applications. Specifically, the inputs to the SCTs and the trigger inputs of the ADCs and DMA are selected through the input mux which offers a choice of many possible sources for each input or trigger. SCT outputs are assigned to pins through the switch matrix allowing for many pinout solutions.

1.4.3.1 PWW/timer subsystem

The SCTs can be configured to build a PWM controller with multiple outputs by programming the MATCH and MATCHRELOAD registers of the SCTs to control the base frequency and the duty cycle of each SCT output. More complex waveforms that span multiple counter cycles or change behavior across or within counter cycles can be generated using the state capability built into the SCT timers.

Combining the PWM functions with the analog functions, the PWM output can react to control signals like comparator outputs or the ADC interrupts. The SCT IPU adds emergency shut-down functions and pre-processing of controlling events. For an overview of the PWM subsystem, see Figure 2 “PWM-Analog subsystem”.

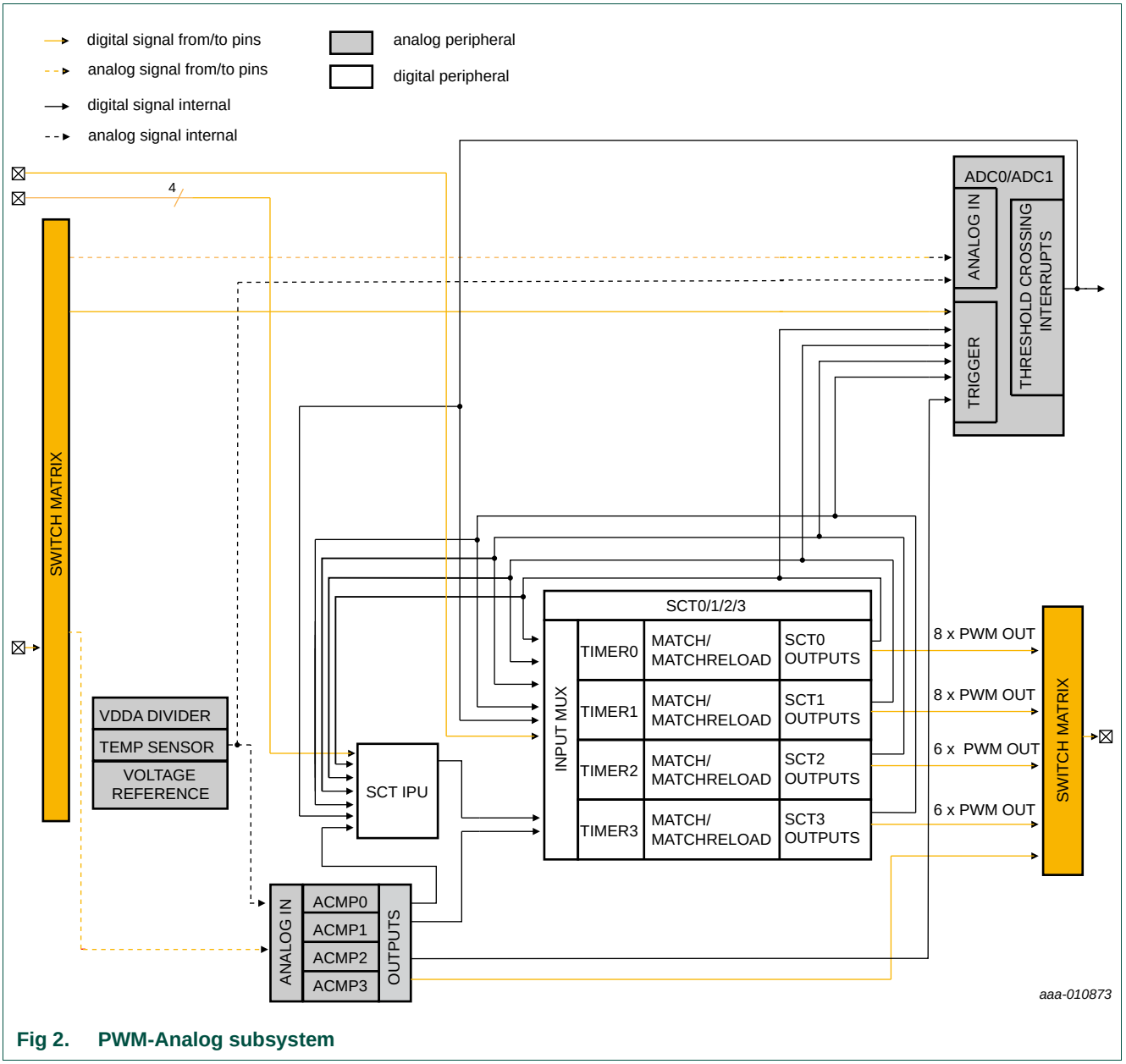


Fig 2. PWM-Analog subsystem

1.4.3.2 Timer controlled subsystem

The timers, the analog components, and the DMA can be configured to form a subsystem that can run independently of the main processor under the control of the SCTs and any events that are generated by the A/D converters, the comparators, the SCT output themselves, or the external pins. A/D conversions can be triggered by the timer outputs, the comparator outputs or by events from external pins. Data can be transferred from the ADCs to memory using the DMA controller, and the DMA transfers can be triggered by the ADCs, the comparator outputs, or by the timer outputs.

For an overview of the subsystem, see Figure 3 “Subsystem with timers, switch matrix, DMA, and analog components”.

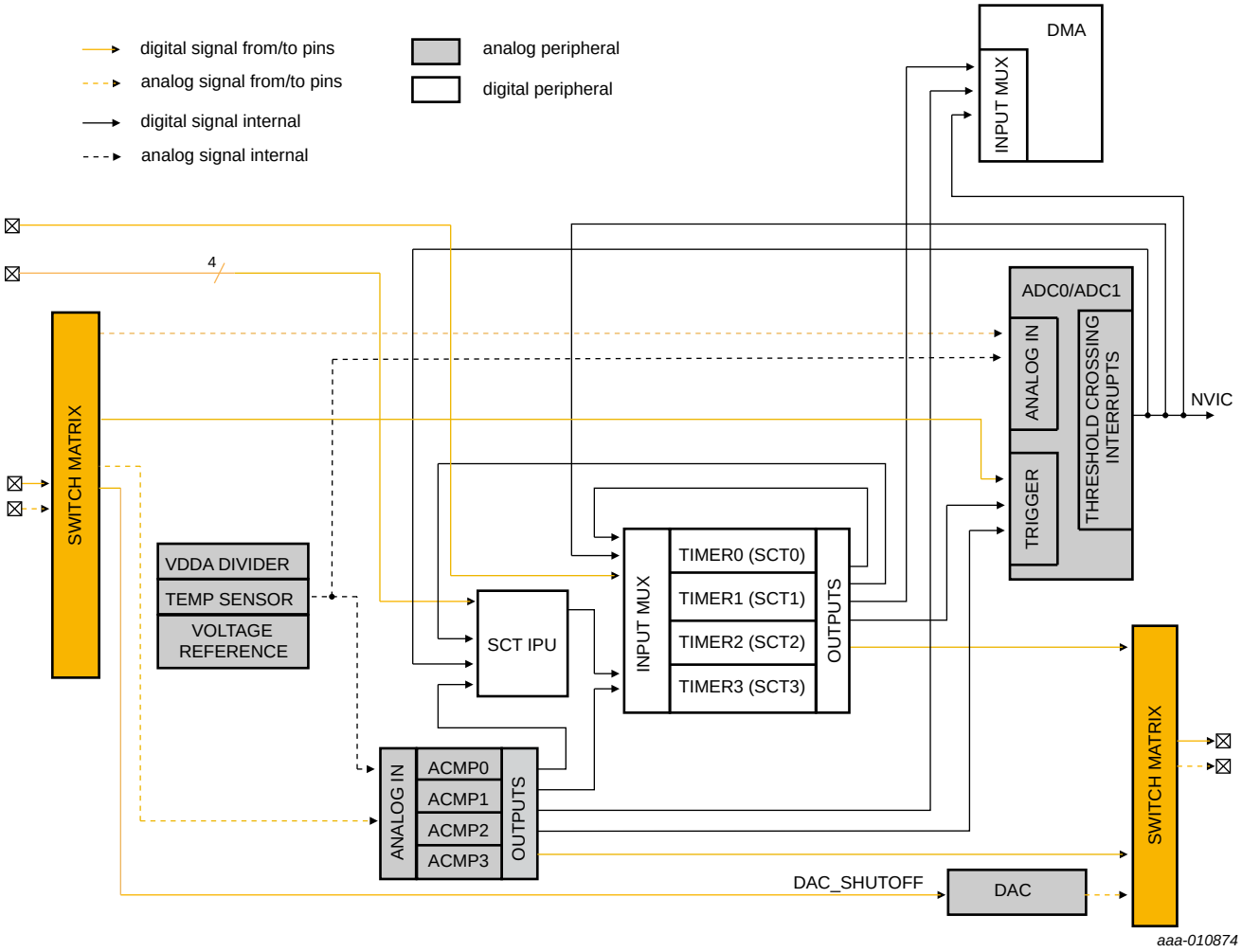


Fig 3. Subsystem with timers, switch matrix, DMA, and analog components

1.1 How to read this chapter

USB is not available on parts LPC1519/18/17.

1.2 General description

The LPC15xx incorporates several distinct memory regions. Figure 1 shows the overall map of the entire address space from the user program viewpoint following reset.

The APB peripheral area is 2 x 512 kB in size and is divided to allow for two blocks of up to 32 peripherals. Each peripheral is allocated 16 kB of space simplifying the address decoding.

The registers incorporated into the ARM Cortex-M3 core, such as NVIC, SysTick, and sleep mode control, are located on the private peripheral bus.

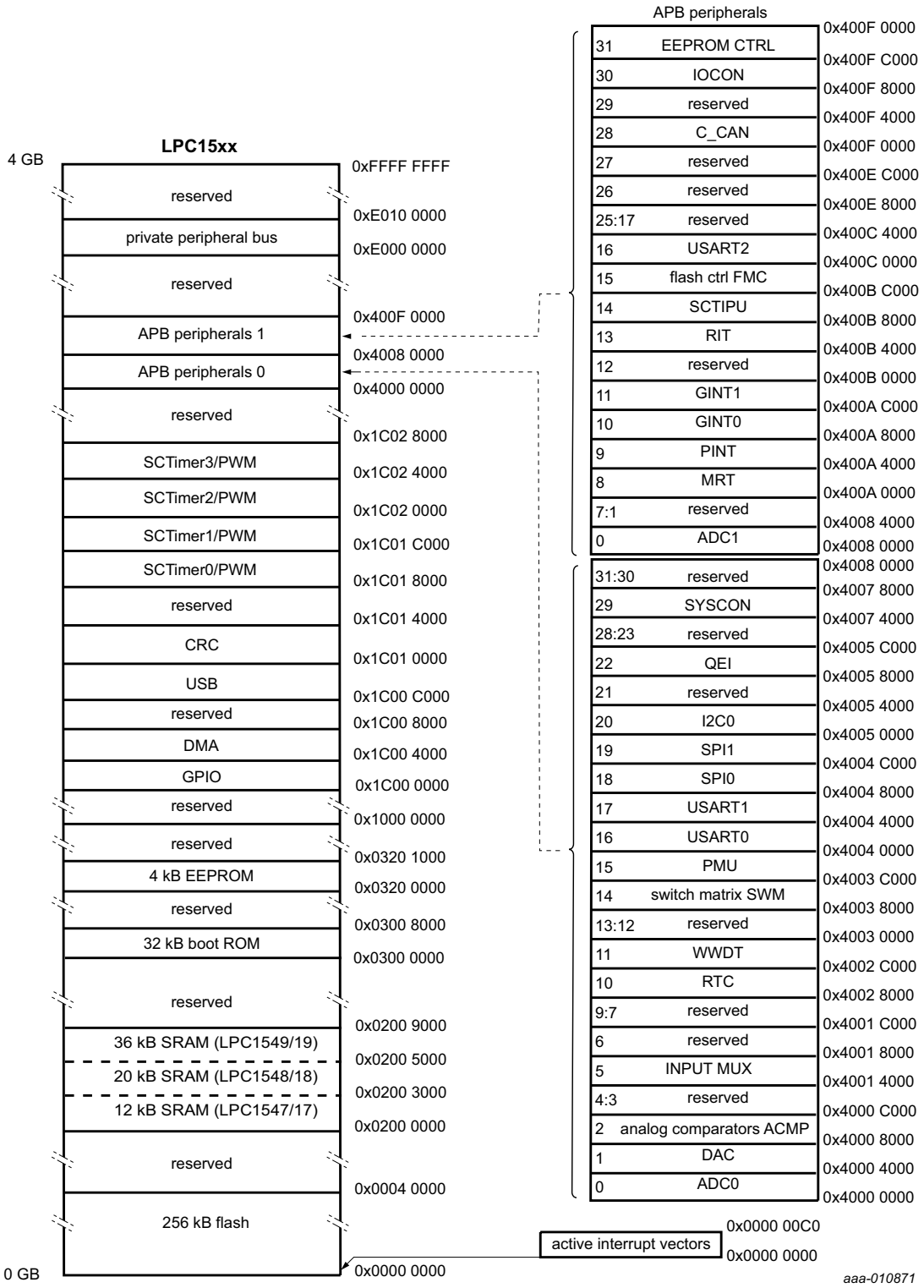
1.2.1 SRAM

The parts contain a total 36 kB, 20 kB or 12 kB of contiguous, on-chip static RAM memory. For each SRAM configuration, the SRAM is divided into three blocks: 2 x 16 kB + 4 kB for 36 kB SRAM, 2 x 8 kB + 4 kB for 20 kB SRAM, and 2 x 4 kB + 4 kB for 12 kB SRAM. The bottom 16 kB, 8 kB, or 4 kB are enabled by the boot loader and cannot be disabled. The next two SRAM blocks in each configuration can be disabled or enabled individually in the SYSCON block to save power. See Section 3.6.22 “System clock control register 0”.

Table 1. LPC15xx SRAM configurations

	SRAM0	SRAM1	SRAM2
LPC1549/19 (total SRAM = 36 kB)			
Address range	0x0200 0000 to 0x0200 3FFF	0x0200 4000 to 0x0200 7FFF	0x0200 8000 to 0x0200 8FFF
Size	16 kB	16 kB	4 kB
Control	cannot be disabled	disable/enable	disable/enable
Default	enabled	enabled	enabled
LPC1548/18 (total SRAM = 20 kB)			
Address range	0x0200 0000 to 0x0200 1FFF	0x0200 2000 to 0x0200 3FFF	0x0200 4000 to 0x0200 4FFF
Size	8 kB	8 kB	4 kB
Control	cannot be disabled	disable/enable	disable/enable
Default	enabled	enabled	enabled
LPC1547/17 (total SRAM = 12 kB)			
Address range	0x0200 0000 to 0x0200 0FFF	0x0200 1000 to 0x0200 1FFF	0x0200 2000 to 0x0200 2FFF
Size	4 kB	4 kB	4 kB
Control	cannot be disabled	disable/enable	disable/enable
Default	enabled	enabled	enabled

1.2.2 Memory mapping



The private peripheral bus includes the ARM Cortex-M3 peripherals such as the NVIC, SysTick, and the core control registers.

Fig 1. Memory mapping

1.2.3 AHB multilayer matrix

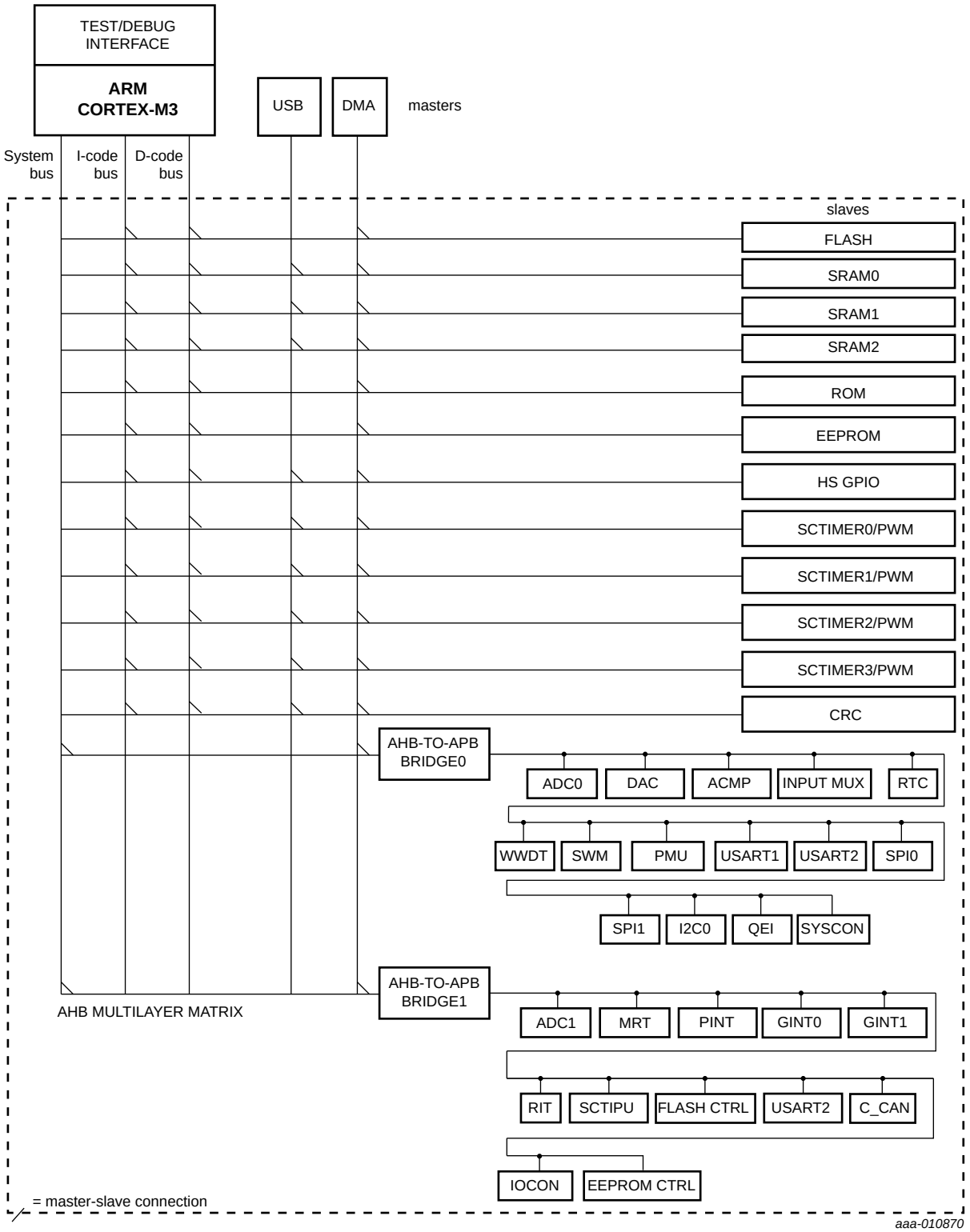


Fig 2. AHB multilayer matrix connections

1.2.4 Memory Protection Unit (MPU)

The Cortex-M3 processor has a memory protection unit (MPU) that provides fine grain memory control, enabling applications to implement security privilege levels, separating code, data and stack on a task-by-task basis. Such requirements are critical in many embedded applications.

The MPU register interface is located on the private peripheral bus and is described in detail in Ref. 1.

2.1 How to read this chapter

USB is available on parts LPC1549/48/47.

2.2 Features

- Nested Vectored Interrupt Controller that is an integral part of the ARM Cortex-M3.
- Tightly coupled interrupt controller provides low interrupt latency.
- Controls system exceptions and peripheral interrupts.
- The NVIC supports 47 vectored interrupts.
- Eight programmable interrupt priority levels with hardware priority level masking.
- Software interrupt generation using the ARM exceptions SVCall and PendSV.
- Support for NMI.
- ARM Cortex-M3 Vector table offset register VTOR implemented.

2.3 General description

The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M3. The tight coupling to the CPU allows for low interrupt latency and efficient processing of late arriving interrupts.

2.3.1 Interrupt sources

Table 2 lists the interrupt sources for each peripheral function. Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller. Each line may represent more than one interrupt source. The interrupt number does not imply any interrupt priority.

See Ref. 1 for a detailed description of the NVIC and the NVIC register description.

Table 2. Connection of interrupt sources to the NVIC

Interrupt number	Name	Description	Flags
0	WDT	Windowed watchdog timer interrupt	WARNINT - watchdog warning interrupt
1	BOD	BOD interrupt	BODINTVAL - BOD interrupt level
2	FLASH	Flash controller	-
3	EE	EEPROM controller interrupt	-
4	DMA	DMA	Interrupt A and interrupt B, error interrupt
5	GINT0	GPIO group0 interrupt	Enabled pin interrupts
6	GINT1	GPIO group1 interrupt	Enabled pin interrupts

Table 2. Connection of interrupt sources to the NVIC

Interrupt number	Name	Description	Flags
7	PIN_INT0	Pin interrupt 0 or pattern match engine slice 0 interrupt	PSTAT - pin interrupt status
8	PIN_INT1	Pin interrupt 1 or pattern match engine slice 1 interrupt	PSTAT - pin interrupt status
9	PIN_INT2	Pin interrupt 2 or pattern match engine slice 2 interrupt	PSTAT - pin interrupt status
10	PIN_INT3	Pin interrupt 3 or pattern match engine slice 3 interrupt	PSTAT - pin interrupt status
11	PIN_INT4	Pin interrupt 4 or pattern match engine slice 4 interrupt	PSTAT - pin interrupt status
12	PIN_INT5	Pin interrupt 5 or pattern match engine slice 5 interrupt	PSTAT - pin interrupt status
13	PIN_INT6	Pin interrupt 6 or pattern match engine slice 6 interrupt	PSTAT - pin interrupt status
14	PIN_INT7	Pin interrupt 7 or pattern match engine slice 7 interrupt	PSTAT - pin interrupt status
15	RIT	RIT interrupt	RITINT; masked compare interrupt
16	SCT0	State configurable timer interrupt	EVFLAG SCT event
17	SCT1	State configurable timer interrupt	EVFLAG SCT event
18	SCT2	State configurable timer interrupt	EVFLAG SCT event
19	SCT3	State configurable timer interrupt	EVFLAG SCT event
20	MRT	Multi-rate timer interrupt	Global MRT interrupt. GFLAG0 GFLAG1 GFLAG2 GFLAG3
21	UART0	USART0 interrupt	See Table 350 “USART Interrupt Enable read and set register (INTENSET, address 0x4004 000C(USART0), 0x4004 400C (USART1), 0x400C 000C (USART2)) bit description”.
22	UART1	USART1 interrupt	Same as UART0
23	UART2	USART2 interrupt	Same as UART0
24	I2C0	I2C0 interrupt	See Table 379 “Interrupt Enable Set and read register (INTENSET, address 0x4005 0008) bit description”.

Table 2. Connection of interrupt sources to the NVIC

Interrupt number	Name	Description	Flags
25	SPI0	SPI0 interrupt	See Table 364 “SPI Interrupt Enable read and Set register (INTENSET, addresses 0x4004 800C (SPI0), 0x4004 C00C (SPI1)) bit description”.
26	SPI1	SPI1 interrupt	Same as SPI0
27	C_CAN0	C_CAN0 interrupt	INTID. See Table 398.
28	USB	USB interrupt	USB_Int_Req. See Table 342.
29	USB_FIQ	USB interrupt	USB_Int_Req_FIQ. See Table 342.
30	USB_WAKEUP	USB wake-up interrupt	USB need_clock signal
31	ADC0_SEQA	ADC0 sequence A completion.	See Table 445.
32	ADC0_SEQB	ADC0 sequence B completion.	See Table 445.
33	ADC0_THCMP	ADC0 threshold compare.	See Table 445.
34	ADC0_OVR	ADC0 overrun.	See Table 445.
35	ADC1_SEQA	ADC1 sequence A completion.	See Table 445.
36	ADC1_SEQB	ADC1 sequence B completion.	See Table 445.
37	ADC1_THCMP	ADC1 threshold compare.	See Table 445.
38	ADC1_OVR	ADC1 overrun.	See Table 445.
39	DAC	DAC interrupt	Internal DMA timer overflow.
40	CMP0	Analog comparator 0 interrupt (ACMP0)	COMPEDGE - rising, falling, or both edges can set the bit.
41	CMP1	Analog comparator 1 interrupt (ACMP1)	COMPEDGE - rising, falling, or both edges can set the bit.
42	CMP2	Analog comparator 2 interrupt (ACMP2)	COMPEDGE - rising, falling, or both edges can set the bit.
43	CMP3	Analog comparator 3 interrupt (ACMP3)	COMPEDGE - rising, falling, or both edges can set the bit.
44	QEI	QEI interrupt	See Table 321.
45	RTC_ALARM	RTC alarm interrupt	ALARM1HZ. See Table 272.
46	RTC_WAKE	RTC wake-up interrupt	WAKEHIRES. See Table 272.

2.4 Register description

The NVIC registers are located on the ARM private peripheral bus.

Table 3. Register overview: NVIC (base address 0xE000 E000)

Name	Access	Address offset	Description	Reset value	Reference
ISER0	R/W	0x100	Interrupt Set Enable Register 0. This register allows enabling interrupts and reading back the interrupt enables for specific peripheral functions.	0	Table 4
ISER1	R/W	0x104	Interrupt Set Enable Register 1.	0	Table 5
ICER0	R/W	0x180	Interrupt Clear Enable Register 0. This register allows disabling interrupts and reading back the interrupt enables for specific peripheral functions.	0	Table 6
ICER1	R/W	0x184	Interrupt Clear Enable Register 1.	0	Table 7
ISPR0	R/W	0x200	Interrupt Set Pending Register 0. This register allows changing the interrupt state to pending and reading back the interrupt pending state for specific peripheral functions.	0	Table 8
ISPR1	R/W	0x204	Interrupt Set Pending Register 1.	0	Table 9
ICPR0	R/W	0x280	Interrupt Clear Pending Register 0. This register allows changing the interrupt state to not pending and reading back the interrupt pending state for specific peripheral functions.	0	Table 10
ICPR1	R/W	0x284	Interrupt Clear Pending Register 1.	0	Table 11
IABR0	R	0x300	Interrupt Active Bit Register 0. This register allows reading the current interrupt active state for specific peripheral functions.	0	Table 12
IABR1	R	0x304	Interrupt Active Bit Register 1.	0	Table 13
IPR0	R/W	0x400	Interrupt Priority Registers 0. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 0 to 3.	0	Table 14
IPR1	R/W	0x404	Interrupt Priority Registers 1 This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 4 to 7.	0	Table 15
IPR2	R/W	0x408	Interrupt Priority Registers 2. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 8 to 11.	0	Table 16
IPR3	R/W	0x40C	Interrupt Priority Registers 3. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 12 to 15.	0	Table 17
IPR4	R/W	0x410	Interrupt Priority Registers 4. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 16 to 19.	0	Table 18
IPR5	R/W	0x414	Interrupt Priority Registers 5. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 20 to 23.	0	Table 19
IPR6	R/W	0x418	Interrupt Priority Registers 6. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 24 to 27.	0	Table 20
IPR7	R/W	0x41C	Interrupt Priority Registers 7. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 28 to 31.	0	Table 21
IPR8	R/W	0x420	Interrupt Priority Registers 8. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 32 to 35.	0	Table 22

Table 3. Register overview: NVIC (base address 0xE000 E000) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
IPR9	R/W	0x424	Interrupt Priority Registers 9. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 36 to 39.	0	Table 23
IPR10	R/W	0x428	Interrupt Priority Registers 10. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 40 to 43.	0	Table 24
IPR11	R/W	0x42C	Interrupt Priority Registers 11. This register allows assigning a priority to each interrupt. This register contains the 3-bit priority fields for interrupts 44 to 46.	0	Table 25
STIR	W	0xF00	Software Trigger Interrupt Register. This register allows software to generate an interrupt.	0	Table 26

2.4.1 Interrupt Set Enable Register 0 register

The ISER0 register allows to enable peripheral interrupts or to read the enabled state of those interrupts. Disable interrupts through the ICER0.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 enables the interrupt.

Read — 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.

Table 4. Interrupt Set Enable Register 0 register (ISER0, address 0xE000 E100) bit description

Bit	Symbol	Description	Reset value
0	ISE_WDT	Interrupt enable.	0
1	ISE_BOD	Interrupt enable.	0
2	ISE_FLASH	Interrupt enable.	0
3	ISE_EE	Interrupt enable.	0
4	ISE_DMA	Interrupt enable.	0
5	ISE_GINT0	Interrupt enable.	0
6	ISE_GINT1	Interrupt enable.	0
7	ISE_PIN_INT0	Interrupt enable.	0
8	ISE_PIN_INT1	Interrupt enable.	0
9	ISE_PIN_INT2	Interrupt enable.	0
10	ISE_PIN_INT3	Interrupt enable.	0
11	ISE_PIN_INT4	Interrupt enable.	0
12	ISE_PIN_INT5	Interrupt enable.	0
13	ISE_PIN_INT6	Interrupt enable.	0
14	ISE_PIN_INT7	Interrupt enable.	0
15	ISE_RIT	Interrupt enable.	0
16	ISE_SCT0	Interrupt enable.	0
17	ISE_SCT1	Interrupt enable.	0
18	ISE_SCT2	Interrupt enable.	0
19	ISE_SCT3	Interrupt enable.	0

Table 4. Interrupt Set Enable Register 0 register (ISER0, address 0xE000 E100) bit description *…continued*

Bit	Symbol	Description	Reset value
20	ISE_MRT	Interrupt enable.	0
21	ISE_UART0	Interrupt enable.	0
22	ISE_UART1	Interrupt enable.	0
23	ISE_UART2	Interrupt enable.	0
24	ISE_I2C0	Interrupt enable.	0
25	ISE_SPI0	Interrupt enable.	0
26	ISE_SPI1	Interrupt enable.	0
27	ISE_CCAN0	Interrupt enable.	0
28	ISE_USB	Interrupt enable.	0
29	ISE_USB_FIQ	Interrupt enable.	0
30	ISE_USB_WAKEKUP	Interrupt enable.	0
31	ISE_ADC0_SEQA	Interrupt enable.	0

2.4.2 Interrupt Set Enable Register 1 register

The ISER1 register allows to enable peripheral interrupts or to read the enabled state of those interrupts. Disable interrupts through the ICER1.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 enables the interrupt.

Read — 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.

Table 5. Interrupt Set Enable Register 1 register (ISER1, address 0xE000 E104) bit description

Bit	Symbol	Description	Reset value
0	ISE_ADC0_SEQB	Interrupt enable.	0
1	ISE_ADC0_THCMP	Interrupt enable.	0
2	ISE_ADC0_OVR	Interrupt enable.	0
3	ISE_ADC1_SEQA	Interrupt enable.	0
4	ISE_ADC1_SEQB	Interrupt enable.	0
5	ISE_ADC1_THCMP	Interrupt enable.	0
6	ISE_ADC1_OVR	Interrupt enable.	0
7	ISE_DAC	Interrupt enable.	0
8	ISE_ACMP0	Interrupt enable.	0
9	ISE_ACMP1	Interrupt enable.	0
10	ISE_ACMP2	Interrupt enable.	0
11	ISE_ACMP3	Interrupt enable.	0
12	ISE_QEI	Interrupt enable.	0
13	ISE_RTC_ALARM	Interrupt enable.	0
14	ISE_RTC_WAKE	Interrupt enable.	0
31:15	-	Reserved	0

2.4.3 Interrupt clear enable register 0

The ICER0 register allows disabling the peripheral interrupts, or for reading the enabled state of those interrupts. Enable interrupts through the ISER0 register.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 disables the interrupt.

Read — 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.

Table 6. Interrupt clear enable register 0 (ICER0, address 0xE000 E180)

Bit	Symbol	Description	Reset value
0	ICE_WDT	Interrupt disable.	0
1	ICE_BOD	Interrupt disable.	0
2	ICE_FLASH	Interrupt disable.	0
3	ICE_EE	Interrupt disable.	0
4	ICE_DMA	Interrupt disable.	0
5	ICE_GINT0	Interrupt disable.	0
6	ICE_GINT1	Interrupt disable.	0
7	ICE_PIN_INT0	Interrupt disable.	0
8	ICE_PIN_INT1	Interrupt disable.	0
9	ICE_PIN_INT2	Interrupt disable.	0
10	ICE_PIN_INT3	Interrupt disable.	0
11	ICE_PIN_INT4	Interrupt disable.	0
12	ICE_PIN_INT5	Interrupt disable.	0
13	ICE_PIN_INT6	Interrupt disable.	0
14	ICE_PIN_INT7	Interrupt disable.	0
15	ICE_RIT	Interrupt disable.	0
16	ICE_SCT0	Interrupt disable.	0
17	ICE_SCT1	Interrupt disable.	0
18	ICE_SCT2	Interrupt disable.	0
19	ICE_SCT3	Interrupt disable.	0
20	ICE_MRT	Interrupt disable.	0
21	ICE_UART0	Interrupt disable.	0
22	ICE_UART1	Interrupt disable.	0
23	ICE_UART2	Interrupt disable.	0
24	ICE_I2C0	Interrupt disable.	0
25	ICE_SPI0	Interrupt disable.	0
26	ICE_SPI1	Interrupt disable.	0
27	ICE_CCAN0	Interrupt disable.	0
28	ICE_USB	Interrupt disable.	0
29	ICE_USB_FIQ	Interrupt disable.	0
30	ICE_USB_WAKEKUP	Interrupt disable.	0
31	ICE_ADC0_SEQA	Interrupt disable.	0

2.4.4 Interrupt clear enable register 1

The ICER1 register allows disabling the peripheral interrupts, or for reading the enabled state of those interrupts. Enable interrupts through the ISER1 register.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 disables the interrupt.

Read — 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.

Table 7. Interrupt clear enable register1 (ICER1, address 0xE000 E184)

Bit	Symbol	Description	Reset value
0	ICE_ADC0_SEQB	Interrupt disable.	0
1	ICE_ADC0_THCMP	Interrupt disable.	0
2	ICE_ADC0_OVR	Interrupt disable.	0
3	ICE_ADC1_SEQA	Interrupt disable.	0
4	ICE_ADC1_SEQB	Interrupt disable.	0
5	ICE_ADC1_THCMP	Interrupt disable.	0
6	ICE_ADC1_OVR	Interrupt disable.	0
7	ICE_DAC	Interrupt disable.	0
8	ICE_ACMP0	Interrupt disable.	0
9	ICE_ACMP1	Interrupt disable.	0
10	ICE_ACMP2	Interrupt disable.	0
11	ICE_ACMP3	Interrupt disable.	0
12	ICE_QEI	Interrupt disable.	0
13	ICE_RTC_ALARM	Interrupt disable.	0
14	ICE_RTC_WAKE	Interrupt disable.	0
31:15	-	Reserved	0

2.4.5 Interrupt Set Pending Register 0 register

The ISPR0 register allows setting the pending state of the peripheral interrupts, or for reading the pending state of those interrupts. Clear the pending state of interrupts through the ICPR0 registers.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 changes the interrupt state to pending.

Read — 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.

Table 8. Interrupt set pending register 0 register (ISPR0, address 0xE000 E200) bit description

Bit	Symbol	Description	Reset value
0	ISP_WDT	Interrupt pending set.	0
1	ISP_BOD	Interrupt pending set.	0
2	ISP_FLASH	Interrupt pending set.	0
3	ISP_EE	Interrupt pending set.	0
4	ISP_DMA	Interrupt pending set.	0

Table 8. Interrupt set pending register 0 register (ISPR0, address 0xE000 E200) bit description *…continued*

Bit	Symbol	Description	Reset value
5	ISP_GINT0	Interrupt pending set.	0
6	ISP_GINT1	Interrupt pending set.	0
7	ISP_PIN_INT0	Interrupt pending set.	0
8	ISP_PIN_INT1	Interrupt pending set.	0
9	ISP_PIN_INT2	Interrupt pending set.	0
10	ISP_PIN_INT3	Interrupt pending set.	0
11	ISP_PIN_INT4	Interrupt pending set.	0
12	ISP_PIN_INT5	Interrupt pending set.	0
13	ISP_PIN_INT6	Interrupt pending set.	0
14	ISP_PIN_INT7	Interrupt pending set.	0
15	ISP_RIT	Interrupt pending set.	0
16	ISP_SCT0	Interrupt pending set.	0
17	ISP_SCT1	Interrupt pending set.	0
18	ISP_SCT2	Interrupt pending set.	0
19	ISP_SCT3	Interrupt pending set.	0
20	ISP_MRT	Interrupt pending set.	0
21	ISP_UART0	Interrupt pending set.	0
22	ISP_UART1	Interrupt pending set.	0
23	ISP_UART2	Interrupt pending set.	0
24	ISP_I2C0	Interrupt pending set.	0
25	ISP_SPI0	Interrupt pending set.	0
26	ISP_SPI1	Interrupt pending set.	0
27	ISP_CCAN0	Interrupt pending set.	0
28	ISP_USB	Interrupt pending set.	0
29	ISP_USB_FIQ	Interrupt pending set.	0
30	ISP_USB_WAKEKUP	Interrupt pending set.	0
31	ISP_ADC0_SEQA	Interrupt pending set.	0

2.4.6 Interrupt Set Pending Register 1 register

The ISPR1 register allows setting the pending state of the peripheral interrupts, or for reading the pending state of those interrupts. Clear the pending state of interrupts through the ICPR1 registers.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 changes the interrupt state to pending.

Read — 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.

Table 9. Interrupt set pending register 1 register (ISPR1, address 0xE000 E204) bit description

Bit	Symbol	Description	Reset value
0	ISP_ADC0_SEQB	Interrupt pending set.	0
1	ISP_ADC0_THCMP	Interrupt pending set.	0
2	ISP_ADC0_OVR	Interrupt pending set.	0
3	ISP_ADC1_SEQA	Interrupt pending set.	0
4	ISP_ADC1_SEQB	Interrupt pending set.	0
5	ISP_ADC1_THCMP	Interrupt pending set.	0
6	ISP_ADC1_OVR	Interrupt pending set.	0
7	ISP_DAC	Interrupt pending set.	0
8	ISP_ACMP0	Interrupt pending set.	0
9	ISP_ACMP1	Interrupt pending set.	0
10	ISP_ACMP2	Interrupt pending set.	0
11	ISP_ACMP3	Interrupt pending set.	0
12	ISP_QEI	Interrupt pending set.	0
13	ISP_RTC_ALARM	Interrupt pending set.	0
14	ISP_RTC_WAKE	Interrupt pending set.	0
31:15	-	Reserved	0

2.4.7 Interrupt Clear Pending Register 0 register

The ICPR0 register allows clearing the pending state of the peripheral interrupts, or for reading the pending state of those interrupts. Set the pending state of interrupts through the ISPR0 register.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 changes the interrupt state to not pending.

Read — 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.

Table 10. Interrupt clear pending register 0 register (ICPR0, address 0xE000 E280) bit description

Bit	Symbol	Function	Reset value
0	ICP_WDT	Interrupt pending clear.	0
1	ICP_BOD	Interrupt pending clear.	0
2	ICP_FLASH	Interrupt pending clear.	0
3	ICP_EE	Interrupt pending clear.	0
4	ICP_DMA	Interrupt pending clear.	0
5	ICP_GINT0	Interrupt pending clear.	0
6	ICP_GINT1	Interrupt pending clear.	0
7	ICP_PIN_INT0	Interrupt pending clear.	0
8	ICP_PIN_INT1	Interrupt pending clear.	0
9	ICP_PIN_INT2	Interrupt pending clear.	0
10	ICP_PIN_INT3	Interrupt pending clear.	0

Table 10. Interrupt clear pending register 0 register (ICPR0, address 0xE000 E280) bit description *…continued*

Bit	Symbol	Function	Reset value
11	ICP_PIN_INT4	Interrupt pending clear.	0
12	ICP_PIN_INT5	Interrupt pending clear.	0
13	ICP_PIN_INT6	Interrupt pending clear.	0
14	ICP_PIN_INT7	Interrupt pending clear.	0
15	ICP_RIT	Interrupt pending clear.	0
16	ICP_SCT0	Interrupt pending clear.	0
17	ICP_SCT1	Interrupt pending clear.	0
18	ICP_SCT2	Interrupt pending clear.	0
19	ICP_SCT3	Interrupt pending clear.	0
20	ICP_MRT	Interrupt pending clear.	0
21	ICP_UART0	Interrupt pending clear.	0
22	ICP_UART1	Interrupt pending clear.	0
23	ICP_UART2	Interrupt pending clear.	0
24	ICP_I2C0	Interrupt pending clear.	0
25	ICP_SPI0	Interrupt pending clear.	0
26	ICP_SPI1	Interrupt pending clear.	0
27	ICP_CCAN0	Interrupt pending clear.	0
28	ICP_USB	Interrupt pending clear.	0
29	ICP_USB_FIQ	Interrupt pending clear.	0
30	ICP_USB_WAKEKUP	Interrupt pending clear.	0
31	ICP_ADC0_SEQA	Interrupt pending clear.	0

2.4.8 Interrupt Clear Pending Register 1 register

The ICPR1 register allows clearing the pending state of the peripheral interrupts, or for reading the pending state of those interrupts. Set the pending state of interrupts through the ISPR1 register.

The bit description is as follows for all bits in this register:

Write — Writing 0 has no effect, writing 1 changes the interrupt state to not pending.

Read — 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.

Table 11. Interrupt clear pending register 1 register (ICPR1, address 0xE000 E284) bit description

Bit	Symbol	Function	Reset value
0	ICP_ADC0_SEQB	Interrupt pending clear.	0
1	ICP_ADC0_THCMP	Interrupt pending clear.	0
2	ICP_ADC0_OVR	Interrupt pending clear.	0
3	ICP_ADC1_SEQA	Interrupt pending clear.	0
4	ICP_ADC1_SEQB	Interrupt pending clear.	0
5	ICP_ADC1_THCMP	Interrupt pending clear.	0

Table 11. Interrupt clear pending register 1 register (ICPR1, address 0xE000 E284) bit description *…continued*

Bit	Symbol	Function	Reset value
6	ICP_ADC1_OVR	Interrupt pending clear.	0
7	ICP_DAC	Interrupt pending clear.	0
8	ICP_ACMP0	Interrupt pending clear.	0
9	ICP_ACMP1	Interrupt pending clear.	0
10	ICP_ACMP2	Interrupt pending clear.	0
11	ICP_ACMP3	Interrupt pending clear.	0
12	ICP_QEI	Interrupt pending clear.	0
13	ICP_RTC_ALARM	Interrupt pending clear.	0
14	ICP_RTC_WAKE	Interrupt pending clear.	0
31:15	-	Reserved	0

2.4.9 Interrupt Active Bit Register 0

The IABR0 register is a read-only register that allows reading the active state of the peripheral interrupts. Use this register to determine which peripherals are asserting an interrupt to the NVIC and may also be pending if there are enabled.

The bit description is as follows for all bits in this register:

Write — n/a.

Read — 0 indicates that the interrupt is not active, 1 indicates that the interrupt is active.

Table 12. Interrupt Active Bit Register 0 (IABR0, address 0xE000 E300) bit description

Bit	Symbol	Function	Reset value
0	IAB_WDT	Interrupt active state.	0
1	IAB_BOD	Interrupt active state.	0
2	IAB_FLASH	Interrupt active state.	0
3	IAB_EE	Interrupt active state.	0
4	IAB_DMA	Interrupt active state.	0
5	IAB_GINT0	Interrupt active state.	0
6	IAB_GINT1	Interrupt active state.	0
7	IAB_PIN_INT0	Interrupt active state.	0
8	IAB_PIN_INT1	Interrupt active state.	0
9	IAB_PIN_INT2	Interrupt active state.	0
10	IAB_PIN_INT3	Interrupt active state.	0
11	IAB_PIN_INT4	Interrupt active state.	0
12	IAB_PIN_INT5	Interrupt active state.	0
13	IAB_PIN_INT6	Interrupt active state.	0
14	IAB_PIN_INT7	Interrupt active state.	0
15	IAB_RIT	Interrupt active state.	0
16	IAB_SCT0	Interrupt active state.	0
17	IAB_SCT1	Interrupt active state.	0
18	IAB_SCT2	Interrupt active state.	0

Table 12. Interrupt Active Bit Register 0 (IABR0, address 0xE000 E300) bit description

Bit	Symbol	Function	Reset value
19	IAB_SCT3	Interrupt active state.	0
20	IAB_MRT	Interrupt active state.	0
21	IAB_UART0	Interrupt active state.	0
22	IAB_UART1	Interrupt active state.	0
23	IAB_UART2	Interrupt active state.	0
24	IAB_I2C0	Interrupt active state.	0
25	IAB_SPI0	Interrupt active state.	0
26	IAB_SPI1	Interrupt active state.	0
27	IAB_CCAN0	Interrupt active state.	0
28	IAB_USB	Interrupt active state.	0
29	IAB_USB_FIQ	Interrupt active state.	0
30	IAB_USB_WAKEKUP	Interrupt active state.	0
31	IAB_ADC0_SEQA	Interrupt active state.	0

2.4.10 Interrupt Active Bit Register 1

The IABR1 register is a read-only register that allows reading the active state of the peripheral interrupts. Use this register to determine which peripherals are asserting an interrupt to the NVIC and may also be pending if there are enabled.

The bit description is as follows for all bits in this register:

Write — n/a.

Read — 0 indicates that the interrupt is not active, 1 indicates that the interrupt is active.

Table 13. Interrupt Active Bit Register 1 (IABR1, address 0xE000 E304) bit description

Bit	Symbol	Function	Reset value
0	IAB_ADC0_SEQB	Interrupt active state.	0
1	IAB_ADC0_THCMP	Interrupt active state.	0
2	IAB_ADC0_OVR	Interrupt active state.	0
3	IAB_ADC1_SEQA	Interrupt active state.	0
4	IAB_ADC1_SEQB	Interrupt active state.	0
5	IAB_ADC1_THCMP	Interrupt active state.	0
6	IAB_ADC1_OVR	Interrupt active state.	0
7	IAB_DAC	Interrupt active state.	0
8	IAB_ACMP0	Interrupt active state.	0
9	IAB_ACMP1	Interrupt active state.	0
10	IAB_ACMP2	Interrupt active state.	0
11	IAB_ACMP3	Interrupt active state.	0
12	IAB_QEI	Interrupt active state.	0
13	IAB_RTC_ALARM	Interrupt active state.	0
14	IAB_RTC_WAKE	Interrupt active state.	0
31:15	-	Reserved	0

2.4.11 Interrupt Priority Register 0

The IPR0 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 14. Interrupt Priority Register 0 (IPR0, address 0xE000 E400) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_WDT	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_BOD	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_FLASH	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_EE	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.12 Interrupt Priority Register 1

The IPR0 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 15. Interrupt Priority Register 1 (IPR1, address 0xE000 E404) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_DMA	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_GINT0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_GINT1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_PIN_INT0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.13 Interrupt Priority Register 2

The IPR0 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 16. Interrupt Priority Register 2 (IPR2, address 0xE000 E408) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_PIN_INT1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_PIN_INT2	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_PIN_INT3	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_PIN_INT4	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.14 Interrupt Priority Register 3

The IPR3 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 17. Interrupt Priority Register 3 (IPR3, address 0xE000 E40C) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_PIN_INT5	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_PIN_INT6	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_PIN_INT7	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_RIT	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.15 Interrupt Priority Register 4

The IPR4 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 18. Interrupt Priority Register 4 (IPR4, address 0xE000 E410) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_SCT0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_SCT1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_SCT2	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_SCT3	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.16 Interrupt Priority Register 5

The IPR5 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 19. Interrupt Priority Register 5 (IPR5, address 0xE000 E414) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_MRT	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_UART0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_UART1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_UART2	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.17 Interrupt Priority Register 6

The IPR6 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 20. Interrupt Priority Register 6 (IPR6, address 0xE000 E418) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_I2C0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_SPI0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_SPI1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_C_CAN0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.18 Interrupt Priority Register 7

The IPR7 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 21. Interrupt Priority Register 7 (IPR7, address 0xE000 E41C) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_USB	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_USB_FIQ	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_WAKEUP	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_ADC0_SEQA	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.19 Interrupt Priority Register 8

The IPR8 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 22. Interrupt Priority Register 8 (IPR8, address 0xE000 E420) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_ADC0_SEQB	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_ADC0_THCMP	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_ADC0_OVR	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_ADC1_SEQA	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.20 Interrupt Priority Register 9

The IPR9 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 23. Interrupt Priority Register 9 (IPR9, address 0xE000 E424) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_ADC1_SEQB	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_ADC1_THCMP	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_ADC1_OVR	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_DAC	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.21 Interrupt Priority Register 10

The IPR10 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 24. Interrupt Priority Register 10 (IPR10, address 0xE000 E428) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_ACMP0	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_ACMP1	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_ACMP2	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	IP_ACMP3	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0

2.4.22 Interrupt Priority Register 11

The IPR11 register controls the priority of four peripheral interrupts. Each interrupt can have one of 8 priorities, where 0 is the highest priority.

Table 25. Interrupt Priority Register 11 (IPR11, address 0xE000 E42C) bit description

Bit	Symbol	Description	Reset value
4:0	-	These bits ignore writes, and read as 0.	0
7:5	IP_QEI	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
12:8	-	These bits ignore writes, and read as 0.	0
15:13	IP_RTC_ALARM	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
20:16	-	These bits ignore writes, and read as 0.	0
23:21	IP_RTC_WAKE	Interrupt Priority. 0 = highest priority. 7 = lowest priority.	0
28:24	-	These bits ignore writes, and read as 0.	0
31:29	-	Reserved	0

2.4.23 Software Trigger Interrupt Register

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

By default, only privileged software can write to the STIR register. Unprivileged software can be given this ability if privileged software sets the USERSETMPEND bit in the ARM Cortex-M3 CCR register.

The interrupt number to be programmed in this register is listed in Table 2.

Table 26. Software Trigger Interrupt Register (STIR, address 0xE000 EF00) bit description

Bit	Symbol	Description
8:0	INTID	Writing a value to this field generates an interrupt for the specified the interrupt number. The range allowed for this part is 0 to 46.
31:9	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

3.1 How to read this chapter

USB and USB PLL related registers are available only on parts LPC1549/48/47.

3.2 Features

- System and bus configuration.
- Clock select and control.
- Reset control.
- Wake-up control.
- BOD configuration.
- High-accuracy frequency measurement function for on-chip and off-chip clocks.
- Uses a selection of on-chip clocks as reference clock.
- Device ID register.

3.3 Basic configuration

Configure the SYSCON block as follows:

- The SYSCON uses the CLKOUT, $\overline{\text{RESET}}$ pins which can be configured through the switch matrix. See Section 3.4. $\overline{\text{RESET}}$ is enabled by default.
- No clock configuration is needed. The clock to the SYSCON block is always enabled. By default, the SYSCON block is clocked by the IRC.
- Target and reference clocks for the frequency measurement function are selected in the input mux block. See Table 134.

3.3.1 Set up the PLL

The PLL creates a stable output clock at a higher frequency than the input clock. If you need a main clock with a frequency higher than the 12 MHz IRC clock, use the PLL to boost the input frequency.

1. Power up the system PLL in the PDRUNCFG register.

   Section 3.6.47 “Power configuration register”
2. Select the PLL input in the SYSPLLCLKSEL register. You have the following input options:
   - IRC: 12 MHz internal oscillator.
   - System oscillator: External crystal oscillator using the XTALIN/XTALOUT pins.

   Section 3.6.18 “System PLL clock source select register”
3. Configure the PLL M and N dividers.

   Section 3.6.40 “System PLL control register”
4. Wait for the PLL to lock by monitoring the PLL lock status.

Section 3.6.41 “System PLL status register”

3.3.2 Configure the main clock and system clock

The clock source for the registers and memories is derived from main clock. The main clock can be sourced from the IRC at a fixed clock frequency of 12 MHz or from the PLL.

The divided main clock is called the system clock and clocks the core, the memories, and the peripherals (register interfaces and peripheral clocks).

1. Select the main clock. You have the following options:
   - IRC: 12 MHz internal oscillator (default)
   - System oscillator
   - Watchdog oscillator
   - The output of the system PLL
   - The RTC 32 kHz oscillator

   Section 3.6.12 “Main clock source select register A” and Section 3.6.13 “Main clock source select register B”.
2. Select the divider value for the system clock. A divider value of 0 disables the system clock.

   Section 3.6.21 “System clock divider register”
3. Select the memories and peripherals that are operating in your application and therefore must have an active clock. The core is always clocked.

   Section 3.6.22 “System clock control register 0” and Section 3.6.23 “System clock control register 1”.

3.3.3 Set up the system oscillator using XTALIN and XTALOUT

To use the system oscillator, follow these steps:

1. Connect the XTALIN and XTALOUT pins to an external crystal.
2. In the SYSOSCCTRL register, disable the BYPASS bit and select the oscillator frequency range according to the desired oscillator output clock. See Table 66 “System oscillator control register (SYSOSCCTRL, address 0x4007 4188) bit description”.

3.3.4 Measure the frequency of a clock signal

The frequency of any on-chip or off-chip clock signal can be measured accurately with a selectable reference clock. For example, the frequency measurement function can be used to accurately determine the frequency of the watchdog oscillator which varies over a wide range depending on process and temperature.

The clock frequency to be measured and the reference clock are selected in the input mux block. See Table 134 “Frequency measure function frequency clock select register (FREQMEAS_REF, address 0x4001 4160) bit description” and Table 135 “Frequency measure function target clock select register (FREQMEAS_TARGET, address 0x4001 4164) bit description”.

Details on the accuracy and measurement process are described in Section 3.7.5 “Frequency measure function”.

To start a frequency measurement cycle and read the result, see Table 59 “Frequency measure function control register (FREQMECTRL, address 0x4007 4120) bit description”.

3.4 Pin description

Table 27. SYSCON pin description

Function	Direction	Pin	Description	SWM register	Reference
CLKOUT	O	any	CLKOUT clock output.	PINASSIGN13	Table 120
XTALIN	I	XTALIN	Input to the system oscillator.	n/a	-
XTALOUT	O	XTALOUT	Output from the system oscillator.	n/a	-
$\overline{\text{RESET}}$	I	$\overline{\text{RESET}}$/PIO0_21	External reset input	PINENABLE1	Table 124

3.5 General description

3.5.1 Clock generation

The system control block facilitates the clock generation. Except for the peripheral clocks (SYSTICK clock, fractional baud rate generator clock, glitch filter clock, ARM trace clock), the clocks to the core and peripherals run at the same frequency. The maximum clock frequency is 72 MHz. See Figure 3.

The low-power watchdog oscillator provides a fixed 503 kHz clock. The accuracy of this clock is limited to +/- 40 % over temperature and processing. To determine the actual watchdog oscillator output, use the frequency measure block. See Section 3.3.4.

The part contains three identical PLLs. The system PLL produces the clock for the core and peripherals. Two additional PLLs can create independent clocks for the USB and the SCT. The output of all three PLLs can be monitored through the CLKOUT pin.

You can select the output of any of the three PLLs to derive an asynchronous ADC sampling clock. See also Section 28.3.

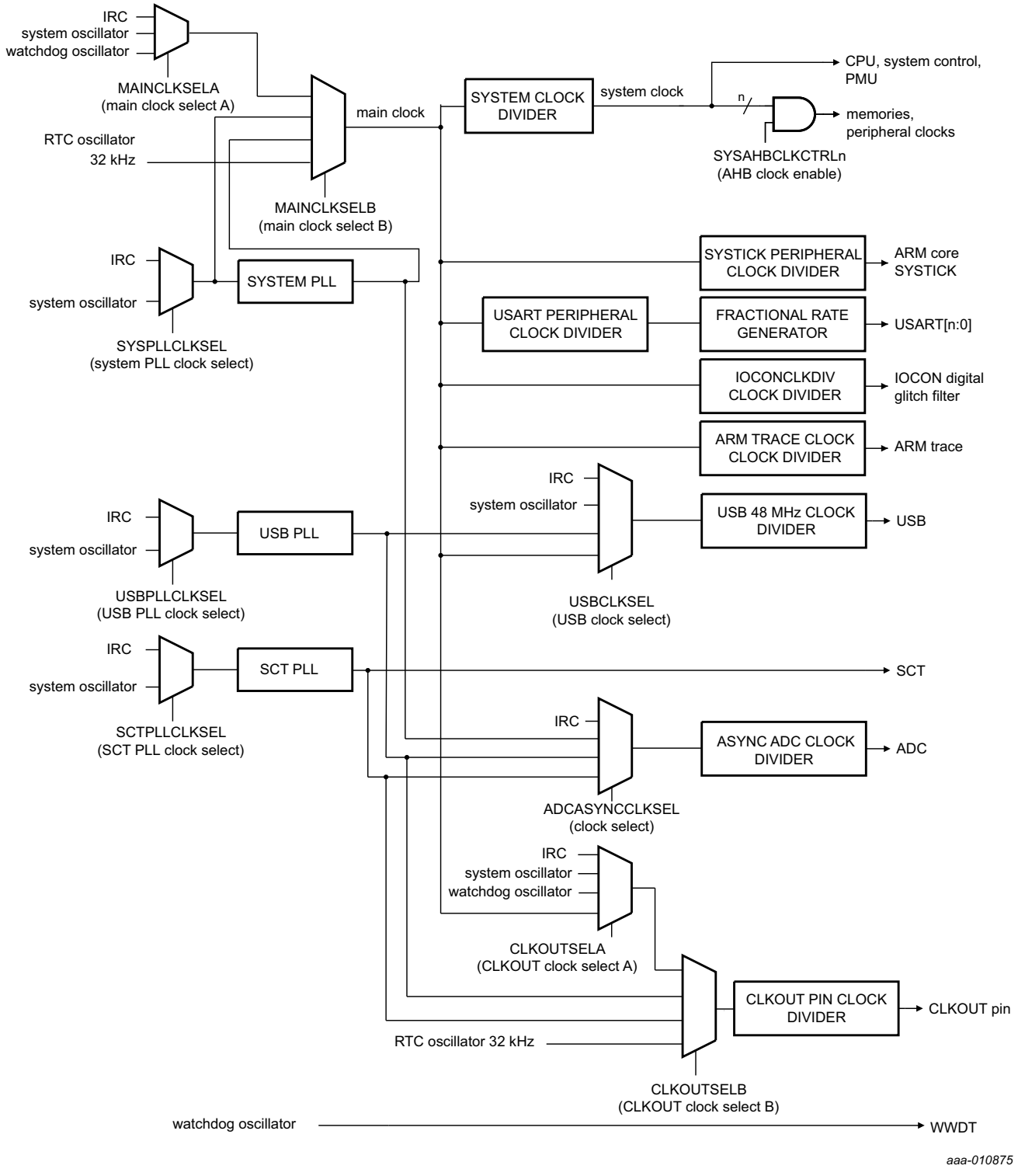


Fig 3. Clock generation

3.6 Register description

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

All address offsets not shown in Table 28 are reserved and should not be written to. The reset value after boot shows the reset value seen when the boot loader executes but does not enter ISP mode, that is the flash contains valid user code.

Table 28. Register overview: System configuration (base address 0x4007 4000)

Name	Access	Offset	Description	Reset value	Reset value after boot	Reference
SYSMEMREMAP	R/W	0x000	System memory remap	0x0	0x2	Table 29
-	-	0x004	Reserved	-	-	-
-	-	0x008	Reserved	-	-	-
AHBBUFEN0	R/W	0x00C	AHB-to-APB bridge 0 write buffering control			
AHBBUFEN1	R/W	0x010	AHB-to-APB bridge 1 write buffering control			
SYSTCKCAL	R/W	0x014	System tick counter calibration	0x0	0x0	Table 32
-	R/W	0x018	Reserved	-	-	-
NMISRC	R/W	0x01C	NMI Source Control	0x0	0x0	Table 33
-	-	0x020 - 0x03C	Reserved	-	-	-
SYSRSTSTAT	R/W	0x040	System reset status register	0x0	0x0	Table 34
PRESETCTRL0	R/W	0x044	Peripheral reset control 0	0x0	0x0	Table 35
PRESETCTRL1	R/W	0x048	Peripheral reset control 1	0x0	0x0	Table 36
PIOPORCAP0	R	0x04C	POR captured PIO status 0	user dependent	user dependent	Table 37
PIOPORCAP1	R	0x050	POR captured PIO status 1	user dependent	user dependent	Table 38
PIOPORCAP2	R	0x054	POR captured PIO status 2	user dependent	user dependent	Table 39
-	-	0x058 - 0x07C	Reserved	-	-	-
MAINCLKSELA	R/W	0x080	Main clock source select A	0x0	0x0	Table 40
MAINCLKSELB	R/W	0x084	Main clock source select B	0x0	0x0	Table 41
USBCLKSEL	R/W	0x088	USB clock source select	0x0	0x0	Table 42
ADCASYNCCLKSEL	R/W	0x08C	ADC asynchronous clock source select	0x0	0x0	Table 43
-	R/W	0x090	Reserved	-	-	-
CLKOUTSELA	R/W	0x094	CLKOUT clock source select A	0x0	0x0	Table 44
CLKOUTSELB	R/W	0x098	CLKOUT clock source select B	0x0	0x0	Table 45
-	R/W	0x09C	Reserved	-	-	-
SYSPLLCLKSEL	R/W	0x0A0	System PLL clock source select	0x0	0x0	Table 46
USBPLLCLKSEL	R/W	0x0A4	USB PLL clock source select	0x0	0x0	Table 47

Table 28. Register overview: System configuration (base address 0x4007 4000) *…continued*

Name	Access	Offset	Description	Reset value	Reset value after boot	Reference
SCTPLLCLKSEL	R/W	0x0A8	SCT PLL clock source select	0x0	0x0	Table 48
-	-	0x0AC - 0x0BC	Reserved	-	-	-
SYSAHBCLKDIV	R/W	0x0C0	System clock divider	0x1	0x1	Table 49
SYSAHBCLKCTRL0	R/W	0x0C4	System clock control 0	0x1	0x19B	Table 50
SYSAHBCLKCTRL1	R/W	0x0C8	System clock control 1	0x0	0x0	Table 51
SYSTICKCLKDIV	R/W	0x0CC	SYSTICK clock divider	0x0	0x0	Table 52
UARTCLKDIV	R/W	0x0D0	USART clock divider. Clock divider for the USART fractional baud rate generator.	0x0	0x0	Table 53
IOCONCLKDIV	R/W	0x0D4	Peripheral clock to the IOCON block for programmable glitch filter	0x0	0x0	Table 54
TRACECLKDIV	R/W	0x0D8	ARM trace clock divider	0x0	0x0	Table 55
-	-	0x0DC - 0x0E8	Reserved	-	-	-
USBCLKDIV	R/W	0x0EC	USB clock divider	0x0	0x0	Table 56
ADCASYNCCLKDIV	R/W	0x0F0	Asynchronous ADC clock divider	0x0	0x0	Table 57
-	-	0x0F4	Reserved	-	-	-
CLKOUTDIV	R/W	0x0F8	CLKOUT clock divider	0x0	0x0	Table 58
-	-	0x0FC - 0x11C	Reserved	-	-	-
FREQMECTRL	R/W	0x120	Frequency measure register	0x0	0x0	Table 59
FLASHCFG	R/W	0x124	Flash waitstates configuration register	-	0x201A	Table 60
FRGCTRL	R/W	0x128	USART fractional baud rate generator control	0xFF	0xFF	Table 61
USBCLKCTRL	R/W	0x12C	USB clock control	0x0	0x0	Table 62
USBCLKST	R/W	0x130	USB clock status	0x1	0x1	Table 63
-	-	0x134 - 0x17C	Reserved			
BODCTRL	R/W	0x180	Brown-Out Detect	0x0	0x0	Table 64
IRCCTRL	R/W	0x184	IRC trim value	part dependent	part dependent	Table 65
SYSOSCCTRL	R/W	0x188	System oscillator control	0x0	0x0	Table 66
-	-	0x18C	Reserved	-	-	-
RTCOSCCTRL		0x190	RTC oscillator 32 kHz output control	0x1	0x1	Table 67
-	-	0x194	Reserved	-	-	-
SYSPLLCTRL	R/W	0x198	System PLL control	0x0	0x0	Table 68
SYSPLLSTAT	R	0x19C	System PLL status	0x0	0x0	Table 69
USBPLLCTRL	R/W	0x1A0	USB PLL control	0x0	0x0	Table 70
USBPLLSTAT	R	0x1A4	USB PLL status	0x0	0x0	Table 71
SCTPLLCTRL	R/W	0x1A8	SCT PLL control	0x0	0x0	Table 72

Table 28. Register overview: System configuration (base address 0x4007 4000) *…continued*

Name	Access	Offset	Description	Reset value	Reset value after boot	Reference
SCTPLLSTAT	R	0x1AC	SCT PLL status	0x0	0x0	Table 73
-	-	0x1B0 - 0x1FC	Reserved	-	-	-
-	-	0x200	Reserved	-	-	-
PDAWAKECFG	R/W	0x204	Power-down states for wake-up from deep-sleep	0xFFFF FF00	0xFFFF FE00	Table 74
PDRUNCFG	R/W	0x208	Power configuration register	0xFFFF FF00	0xFFFF FE00	Table 75
-	-	0x20C - 0x214	-	-	-	-
STARTERP0	R/W	0x218	Start logic 0 wake-up enable register	0x0	0x0	Table 76
STARTERP1	R/W	0x21C	Start logic 1 wake-up enable register	0x0	0x0	Table 77
-	-	0x220 - 0x3F0	Reserved	-	-	-
JTAG_IDCODE	R	0x3F4	JTAG ID code register	0x1906 C02B	0x19D6 C02B	Table 78
DEVICE_ID0	R	0x3F8	Part ID register	part dependent	part dependent	Table 79
DEVICE_ID1	R	0x3FC	Boot ROM and die revision register	0x0841 9D6C	0x0841 9D6C	Table 80

3.6.1 System memory remap register

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

If the flash contains valid user code, the boot loader sets the MAP bits to 0x2. In ISP mode, the MAP bits are set to 0x0.

Table 29. System memory remap register (SYSMEMREMAP, address 0x4007 4000) bit description

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

3.6.2 AHB-to-APB bridge 0 write buffering control register

This register controls the buffering of write transactions between the AHB and each APB peripheral individually across the AHB-to-APB bridge 0.

When write buffering is disabled, it takes four cycles on the AHB to finish a write transaction from the AHB side to the APB peripheral as three waitstates are inserted in each transaction.

When the write buffering is enabled, the following two situations can occur:

1. If there is no outstanding transaction on the bus bridge, a write transaction from the AHB master finishes in one cycle (no waitstates are inserted). The write data is queued in the bridge's buffer register, and the bridge continues to handle the write transaction to the APB peripheral while the AHB is ready to take another transaction (read or write).
2. If there is an outstanding transaction from a previous read or write by any of the AHB masters, a write from the AHB is held by inserting waitstates until the write buffer is available for the latest request.

Remark: Buffering is available for write transactions only. Read transactions between the APB and the AHB always take four cycles regardless of the buffer setting.

Table 30. AHB-to-APB bridge 0 write buffering control register (AHBBUFEN0, address 0x4007 400C) bit description

Bit	Symbol	Value	Description	Reset value
0	ADC0_BUF		ADC AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
1	DAC_BUF		DAC AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
2	ACMP_BUF		ACMP AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
4:3	-		Reserved.	0
5	INPUTMUX_BUF		INPUT MUX AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
9:6	-		Reserved.	0
10	RTC_BUF		RTC AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
11	WWDT_BUF		WWDT AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
13:12	-		Reserved.	0

Table 30. AHB-to-APB bridge 0 write buffering control register (AHBBUFEN0, address 0x4007 400C) bit description

Bit	Symbol	Value	Description	Reset value
14	SWM_BUF		SWM AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
15	PMU_BUF		PMU AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
16	UART0_BUF		USART0 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
17	UART1_BUF		USART1 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
18	SPI0_BUF		SPI0 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
19	SPI1_BUF		SPI1 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
20	I2C0_BUF		I2C0 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
21	-		Reserved.	0
22	QEI_BUF		QEI AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
28:23	-		Reserved.	0
29	SYSCON_BUF		SYSCON AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
31:30	-		Reserved.	0

3.6.3 AHB-to-APB bridge 1 write buffering control register

This register controls the buffering of write transactions between the AHB and each APB peripheral individually across the AHB-to-APB bridge 1.

When write buffering is disabled, it takes four cycles on the AHB to finish a write transaction from the AHB side to the APB peripheral as three waitstates are inserted in each transaction.

When the write buffering is enabled, the following two situations can occur:

1. If there is no outstanding transaction on the bus bridge, a write transaction from the AHB master finishes in one cycle (no waitstates are inserted). The write data is queued in the bridge's buffer register, and the bridge continues to handle the write transaction to the APB peripheral while the AHB is ready to take another transaction (read or write).
2. If there is an outstanding transaction from a previous read or write by any of the AHB masters, a write from the AHB is held by inserting waitstates until the write buffer is available for the latest request.

Remark: Buffering is available for write transactions only. Read transactions between the APB and the AHB always take four cycles regardless of the buffer setting.

Table 31. AHB-to-APB bridge 1 write buffering control register (AHBBUFEN1, address 0x4007 4010) bit description

Bit	Symbol	Value	Description	Reset value
0	ADC1_BUF		ADC1 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
7:1	-		Reserved.	0
8	MRT_BUF		MRT AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
9	PINT_BUF		PINT AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
10	GINT0_BUF		GINT0 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
11	GINT1_BUF		GINT1 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
12	-		Reserved.	0
13	RIT_BUF		RIT AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
14	SCTIPU_BUF		SCTIPU AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
15	FMC_BUF		FMC AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
16	UART2_BUF		USART2 AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
27:17	-		Reserved.	0

Table 31. AHB-to-APB bridge 1 write buffering control register (AHBBUFEN1, address 0x4007 4010) bit description

Bit	Symbol	Value	Description	Reset value
28	CCAN_BUF		C_CAN AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
29	-		Reserved.	0
30	IOCON_BUF		IOCON AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	
31	EEPROM_BUF		EEPROM AHB-APB write buffering	0
		0	Disabled.	
		1	Enabled.	

3.6.4 System tick counter calibration register

This register determines the value of the SYST_CALIB register. See Table 295.

Table 32. System tick timer calibration register (SYSTCKCAL, address 0x4007 4014) bit description

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

3.6.5 NMI source selection register

The NMI source selection register selects a peripheral interrupts as source for the NMI interrupt of the ARM Cortex-M3 core. For a list of all peripheral interrupts and their IRQ numbers see Table 2. For a description of the NMI functionality, see Ref. 1.

Remark: When you want to change the interrupt source for the NMI, you must first disable the NMI source by setting bit 31 in this register to 0. Then change the source by updating the IRQN bits and re-enable the NMI source by setting bit 31 to 1.

Table 33. NMI source selection register (NMISRC, address 0x4007 401C) bit description

Bit	Symbol	Description	Reset value
5:0	IRQN	The IRQ number of the interrupt that acts as the Non-Maskable Interrupt (NMI) if bit 31 is 1.	0
30:6	-	Reserved	-
31	NMIEN	Write a 1 to this bit to enable the Non-Maskable Interrupt (NMI) source selected by bits 5:0.	0

Remark: 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.

3.6.6 System reset status register

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

The reset value given in Table 34 applies to the POR reset.

Table 34. System reset status register (SYSRSTSTAT, address 0x4007 4040) bit description

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

3.6.7 Peripheral reset control register 0

The PRESETCTRL0 register allows software to reset specific peripherals. Writing a zero to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a one asserts the reset.

Table 35. Peripheral reset control register 0 (PRESETCTRL0, address 0x4007 4044) bit description

Bit	Symbol	Value	Description	Reset value
6:0	-		Reserved. Only write 0 to these bits.	0
7	FLASH_RST		Flash controller reset control	0
		0	Clear flash reset.	
		1	Assert flash reset.	
8	-		Reserved. Only write 0 to this bit.	-
9	EEPROM_RST		EEPROM controller reset control	0
		0	Clear EEPROM reset.	
		1	Assert EEPROM reset.	
10	-		Reserved. Only write 0 to this bit.	0

Table 35. Peripheral reset control register 0 (PRESETCTRL0, address 0x4007 4044) bit description

Bit	Symbol	Value	Description	Reset value
11	MUX_RST		Input mux reset control	0
		0	Clear pin mux reset.	
		1	Assert pin mux reset.	
12	-		Reserved. Only write 0 to this bit.	0
13	IOCON_RST		IOCON reset control	0
		0	Clear IOCON reset.	
		1	Assert IOCON reset.	
17:14	-		Reserved. Only write 0 to this bit.	0
18	PINT_RST		Pin interrupt (PINT) reset control	0
		0	Clear PINT reset.	
		1	Assert PINT reset.	
19	GINT_RST		Grouped interrupt (GINT) reset control	0
		0	Clear GINT reset.	
		1	Assert GINT reset.	
20	DMA_RST		DMA reset control	0
		0	Clear DMA reset.	
		1	Assert DMA reset.	
21	CRC_RST		CRC generator reset control	0
		0	Clear CRC reset.	
		1	Assert CRC reset.	
26:22	-		Reserved. Only write 0 to these bits.	0
27	ADC0_RST		ADC0 reset control	0
		0	Clear ADC0 reset.	
		1	Assert ADC0 reset.	
28	ADC1_RST		ADC1 reset control	0
		0	Clear ADC1 reset.	
		1	Assert ADC1 reset.	
29	-		Reserved.Only write 0 to this bit.	0
30	ACMP_RST		Analog Comparator (ACMP) reset control for all four 4 comparators in the analog comparator block.	0
		0	Clear ACMP reset.	
		1	Assert ACMP reset.	
31	-	-	Reserved	-

3.6.8 Peripheral reset control register 1

The PRESETCTRL1 register allows software to reset specific peripherals. Writing a zero to any assigned bit in this register clears the reset and allows the specified peripheral to operate. Writing a one asserts the reset.

Table 36. Peripheral reset control register 1 (PRESETCTRL1, address 0x4007 4048) bit description

Bit	Symbol	Value	Description	Reset value
0	MRT_RST		Multi-rate timer (MRT) reset control	0
		0	Clear MRT reset.	
		1	Assert MRT reset.	
1	RIT_RST		Repetitive interrupt timer (RIT) reset control	0
		0	Clear RIT reset.	
		1	Assert RIT reset.	
2	SCT0_RST		State configurable timer 0 (SCT0) reset control	0
		0	Clear SCT0 reset.	
		1	Assert SCT0 reset.	
3	SCT1_RST		State configurable timer 1 (SCT1) reset control	0
		0	Clear SCT1 reset.	
		1	Assert SCT1 reset.	
4	SCT2_RST		State configurable timer 2 (SCT2) reset control	0
		0	Clear SCT2 reset.	
		1	Assert SCT2 reset.	
5	SCT3_RST		State configurable timer 3 (SCT3) reset control	0
		0	Clear SCT3 reset.	
		1	Assert SCT3 reset.	
6	SCTIPU_RST		State configurable timer IPU (SCTIPU) reset control	0
		0	Clear SCTIPU reset.	
		1	Assert SCTIPU reset.	
7	CCAN_RST		CCAN reset control	0
		0	Clear CCAN reset.	
		1	Assert CCAN reset.	
8	-		Reserved.Only write 0 to this bit.	0
9	SPI0_RST		SPI0 reset control	0
		0	Clear SPI0 reset.	
		1	Assert SPI0 reset.	
10	SPI1_RST		SPI1 reset control	0
		0	Clear SPI1 reset.	
		1	Assert SPI1 reset.	
12:11	-		Reserved.Only write 0 to these bits.	0
13	I2C0_RST		I2C0 reset control	0
		0	Clear I2C0 reset.	
		1	Assert I2C0 reset.	
16:14	-		Reserved. Only write 0 to these bits.	0

Table 36. Peripheral reset control register 1 (PRESETCTRL1, address 0x4007 4048) bit description

Bit	Symbol	Value	Description	Reset value
17	UART0_RST		USART0 reset control	0
		0	Clear USART0 reset.	
		1	Assert USART0 reset.	
18	UART1_RST		USART1 reset control	0
		0	Clear USART1 reset.	
		1	Assert USART1 reset.	
19	UART2_RST		USART2 reset control	0
		0	Clear USART2 reset.	
		1	Assert USART2 reset.	
20	-		Reserved. Only write 0 to these bits.	0
21	QEI_RST		QEI reset control	0
		0	Clear QEI reset.	
		1	Assert QEI reset.	
22	-		Reserved. Only write 0 to this bit.	0
23	USB_RST		USB reset control	0
		0	Clear USB reset.	
		1	Assert USB reset.	
27:24	-		Reserved. Only write 0 to these bits.	0
28			Reserved	0
		0	Clear PVT reset.	
		1	Assert PVT reset.	
31:29	-	-	Reserved. Only write 0 to these bits.	-

3.6.9 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 37. POR captured PIO status register 0 (PIOPORCAP0, address 0x4007 404C) bit description

Bit	Symbol	Description	Reset value
31:0	PIOSTAT	State of PIO0_31 through PIO0_0 at power-on reset	Implementation dependent

3.6.10 POR captured PIO status register 1

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

Table 38. POR captured PIO status register 1 (PIOPORCAP1, address 0x4007 4050) bit description

Bit	Symbol	Description	Reset value
31:0	PIOSTAT	State of PIO1_31 through PIO1_0 at power-on reset	Implementation dependent

3.6.11 POR captured PIO status register 2

The PIOPORCAP2 register captures the state of GPIO port 2 at power-on-reset. Each bit represents the reset state of one GPIO pin. This register is a read-only status register.

Table 39. POR captured PIO status register 2 (PIOPORCAP2, address 0x4007 4054) bit description

Bit	Symbol	Description	Reset value
11:0	PIOSTAT	State of PIO2_11 through PIO2_0 at power-on reset	Implementation dependent
31:12	-	Reserved.	-

3.6.12 Main clock source select register A

This register selects one of the internal oscillators, IRC, system oscillator, or watchdog oscillator. The oscillator selected is then one of the inputs to the main clock source select register B (see Table 41), which selects the clock source for the main clock. All clocks to the core, memories, and peripherals are derived from the main clock.

Table 40. Main clock source select register A (MAINCLKSELA, address 0x4007 4080) bit description

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

3.6.13 Main clock source select register B

This register selects the clock source for the main clock. All clocks to the core, memories, and peripherals are derived from the main clock.

One input to this register is the main clock source select register A (see Table 40), which selects one of the three internal oscillators, IRC, system oscillator, or watchdog oscillator.

Table 41. Main clock source select register B (MAINCLKSELB, address 0x4007 4084) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		Clock source for main clock source selector B. Selects the clock source for the main clock.	0
		0x0	MAINCLKSELA. Clock source selected in MAINCLKSELA register.	
		0x1	System PLL input.	
		0x2	System PLL output.	
		0x3	RTC oscillator 32 kHz output.	
31:2	-	-	Reserved	-

3.6.14 USB clock source select register

This register selects the clock source for the USB usb_clk. The clock source can be either the USB PLL output or the main clock, and the clock can be further divided by the USBCLKDIV register (see Table 56) to obtain a 48 MHz clock.

Table 42. USB clock source select (USBCLKSEL, address 0x4007 4088) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		USB clock source.	0x0
		0x0	IRC Oscillator	
		0x1	System oscillator	
		0x2	USB PLL out	
		0x3	Main clock	
31:2	-		Reserved	0x00

3.6.15 ADC asynchronous clock source select register

This register selects an clock source from the IRC or any of the PLL outputs for the 12-bit ADCs that is asynchronous to the system clock. To use this clock, select the asynchronous clock mode in the ADC control register.

Table 43. ADC asynchronous clock source select (ADCASYNCCLKSEL, address 0x4007 408C) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		ADC clock source.	0x0
		0x0	IRC Oscillator	
		0x1	System PLL output	
		0x2	USB PLL output	
		0x3	SCT PLL output	
31:2	-		Reserved	0x00

3.6.16 CLKOUT clock source select register A

This register pre-selects one of the internal oscillators for the clock sources visible on the CLKOUT pin. The selection for the CLKOUT clock source is done in the CLKOUT clock source B register.

Table 44. CLKOUT clock source select register (CLKOUTSELA, address 0x4007 4094) bit description

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

3.6.17 CLKOUT clock source select register B

This register selects the clock source visible on the CLKOUT pin. The internal oscillators are pre-selected in the CLKOUTSELA register (see Table 44).

Table 45. CLKOUT clock source select register (CLKOUTSELB, address 0x4007 4098) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		CLKOUT clock source	0
		0x0	CLKOUTSELA. Clock source selected in the CLKOUTSELA register.	
		0x1	USB PLL output.	
		0x2	SCT PLL output.	
		0x3	RTC 32 kHz output.	
31:2	-	-	Reserved	0

3.6.18 System PLL clock source select register

This register selects the clock source for the system PLL.

Table 46. System PLL clock source select register (SYSPLLCLKSEL, address 0x4007 40A0) bit description

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

3.6.19 USB PLL clock source select register

This register selects the clock source for the dedicated USB PLL.

Table 47. USB PLL clock source select (USBPLLCLKSEL, address 0x4007 40A4) bit description

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

3.6.20 SCT PLL clock source select register

This register selects the clock source for the dedicated SCT PLL.

Table 48. SCT PLL clock source select (SCTPLLCLKSEL, address 0x4007 40A8) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SEL		SCT PLL clock source	0x00
		0x0	IRC	
		0x1	System oscillator	
		0x2	Reserved	
		0x3	Reserved	
31:2	-		Reserved	0x00

3.6.21 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 49. System clock divider register (SYSAHBCLKDIV, address 0x4007 40C0) 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	-

3.6.22 System clock control register 0

The SYSAHBCLKCTRL0 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-M3, the SYSCON block, and the PMU. This clock cannot be disabled.

See Section 1.2.1 for details on the SRAM configuration.

Table 50. System clock control register 0 (SYSAHBCLKCTRL0, address 0x4007 40C4) bit description

Bit	Symbol	Value	Description	Reset value
0	SYS		This bit is read-only and always reads as 1. It configures the always-on clock for the AHB, the APB bridges, the Cortex-M3 core clocks, SYSCON, reset control, SRAM0, and the PMU. Writes to this bit are ignored.	1
1	ROM		Enables clock for ROM.	1
		0	Disable	
		1	Enable	
2	-		Reserved	0
3	SRAM1		Enables clock for SRAM1.	1
		0	Disable	
		1	Enable	
4	SRAM2		Enables clock for SRAM2.	1
		0	Disable	
		1	Enable	
6:5	-		Reserved	0
7	FLASH		Enables clock for flash controller.	1
		0	Disable	
		1	Enable	
8	-		Reserved	1
9	EEPROM		Enables clock for EEPROM controller.	0
		0	Disable	
		1	Enable	
10	-		Reserved	0
11	MUX		Enables clock for input mux.	0
		0	Disable	
		1	Enable	
12	SWM		Enables clock for switch matrix.	0
		0	Disable	
		1	Enable	
13	IOCON		Enables clock for IOCON block.	0
		0	Disable	
		1	Enable	
14	GPIO0		Enables clock for GPIO0 port registers.	0
		0	Disable	
		1	Enable	
15	GPIO1		Enables clock for GPIO1 port registers.	0
		0	Disable	
		1	Enable	

Table 50. System clock control register 0 (SYSAHBCLKCTRL0, address 0x4007 40C4) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
16	GPIO2		Enables clock for GPIO2 port registers.	0
		0	Disable	
		1	Enable	
17	-		Reserved	0
18	PINT		Enables clock for pin interrupt block.	0
		0	Disable	
		1	Enable	
19	GINT		Enables clock for grouped pin interrupt block.	0
		0	Disable	
		1	Enable	
20	DMA		Enables clock for DMA.	0
		0	Disable	
		1	Enable	
21	CRC		Enables clock for CRC.	0
		0	Disable	
		1	Enable	
22	WWDT		Enables clock for WWDT.	0
		0	Disable	
		1	Enable	
23	RTC		Enables clock for RTC.	0
		0	Disable	
		1	Enable	
26:24	-		Reserved	0
27	ADC0		Enables clock for ADC0 register interface.	0
		0	Disable	
		1	Enable	
28	ADC1		Enables clock for ADC1 register interface.	0
		0	Disable	
		1	Enable	
29	DAC		Enables clock for DAC.	0
		0	Disable	
		1	Enable	
30	ACMP		Enables clock to analog comparator block. This is the clock to the register interface for all 4 comparators.	0
		0	Disable	
		1	Enable	
31	-		Reserved	0

3.6.23 System clock control register 1

The SYSAHBCLKCTRL register enables the clocks to individual peripheral blocks.

Table 51. System clock control register 1 (SYSAHBCLKCTRL1, address 0x4007 40C8) bit description

Bit	Symbol	Value	Description	Reset value
0	MRT		Enables clock for multi-rate timer.	0
		0	Disable	
		1	Enable	
1	RIT		Enables clock for repetitive interrupt timer.	0
		0	Disable	
		1	Enable	
2	SCT0		Enables clock for SCT0.	0
		0	Disable	
		1	Enable	
3	SCT1		Enables clock for SCT1.	0
		0	Disable	
		1	Enable	
4	SCT2		Enables clock for SCT2.	0
		0	Disable	
		1	Enable	
5	SCT3		Enables clock for SCT3.	0
		0	Disable	
		1	Enable	
6	SCTIPU		Enables clock for SCTIPU.	0
		0	Disable	
		1	Enable	
7	CCAN		Enables clock for CCAN.	0
		0	Disable	
		1	Enable	
8	-		Reserved	0
9	SPI0		Enables clock for SPI0.	0
		0	Disable	
		1	Enable	
10	SPI1		Enables clock for SPI1.	
		0	Disable	
		1	Enable	
12:11	-		Reserved	0
13	I2C0		Enables clock for I2C0.	0
		0	Disable	
		1	Enable	
16:14	-		Reserved	0
17	UART0		Enables clock for USART0.	0
		0	Disable	
		1	Enable	

Table 51. System clock control register 1 (SYSAHBCLKCTRL1, address 0x4007 40C8) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
18	UART1		Enables clock for USART1.	0
		0	Disable	
		1	Enable	
19	UART2		Enables clock for USART2.	0
		0	Disable	
		1	Enable	
20	-		Reserved	0
21	QEI		Enables clock for QEI.	0
		0	Disable	
		1	Enable	
22	-		Reserved	0
23	USB		Enables clock for USB register interface.	0
		0	Disable	
		1	Enable	
27:24	-	-	Reserved	-
28			Reserved	0
31:29	-	-	Reserved	-

3.6.24 SYSTICK clock divider register

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

Table 52. SYSTICK clock divider (SYSTICKCLKDIV, address 0x4007 40CC) bit description

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

3.6.25 USART clock divider register

This register configures the clock for the fractional baud rate generator and all USARTs. The UART clock can be disabled by setting the DIV field to zero (this is the default setting).

Table 53. USART clock divider register (UARTCLKDIV, address 0x4007 40D0) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	USART fractional baud rate generator clock divider values. 0: Clock disabled. 1: Divide by 1. to 255: Divide by 255.	0
31:8	-	Reserved	-

3.6.26 IOCON glitch filter clock divider register

This register configures the peripheral input clock (IOCONFILTR_PCLK) to the IOCON programmable glitch filter. The clock can be shut down by setting the DIV bits to 0x0.

Table 54. IOCON glitch filter clock divider register (IOCONCLKDIV, address 0x4007 40D4) bit description

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

3.6.27 ARM trace clock divider register

This register configures the ARM trace clock. The ARM trace clock can be shut down by setting the DIV field to zero.

Table 55. ARM trace clock divider (TRACECLKDIV, address 0x4007 40D8) bit description

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

3.6.28 USB clock source divider register

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

Table 56. USB clock source divider (USBCLKDIV, address 0x4007 40EC) bit description

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

3.6.29 ADCASYNCCLKDIV clock source divider register

This register divides the asynchronous clock to the ADCs. The clock can be shut down by setting the DIV bits to 0x0.

Table 57. ADC asynchronous clock source divider (ADCASYNCCLKDIV, address 0x4007 40F0) bit description

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

3.6.30 CLKOUT clock divider register

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

Table 58. CLKOUT clock divider register (CLKOUTDIV, address 0x4007 40F8) 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	-

3.6.31 Frequency measure function control register

This register starts the frequency measurement function and stores the result in the CAPVAL field. The target frequency can be calculated as follows with the frequencies given in MHz:

$F_{target} = (CAPVAL - 2) \times F_{reference}/2^{14}$

Select the target and reference frequencies using the

Table 59. Frequency measure function control register (FREQMECTRL, address 0x4007 4120) bit description

Bit	Symbol	Description	Reset value
13:0	CAPVAL	Stores the capture result which is used to calculate the frequency of the target clock. This field is read-only.	0
30:14	-	Reserved.	-
31	PROG	Set this bit to one to initiate a frequency measurement cycle. Hardware clears this bit when the measurement cycle has completed and there is valid capture data in the CAPVAL field (bits 13:0).	0

3.6.32 Flash configuration 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.

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

Remark: Improper setting of this register may result in incorrect operation of the flash memory. Do not change the flash access time when using the power API in efficiency, low-current, or performance modes.

Table 60. Flash configuration register (FLASHCFG, address 0x4007 4124) bit description

Bit	Symbol	Value	Description	Reset value
11:0	-		Reserved. **Do not change the value of these bits. Bits 11:0 must be written back exactly as read**.	0x1A
13:12	FLASHTIM		Flash memory access time. FLASHTIM +1 is equal to the number of system clocks used for flash access.	0x2
		0x0	1 clock cycle. 1 system clock flash access time (for system clock frequencies of up to 25 MHz).	
		0x1	2 clock cycles. 2 system clocks flash access time (for system clock frequencies of up to 55 MHz).	
		0x2	3 clock cycles. 3 system clocks flash access time (for system clock frequencies of up to 72 MHz).	
		0x3	Reserved.	
31:14	-	-	Reserved. **Do not change the value of these bits. Bits 31:14 must be written back exactly as read**.	-

3.6.33 USART fractional baud rate generator register

All USART peripherals share a common clock U_PCLK, which can be adjusted by a fractional divider:

U_PCLK = UARTCLKDIV/(1 + MULT/DIV).

This register sets the MULT and DIV values. UARTCLKDIV is the USART clock configured in the UARTCLKDIV register.

Remark: To use of the fractional baud rate generator, you must write 0xFF to the DIV value to yield a denominator value of 256. All other values are not supported.

See also:

Section 24.3.1 “Configure the USART clock and baud rate”

Section 24.7.1 “Clocking and baud rates”

Table 61. USART fractional baud rate generator register (FRGCTRL, address 0x4007 4128) bit description

Bit	Symbol	Description	Reset value
7:0	DIV	Denominator of the fractional divider. DIV is equal to the programmed value +1. Always set to 0xFF to use with the fractional baud rate generator.	0xFF
15:8	MULT	Numerator of the fractional divider. MULT is equal to the programmed value.	0
31:16	-	Reserved	-

3.6.34 USB clock control register

This register controls the use of the USB need_clock signal and the polarity of the need_clock signal for triggering the USB wake-up interrupt. For details of how to use the USB need_clock signal for waking up the part from Deep-sleep or Power-down modes, see Section 23.7.6.

Table 62. USB clock control (USBCLKCTRL, address 0x4007 412C) bit description

Bit	Symbol	Value	Description	Reset value
0	AP_CLK		USB need_clock signal control	0x0
		0	Under hardware control.	
		1	Forced HIGH.	
1	POL_CLK		USB need_clock polarity for triggering the USB wake-up interrupt	0x0
		0	Falling edge of the USB need_clock triggers the USB wake-up (default).	
		1	Rising edge of the USB need_clock triggers the USB wake-up.	
31:2	-		Reserved. Do not write one to reserved bits.	0x00

3.6.35 USB clock status register

This register is read-only and returns the status of the USB need_clock signal. For details of how to use the USB need_clock signal for waking up the part from Deep-sleep or Power-down modes, see Section 23.7.6.

Table 63. USB clock status (USBCLKST, address 0x4007 4130) bit description

Bit	Symbol	Value	Description	Reset value
0	NEED_CLKST		USB need_clock signal status	0x1
		0	LOW	
		1	HIGH	
31:1	-		Reserved	0x00

3.6.36 BOD control register

The BOD control register selects two separate threshold values for sending a BOD interrupt to the NVIC and for forced reset. Reset and interrupt threshold values listed in Table 64 are typical values.

Both the BOD interrupt and the BOD reset, depending on the value of bit BODRSTENA in this register, can wake-up the chip from Sleep, Deep-sleep, and Power-down modes. See Table 517 “power_mode_configure routine”.

The BOD levels are defined in the LPC15xx data sheet.

Table 64. BOD control register (BODCTRL, address 0x4007 4180) bit description

Bit	Symbol	Value	Description	Reset value
1:0	BODRSTLEV		BOD reset level	0
		0x0	Reserved.	
		0x1	Reserved.	
		0x2	Level 2	
		0x3	Level 3	
3:2	BODINTVAL		BOD interrupt level	0
		0x0	Reserved.	
		0x1	Reserved.	
		0x2	Level 2:	
		0x3	Level 3	
4	BODRSTENA		BOD reset enable	0
		0	Disable reset function.	
		1	Enable reset function.	
31:5	-	-	Reserved	0x00

3.6.37 IRC 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 65. IRC control register (IRCCTRL, address 0x4007 4184) bit description

Bit	Symbol	Description	Reset value
7:0	TRIM	Trim value	0x80, then flash will reprogram
31:8	-	Reserved	0x00

3.6.38 System oscillator control register

This register configures the frequency range for the system oscillator. The system oscillator itself is powered on or off in the PDRUNCFG register. See Table 75.

Table 66. System oscillator control register (SYSOSCCTRL, address 0x4007 4188) bit description

Bit	Symbol	Value	Description	Reset value
0	BYPASS		Bypass system oscillator	0x0
		0	Disabled. Oscillator is not bypassed.	
		1	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 system oscillator.	0x0
		0	Low frequency. 1 MHz - 20 MHz frequency range.	
		1	High frequency. 15 MHz - 25 MHz frequency range	
31:2	-	-	Reserved	0x00

3.6.39 RTC oscillator control register

This register enables the 32 kHz output of the RTC oscillator. This clock can be used to create the main clock when the PLL **input** is selected as the clock source to the main clock. Do not use the system PLL with 32 kHz clock.

Table 67. RTC oscillator control register (RTCOSCCTRL, address 0x4007 4190) bit description

Bit	Symbol	Value	Description	Reset value
0	EN		RTC 32 kHz clock enable.	1
		0	Disabled. RTC clock off.	
		1	Enabled. RTC clock on.	
31:1	-	-	Reserved	0x00

3.6.40 System PLL control register

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

Remark: The divider values for P and M must be selected so that the PLL output clock frequency FCLKOUT is lower than 100 MHz.

Remark: This PLL supports a 6-bit feedback divider MSEL.

Table 68. System PLL control register (SYSPLLCTRL, address 0x4007 4198) bit description

Bit	Symbol	Value	Description	Reset value
5:0	MSEL		Feedback divider value. The division value M is the programmed MSEL value + 1. 00000: Division ratio M = 1 to 111111: Division ratio M = 64	0
7:6	PSEL		Post divider ratio P. The division ratio is 2 × P.	0
		0x0	P = 1	
		0x1	P = 2	
		0x2	P = 4	
		0x3	P = 8	
31:8	-	-	Reserved. Do not write ones to reserved bits.	-

3.6.41 System PLL status register

This register is a Read-only register and supplies the PLL lock status.

Table 69. System PLL status register (SYSPLLSTAT, address 0x4007 419C) bit description

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

3.6.42 USB PLL control register

The USB PLL is identical to the system PLL and is used to provide a dedicated clock to the USB block.

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

Remark: This PLL supports a 6-bit feedback divider MSEL.

Table 70. USB PLL control (USBPLLCTRL, address 0x4007 41A0) bit description

Bit	Symbol	Value	Description	Reset value
5:0	MSEL		Feedback divider value. The division value M is the programmed MSEL value + 1. 00000: Division ratio M = 1 to 111111: Division ratio M = 64.	0x000

Table 70. USB PLL control (USBPLLCTRL, address 0x4007 41A0) bit description

Bit	Symbol	Value	Description	Reset value
7:6	PSEL		Post divider ratio P. The division ratio is 2 x P.	0x00
		0x0	P = 1	
		0x1	P = 2	
		0x2	P = 4	
		0x3	P = 8	
31:8	-		Reserved. Do not write ones to reserved bits.	0x00

3.6.43 USB PLL status register

This register is a Read-only register and supplies the PLL lock status.

Table 71. USB PLL status (USBPLLSTAT, address 0x4007 41A4) bit description

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

3.6.44 SCT PLL control register

The SCT PLL is identical to the system PLL and is used to provide a dedicated clock to the SCTs.

Remark: This PLL supports a 6-bit feedback divider MSEL.

Table 72. SCT PLL control (SCTPLLCTRL, address 0x4007 41A8) bit description

Bit	Symbol	Value	Description	Reset value
5:0	MSEL		Feedback divider value. The division value M is the programmed MSEL value + 1. 00000: Division ratio M = 1 to 111111: Division ratio M = 64.	0x000
7:6	PSEL		Post divider ratio P. The division ratio is 2 x P.	0x00
		0x0	P = 1	
		0x1	P = 2	
		0x2	P = 4	
		0x3	P = 8	
31:8	-		Reserved. Do not write ones to reserved bits.	0x00

3.6.45 SCT PLL status register

This register is a Read-only register and supplies the PLL lock status.

Table 73. SCT PLL status (SCTPLLSTAT, address 0x4007 41AC) bit description

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

3.6.46 Wake-up configuration register

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

Table 74. Wake-up configuration register (PDAWAKECFG, address 0x4007 4204) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Always write these bits as 0s.	0
3	IRCOUT_PD		IRC oscillator output wake-up configuration	0
		0	Powered	
		1	Powered down	
4	IRC		IRC oscillator wake-up configuration	0
		0	Powered	
		1	Powered down	
5	FLASH		Flash memory wake-up configuration	0
		0	Powered	
		1	Powered down	
6	EEPROM		EEPROM wake-up configuration	0
		0	Powered	
		1	Powered down	
7	-		Reserved	0
8	BOD_PD		BOD wake-up configuration	0
		0	Powered	
		1	Powered down	
9	USBPHY_PD		USB PHY wake-up configuration	1
		0	Powered	
		1	Powered down	
10	ADC0_PD		ADC0 wake-up configuration	1
		0	Powered	
		1	Powered down	
11	ADC1_PD		ADC1 wake-up configuration	1
		0	Powered	
		1	Powered down	
12	DAC_PD		DAC wake-up configuration	1
		0	Powered	
		1	Powered down	

Table 74. Wake-up configuration register (PDAWAKECFG, address 0x4007 4204) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
13	ACMP0_PD		Analog comparator 0 wake-up configuration	1
		0	Powered	
		1	Powered down	
14	ACMP1_PD		Analog comparator 1 wake-up configuration	1
		0	Powered	
		1	Powered down	
15	ACMP2_PD		Analog comparator 2 wake-up configuration	1
		0	Powered	
		1	Powered down	
16	ACMP3_PD		Analog comparator 3 wake-up configuration	1
		0	Powered	
		1	Powered down	
17	IREF_PD		Internal voltage reference wake-up configuration	1
		0	Powered	
		1	Powered down	
18	TS_PD		Temperature sensor wake-up configuration	1
		0	Powered	
		1	Powered down	
19	VDDADIV_PD		VDDA divider wake-up configuration. This is the divider for the VDDA/2 input to the ADCs.	1
		0	Powered	
		1	Powered down	
20	WDTOSC_PD		Watchdog oscillator wake-up configuration.	1
		0	Powered	
		1	Powered down	
21	SYSOSC_PD		System oscillator wake-up configuration	1
		0	Powered	
		1	Powered down	
22	SYSPLL_PD		System PLL wake-up configuration	1
		0	Powered	
		1	Powered down	
23	USBPLL_PD		USB PLL wake-up configuration	1
		0	Powered	
		1	Powered down	
24	SCTPLL_PD		USB PLL wake-up configuration	1
		0	Powered	
		1	Powered down	
31:25	-	-	Reserved	0

3.6.47 Power configuration register

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

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

The system oscillator requires typically 500 µs to start up after the SYSOSC_PD bit has been changed from 1 to 0. There is no hardware flag to monitor the state of the system oscillator. Therefore, add a software delay of about 500 µs before using the system oscillator after power-up.

Table 75. Power configuration register (PDRUNCFG, address 0x4007 4208) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Always write these bits as 0s.	0
3	IRCOUT_PD		IRC oscillator output	0
		0	Powered	
		1	Powered down	
4	IRC		IRC oscillator	0
		0	Powered	
		1	Powered down	
5	FLASH		Flash memory	0
		0	Powered	
		1	Powered down	
6	EEPROM		EEPROM	0
		0	Powered	
		1	Powered down	
7	-		Reserved	0
8	BOD_PD		BOD power-down	0
		0	Powered	
		1	Powered down	
9	USBPHY_PD		USB PHY power-down	1
		0	Powered	
		1	Powered down	
10	ADC0_PD		ADC0 power-down	1
		0	Powered	
		1	Powered down	
11	ADC1_PD		ADC1 power-down	1
		0	Powered	
		1	Powered down	
12	DAC_PD		DAC power-down	1
		0	Powered	
		1	Powered down	

Table 75. Power configuration register (PDRUNCFG, address 0x4007 4208) bit description

Bit	Symbol	Value	Description	Reset value
13	ACMP0_PD		Analog comparator 0 power-down	1
		0	Powered	
		1	Powered down	
14	ACMP1_PD		Analog comparator 1 power-down	1
		0	Powered	
		1	Powered down	
15	ACMP2_PD		Analog comparator 2 power-down	1
		0	Powered	
		1	Powered down	
16	ACMP3_PD		Analog comparator 3 power-down	1
		0	Powered	
		1	Powered down	
17	IREF_PD		Internal voltage reference power-down	1
		0	Powered	
		1	Powered down	
18	TS_PD		Temperature sensor power-down	1
		0	Powered	
		1	Powered down	
19	VDDADIV_PD		VDDA divider power-down. This is the divider for the VDDA/2 input to the ADCs.	1
		0	Powered	
		1	Powered down	
20	WDTOSC_PD		Watchdog oscillator power-down.	1
		0	Powered	
		1	Powered down	
21	SYSOSC_PD		System oscillator power-down.After power-up, add a software delay of approximately 500 µs before using.	1
		0	Powered	
		1	Powered down	
22	SYSPLL_PD		System PLL power-down	1
		0	Powered	
		1	Powered down	
23	USBPLL_PD		USB PLL power-down	1
		0	Powered	
		1	Powered down	
24	SCTPLL_PD		USB PLL power-down	1
		0	Powered	
		1	Powered down	
31:25	-	-	Reserved	0

3.6.48 Start logic 0 wake-up enable register

The STARTERP0 register enables an interrupt for wake-up from deep-sleep mode and power-down modes. For details see Section 4.7.4 and Section 4.7.5.

Remark: Also enable the corresponding interrupts in the NVIC. See Table 2 “Connection of interrupt sources to the NVIC”.

Table 76. Start logic 0 wake-up enable register 0 (STARTERP0, address 0x4007 4218) bit description

Bit	Symbol	Value	Description	Reset value
0	WWDT		WWDT interrupt wake-up.	0
		0	Disabled	
		1	Enabled	
1	BOD		BOD interrupt wake-up.	0
		0	Disabled	
		1	Enabled	
4:2	-		Reserved.	0
5	GINT0		Group interrupt 0 wake-up.	0
		0	Disabled	
		1	Enabled	
6	GINT1		Group interrupt 1 wake-up.	0
		0	Disabled	
		1	Enabled	
7	PINT0		GPIO pin interrupt 0 wake-up	0
		0	Disabled	
		1	Enabled	
8	PINT1		GPIO pin interrupt 1 wake-up	0
		0	Disabled	
		1	Enabled	
9	PINT2		GPIO pin interrupt 2 wake-up	0
		0	Disabled	
		1	Enabled	
10	PINT3		GPIO pin interrupt 3 wake-up	0
		0	Disabled	
		1	Enabled	
11	PINT4		GPIO pin interrupt 4 wake-up	0
		0	Disabled	
		1	Enabled	
12	PINT5		GPIO pin interrupt 5 wake-up	0
		0	Disabled	
		1	Enabled	
13	PINT6		GPIO pin interrupt 6 wake-up	0
		0	Disabled	
		1	Enabled	

Table 76. Start logic 0 wake-up enable register 0 (STARTERP0, address 0x4007 4218) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
14	PINT7		GPIO pin interrupt 7 wake-up	0
		0	Disabled	
		1	Enabled	
20:15	-		Reserved.	0
21	USART0		USART0 interrupt wake-up. Configure USART in synchronous slave mode or in 32 kHz mode..	0
		0	Disabled	
		1	Enabled	
22	USART1		USART1 interrupt wake-up. Configure USART in synchronous slave mode or in 32 kHz mode...	0
		0	Disabled	
		1	Enabled	
23	USART2		USART2 interrupt wake-up. Configure USART in synchronous slave mode or in 32 kHz mode...	0
		0	Disabled	
		1	Enabled	
24	I2C		I2C interrupt wake-up.	0
		0	Disabled	
		1	Enabled	
25	SPI0		SPI0 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
26	SPI1		SPI1 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
29:27	-		Reserved	0
30	USB_WAKEUP		USB need_clock signal wake-up	0
		0	Disabled	
		1	Enabled	
31	-		Reserved	-

3.6.49 Start logic 1 wake-up enable register

This register selects which interrupts wake up the part from deep-sleep and power-down modes. For details see Section 4.7.4 and Section 4.7.5.

Remark: Also enable the corresponding interrupts in the NVIC. See Table 2 “Connection of interrupt sources to the NVIC”.

Table 77. Start logic 1 wake-up enable register (STARTERP1, address 0x4007 421C) bit description

Bit	Symbol	Value	Description	Reset value
7:0	-		Reserved	-
8	ACMP0		Analog comparator 0 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
9	ACMP1		Analog comparator 1 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
10	ACMP2		Analog comparator 2 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
11	ACMP3		Analog comparator 3 interrupt wake-up	0
		0	Disabled	
		1	Enabled	
12	-		Reserved	-
13	RTCALARM		RTC alarm interrupt wake-up	0
		0	Disabled	
		1	Enabled	
14	RTCWAKE		RTC wake-up interrupt wake-up	0
		0	Disabled	
		1	Enabled	
31:15	-		Reserved.	-

3.6.50 JTAG ID code register

This register contains the JTAG ID code.

Table 78. JTAG ID code register (JTAGIDCODE, address 0x4007 43F4) bit description

Bit	Symbol	Description	Reset value
31:0	JTAGID	JTAG ID code.	0x19D6 C02B

3.6.51 Device ID0 register

This register contains the part ID. The part ID can also be obtained using the ISP or IAP ReadPartID commands. See Table 490 and Table 505.

LPC1549 = 0x00001549

LPC1548 = 0x00001548

LPC1547 = 0x00001547

LPC1519 = 0x00001519

LPC1518 = 0x00001518

LPC1517 = 0x00001517

Table 79. Device ID0 register (DEVICE_ID0, address 0x4007 43F8) bit description

Bit	Symbol	Description	Reset value
31:0	PARTID	Part ID	part dependent

3.6.52 Device ID1 register

This register contains the boot ROM and die revisions.

Table 80. Device ID1 register (DEVICE_ID1, address 0x4007 43FC) bit description

Bit	Symbol	Description	Reset value
31:0	REVID	Revision.	0x0841 9D6C

3.7 Functional description

3.7.1 Reset

Reset has the following sources:

- The $\overline{\text{RESET}}$ pin. The $\overline{\text{RESET}}$ pin is a Schmitt trigger input pin.
- Watchdog reset.
- Power-On Reset (POR).
- Brown Out Detect (BOD).
- ARM software reset.

Assertion of the POR or the BOD reset, once the operating voltage attains a usable level, starts the IRC. After the IRC-start-up time (maximum of 6 μs on power-up), the IRC provides a stable clock output. The reset remains asserted until the external Reset is released, the oscillator is running, and the flash controller has completed its initialization.

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

1. The IRC is enabled or starts up if not running.
2. The flash wake-up timer starts. This takes approximately 100 μs. Then the flash initialization sequence is started, which takes about 250 cycles.
3. The boot code in the ROM starts. The boot code performs the boot tasks and may jump to the flash.

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

Remark: The switch matrix is only reset after a POR or BOD reset.

3.7.2 Start-up behavior

See Figure 4 for the start-up timing after reset. The IRC is the default clock at Reset and provides a clean system clock shortly after the supply voltage reaches the threshold value of 1.8 V.

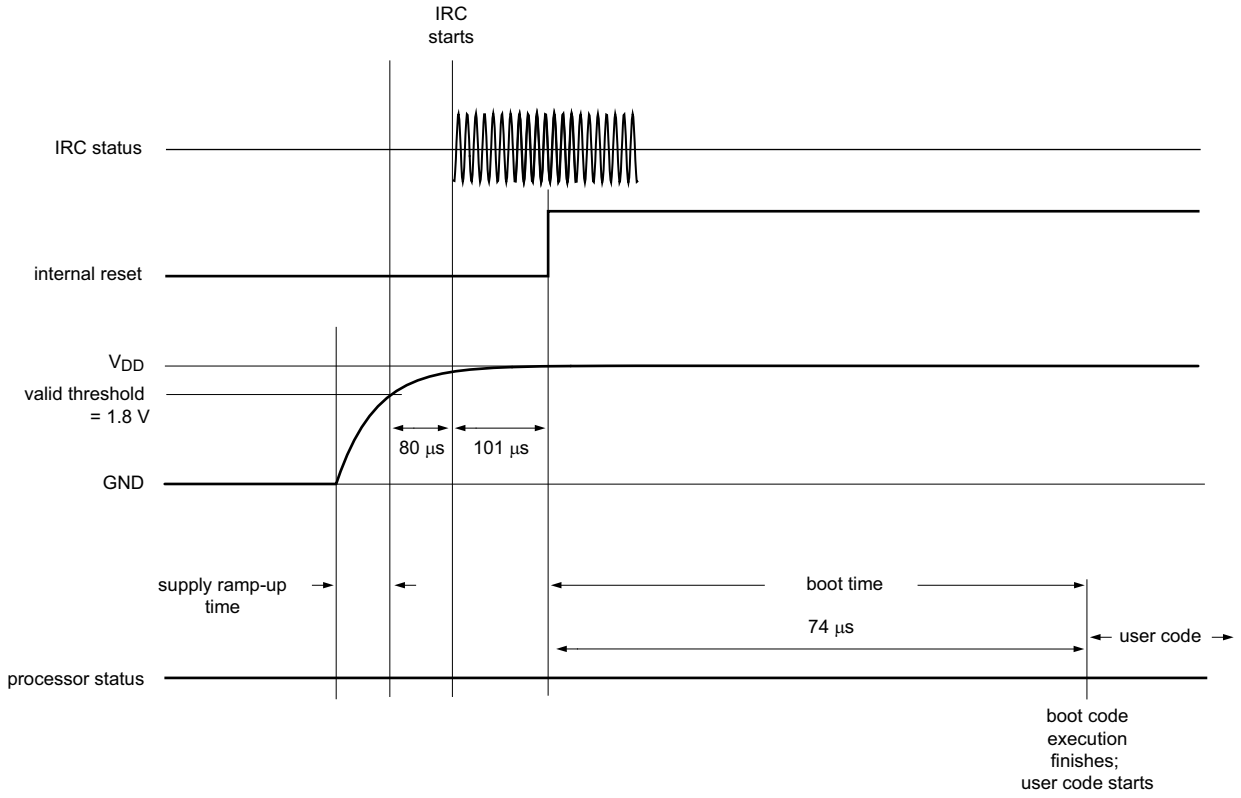


Fig 4. Start-up timing

3.7.3 Brown-out detection

The part includes up to four levels for monitoring the voltage on the V_{DD} pin. If this voltage falls below one of the selected levels, the BOD asserts an interrupt signal to the NVIC or issues a reset, depending on the value of the BODRSTENA bit in the BOD control register (Table 64).

The interrupt signal can be enabled for interrupt in the Interrupt Enable Register in the NVIC (see Table 3) in order to cause a CPU interrupt; if not, software can monitor the signal by reading a dedicated status register.

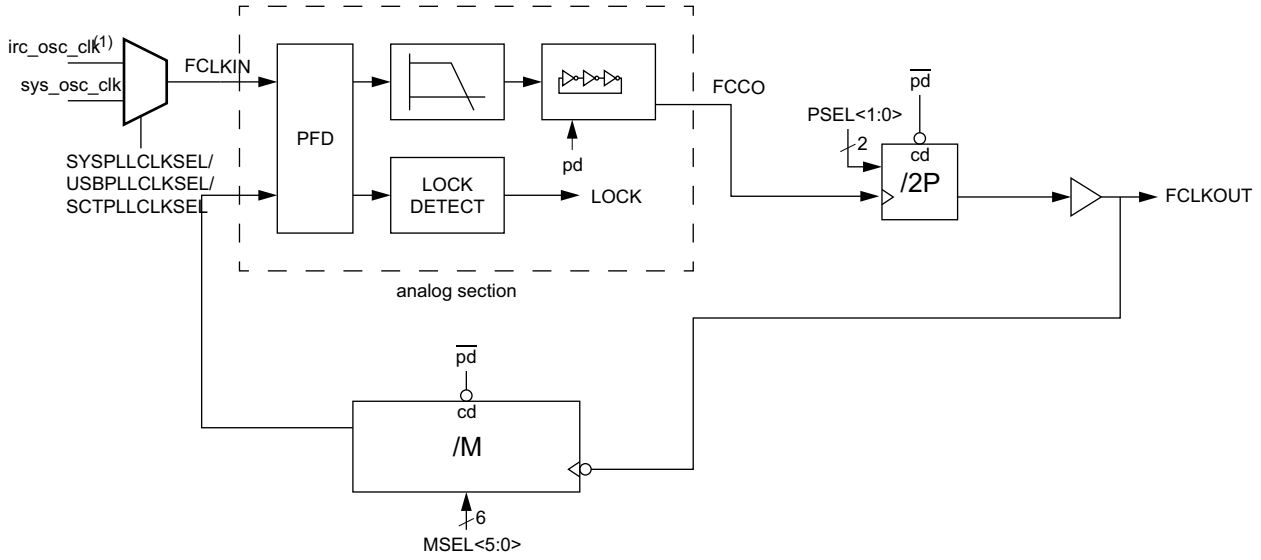
If the BOD interrupt is enabled in the STARTERP register and in the NVIC, the BOD interrupt can wake up the chip from Deep-sleep and power-down mode.

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

3.7.4 PLL functional description

The LPC15xx uses the PLL to create the clocks for the core and peripherals.

All three PLLs (system, USB, and SCT) are identical.



(1) Do not use the IRC as a clock source for the USB full-speed operation.

Fig 5. PLL block diagram

The block diagram of this PLL is shown in Figure 5. The input frequency range is 10 MHz to 25 MHz. The input clock is fed directly to the Phase-Frequency Detector (PFD). This block compares the phase and frequency of its inputs, and generates a control signal when phase and/ or frequency do not match. The loop filter filters these control signals and drives the current controlled oscillator (CCO), which generates the main clock and optionally two additional phases. The CCO frequency range is 156 MHz to 320 MHz. These clocks are either divided by 2×P by the programmable post divider to create the output clocks, or are sent directly to the outputs. The main output clock is then divided by M by the programmable feedback divider to generate the feedback clock. The output signal of the phase-frequency detector is also monitored by the lock detector, to signal when the PLL has locked on to the input clock.

Remark: The divider values for P and M must be selected so that the PLL output clock frequency FCLKOUT is lower than 100 MHz because the main clock is limited to a maximum frequency of 100 MHz

3.7.4.1 Lock detector

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

3.7.4.2 Power-down control

To reduce the power consumption when the PLL clock is not needed, a PLL Power-down mode has been incorporated. This mode is enabled by setting the SYSPLL_PD bit to one in the Power configuration register (Table 75). In this mode, the internal current reference will be turned off, the oscillator and the phase-frequency detector will be stopped and the dividers will enter a reset state. While in PLL Power-down mode, the lock output will be low to indicate that the PLL is not in lock. When the PLL Power-down mode is terminated by setting the SYSPLL_PD bit to zero, the PLL will resume its normal operation and will make the lock signal high once it has regained lock on the input clock.

3.7.4.3 Divider ratio programming

3.7.4.3.1 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 68. This guarantees an output clock with a 50% duty cycle.

3.7.4.3.2 Feedback divider

The feedback divider's division ratio is controlled by the MSEL bits. The division ratio between the PLL's output clock and the input clock is the decimal value on MSEL bits plus one, as specified in Table 68.

3.7.4.3.3 Changing the divider values

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

3.7.4.4 Frequency selection

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

Table 81. PLL frequency parameters

Parameter	System PLL
FCLKIN	Frequency of sys_pllclkin (input clock to the system PLL) from the SYSPLLCLKSEL multiplexer (see Section 3.6.18).
FCCO	Frequency of the Current Controlled Oscillator (CCO); 156 to 320 MHz.
FCLKOUT	Frequency of sys_pllclkout. This is the PLL output frequency and must be < 100 MHz.
P	System PLL post divider ratio; PSEL bits in SYSPLLCTRL (see Section 3.6.40).
M	System PLL feedback divider register; MSEL bits in SYSPLLCTRL (see Section 3.6.40).

3.7.4.4.1 Normal mode

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

(1)

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

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

1. Specify the input clock frequency Fclkin.
2. Calculate M to obtain the desired output frequency Fclkout with M = F_{clkout} / F_{clkin}.
3. Find a value so that FCCO = 2 × P × F_{clkout}.
4. Verify that all frequencies and divider values conform to the limits specified in Table 68.

Remark: The divider values for P and M must be selected so that the PLL output clock frequency FCLKOUT is lower than 100 MHz.

Table 82 shows how to configure the PLL for a 12 MHz crystal oscillator using the SYSPLLCTRL register (Table 68). The main clock is equivalent to the system clock if the system clock divider SYSAHBCLKDIV is set to one (see Table 49).

Table 82. PLL configuration examples

PLL input clock sys_pllclkin (Fclkin)	Main clock (Fclkout)	MSEL bits Table 68	M divider value	PSEL bits Table 68	P divider value	FCCO frequency	SYSAHBCLKDIV	System clock
12 MHz	72 MHz	5	6	1	2	288 MHz	1	72 MHz
12 MHz	60 MHz	4	5	1	2	240 MHz	2	30 MHz
12 MHz	24 MHz	1	2	2	4	192 MHz	1	24 MHz

3.7.4.4.2 PLL Power-down mode

In this mode, the internal current reference will be turned off, the oscillator and the phase-frequency detector will be stopped and the dividers will enter a reset state. While in PLL Power-down mode, the lock output will be low, to indicate that the PLL is not in lock. When the PLL Power-down mode is terminated by SYSPLL_PD bit to zero in the Power-down configuration register (Table 75), the PLL will resume its normal operation and will make the lock signal high once it has regained lock on the input clock.

3.7.5 Frequency measure function

The Frequency Measure circuit is based on two 14-bit counters, one clocked by the reference clock and one by the target clock. Synchronization between the clocks is performed at the start and end of each count sequence.

A measurement cycle is initiated by software setting a control/status bit in the FREQMECTRL register (Table 59). The software can then poll this same measure-in-progress bit which will be cleared by hardware when the measurement operation is completed.

The measurement cycle terminates when the reference counter rolls-over. At that point the state of the target counter is loaded into a capture field in the FREQMEAS register, and the measure-in-progress bit is cleared. Software can read this capture value and apply to it a specific calculation which will return the precise frequency of the target clock in MHz.

3.7.5.1 Accuracy

The frequency measurement function can measure the frequency of any on-chip (or off-chip) clock (referred to as the target clock) to a high degree of accuracy using another on-chip clock of known frequency as a reference.

The following constraints apply:

- The frequency of the reference clock must be (somewhat) greater that the frequency of the target clock.
- The system clock used to access the frequency measure function register must also be greater than the frequency of the target clock.

The frequency measurement function circuit is able to measure the target frequency with an error of less than 0.1%, provided the reference frequency is precisely known.

Uncertainty in the reference clock (for example the +/- 1% accuracy of the IRC) will add to the measurement error of the target clock. In general, though, this additional error is less than the uncertainty of the reference clock.

There can also be a modest loss of accuracy if the reference frequency exceeds the target frequency by a very large margin (25x or more). Accuracy is not a simple function of the magnitude of the frequency difference, however. Nearly identical frequency combinations, still with a spread of about 43x, result in errors of less than 0.05 %.

If the target and reference clocks are different by more than a factor of approximately 500, then the accuracy decreases to +/- 4 %.

4.1 How to read this chapter

The LPC15xx provides an on-chip API in the boot ROM to optimize power consumption in active and sleep modes and configure the part for deep-sleep and power-down modes. See Chapter 35

The PMU is identical for all parts.

4.2 Features

- Control of the reduced power modes
- Four general purpose backup registers to retain data in Deep power-down mode

4.3 Basic configuration

The PMU is always on as long as the V_{DD} or V_{BAT} supply voltages are present.

To turn analog components on or off in active and sleep modes, use the PDRUNCFG register (see Table 75). In deep-sleep and power-down modes, the power profile API controls which analog peripherals remain powered up (see Table 517). There is no register implemented to turn analog peripherals on or off for deep-sleep mode or power-down mode.

4.4 Pin description

In Deep power-down only the WAKEUP pin (PIO0_17) is functional if V_{DD} is present. To conserve additional power, the WAKEUP pin function can be disabled in the GPREG4 register.

Remark: When entering Deep power-down mode, an external pull-up resistor is required on the WAKEUP pin to hold it HIGH until LOW going pulse wakes up the part. In addition, pull the RESET pin HIGH to prevent it from floating while in Deep power-down mode.

4.5 General description

Power to the part is supplied via two power domains. The main power domain is powered by VDD and supplies power to the core, peripheral, memories, inputs and outputs via an on-chip regulator.

A second, always-on power domain can either be powered be VDD if present or by VBAT and supplies power to the RTC and five general-purpose back-up registers which can be used to save data in deep power-down mode or when VDD is shut down.

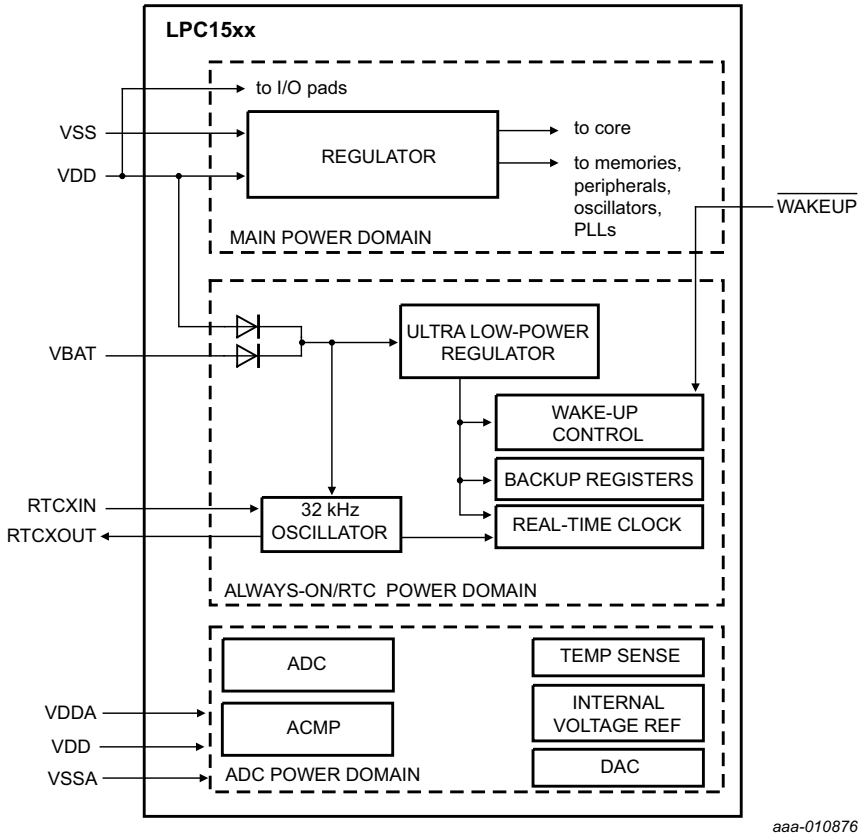


Fig 6. Power domains

The power use is controlled by the PMU, by the SYSCON block, and the ARM Cortex-M3 core. The ROM based power configuration API configures the part for each reduced power mode. The following modes are supported in order from highest to lowest power consumption:

1. Active mode: The part is in active mode after a POR and when it is fully powered and operational after booting.
2. Sleep mode:

   The sleep mode affects the ARM Cortex-M3 core only. The clock to the core is shut off. Peripherals and memories are active and operational.
3. Deep-sleep and power-down modes:

   The Deep-sleep and power-down modes affect the core and the entire system with memories and peripherals. In both modes, the clock to the core is shut down and the peripherals receive no internal clocks. All SRAM and registers maintain their internal states.

   Through the power profiles API, you can opt to keep several peripherals running for safe operation of the part (WWDT and BOD) or for monitoring analog inputs (comparators and internal voltage reference and temperature sensor via one of the comparators).

   The differences between Deep-sleep mode and Power-down modes are the following:

a. In Deep-sleep mode, the flash is in stand-by mode and the IRC is running with its output disabled to minimize wake-up time.

b. In Power-down mode, the flash and the IRC are powered down to conserve power at the expense of longer wake-up times.

4. Deep power-down mode:

   For maximal power savings, the entire system (the core and all peripherals) is shut down except for the PMU with the general purpose registers (GPREG) and the RTC. On wake-up, the part reboots.

Table 83. Peripheral configuration in reduced power modes

Peripheral	Sleep mode	Deep-sleep mode	Power-down mode	Deep power-down mode
IRC	software configurable	on	off	off
IRC output	software configurable	off	off	off
Flash	software configurable	on	off	off
BOD	software configurable	software configurable	software configurable	off
PLL	software configurable	off	off	off
SysOsc	software configurable	off	off	off
WDosc/WWDT	software configurable	software configurable	software configurable	off
USART	software configurable	off; but can create wake-up interrupt in synchronous slave mode or 32 kHz clock mode	off; but can create wake-up interrupt in synchronous slave mode or 32 kHz clock mode	off
SPI	software configurable	off; but can create wake-up interrupt in slave mode	off; but can create wake-up interrupt in slave mode	off
I2C	software configurable	off; but can create wake-up interrupt in slave mode	off; but can create wake-up interrupt in slave mode	off
Other digital peripherals	software configurable	off	off	off
Analog peripherals	software configurable	software configurable	software configurable	off
RTC oscillator	software configurable	software configurable	software configurable	software configurable

4.5.1 Wake-up process

The part always wakes up to the active mode. To wake up from the reduced power modes, you must configure the wake-up source. Each reduced power mode supports its own wake-up sources and needs to be configured accordingly as shown in Table 84.

Table 84. Wake-up sources for reduced power modes

Power mode	Wake-up source	Conditions
Sleep	Any interrupt	Enable interrupt in NVIC.
Deep-sleep and Power-down	Pin interrupts	Enable pin interrupts in NVIC and STARTERP0 registers.
	BOD interrupt	• Enable interrupt in NVIC and STARTERP0 registers. • Enable interrupt in BODCTRL register. • Configure the BOD to keep running in this mode with the power API.
	BOD reset	Enable reset in BODCTRL register.
	WWDT interrupt	• Enable interrupt in NVIC and STARTERP0 registers. • WWDT running. Enable WWDT in WWDT MOD register and feed. • Enable interrupt in WWDT MOD register. • Configure the WDOSC to keep running in this mode with the power API.
	WWDT reset	• WWDT running. • Enable reset in WWDT MOD register.
	ACMP output/temperature sensor/internal voltage reference	• Enable analog comparator interrupt in the NVIC and STARTERP1 register. • Configure the analog comparators or the temperature sensor or the internal voltage reference to keep running with the power API.
	RTC 1 Hz alarm timer	• Enable the RTC 1 Hz oscillator in the RTC CTRL register. • Start RTC alarm timer by writing a time-out value to the RTC COUNT register. • Enable the RTCALARM interrupt in the STARTERP1 register.
	RTC 1 kHz timer time-out and alarm	• Enable the RTC 1 Hz oscillator and the RTC 1 kHz oscillator in the RTC CTRL register. • Start RTC 1 kHz timer by writing a time-out value to the RTC WAKE register. • Enable the RTCWAKE interrupt in the STARTERP1 register.
	I2C interrupt	Interrupt from I2C in slave mode. See Section 26.4.3.2.
	SPI interrupt	Interrupt from SPI in slave mode. See Section 25.3.1.2.
	USART interrupt	Interrupt from USART in slave or 32 kHz mode. See Section 24.3.2.2.
	USB interrupt	USB wake signal. See Section 23.7.6.1.
Deep power-down	WAKEUP pin PIO0_17	Enable the WAKEUP function in the GPREG4 register in the PMU. This is the default.
	RTC 1 Hz alarm timer	• Enable the RTC 1 Hz oscillator in the RTC CTRL register. • Start RTC alarm timer by writing a time-out value to the RTC COUNT register.
	RTC 1 kHz timer time-out and alarm	• Enable the RTC 1 Hz oscillator and the RTC 1 kHz oscillator in the RTC CTRL register. • Start RTC 1 kHz timer by writing a time-out value to the RTC WAKE register.

4.6 Register description

Table 85. Register overview: PMU (base address 0x4003 C000)

Name	Access	Address offset	Description	Reset value	Reference
PCON	R/W	0x000	Power control register	0x0	Table 86
GPREG0	R/W	0x004	General purpose register 0	0x0	Table 87
GPREG1	R/W	0x008	General purpose register 1	0x0	Table 87
GPREG2	R/W	0x00C	General purpose register 2	0x0	Table 87
GPREG3	R/W	0x010	General purpose register 3	0x0	Table 87
GPREG4	R/W	0x014	General purpose register 4. WAKEUP pad control.	0x0	Table 88

4.6.1 Power control register

The power control register provides a lock mechanism to prevent the part from entering Deep power-down mode. Together with the BOD and WWDT, the lock bit allows safe operation of the part at all times.

In addition, this register sets the flags that indicate what reduced power mode the part has exited on wake-up.

Table 86. Power control register (PCON, address 0x4003 C000) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved.	000
3	NODPD		A 1 in this bit prevents entry to Deep power-down mode. This bit is cleared by power-on reset (POR).	0
7:4	-	-	Reserved. Do not write ones to this bit.	0
8	SLEEPFLAG		Sleep mode flag	0
		0	Read: No power-down mode entered. The part is in Active mode. Write: No effect.	
		1	Read: Sleep, Deep-sleep, or Power-down mode entered. Write: Writing a 1 clears the SLEEPFLAG bit to 0.	
10:9	-	-	Reserved. Do not write ones to this bit.	0
11	DPDFLAG		Deep power-down flag	0
		0	Read: Deep power-down mode **not** entered. Write: No effect.	0
		1	Read: Deep power-down mode entered. Write: Clear the Deep power-down flag.	
31:12	-	-	Reserved. Do not write ones to this bit.	0

4.6.2 General purpose registers 0 to 3

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

Table 87. General purpose registers 0 to 3 (GPREG[0:3], address 0x4003 C004 (GPREG0) to 0x4003 C010 (GPREG3)) bit description

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

4.6.3 General purpose register 4

This register provides extra bits for data to be retained during Deep power-down mode. In addition, this register configures the functionality and the hysteresis of the WAKEUP pin (PIO0_17).

The bits in the register not used for the WAKEUP pin control (bits 31:10) can be used for storing additional data which are retained in Deep power-down mode in the same way as registers GPREG0 to GPREG3.

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

Table 88. General purpose register 4 (GPREG4, address 0x4003 C014) bit description

Bit	Symbol	Value	Description	Reset value
0	WAKEUPHYS		WAKEUP pin hysteresis enable	0
		0	Disabled. Hysteresis for WAKEUP pin disabled.	
		1	Enabled. Hysteresis for WAKEUP pin enabled.	
1	WAKEPAD_ DISABLE		WAKEUP pin disable. Setting this bit disables the wake-up pin, so it can be used for other purposes. **Remark:** Never set this bit if you intend to use a pin to wake up the part from Deep power-down mode. You can only disable the wake-up pin if the RTC wake-up is enabled and configured. **Remark:** Setting this bit is not necessary if Deep power-down mode is not used.	0
		0	Enabled. The wake-up function is enabled on pin PIO0_17.	
		1	Disabled. Setting this bit disables the wake-up function on pin PIO0_17.	
9:2	-		Reserved. Do not write 1s to reserved bits.	0x0
31:10	-		Data retained during Deep power-down mode.	0x0

4.7 Functional description

4.7.1 Power management

The LPC15xx support a variety of power control features. In Active mode, when the chip is running, power and clocks to selected peripherals can be optimized for power consumption. In addition, there are four special modes of processor power reduction with different peripherals running: Sleep mode, Deep-sleep mode, Power-down mode, and Deep power-down mode.

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

4.7.2 Active mode

In Active mode, the ARM Cortex-M3 core, memories, and peripherals are clocked by the system clock or main clock.

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

4.7.2.1 Power configuration in Active mode

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

- The SYSAHBCLKCTRL registers controls which memories and peripherals are running (Table 50 and Table 51).
- The power to various analog blocks (PLL, oscillators, the BOD circuit, and the flash block) can be controlled at any time individually through the PDRUNCFG register (Table 75 “Power configuration register (PDRUNCFG, address 0x4007 4208) bit description”).
- The clock source for the system clock can be selected from the IRC (default), the system oscillator, the 32 kHz oscillator, or the watchdog oscillator (see Figure 3 and related registers).
- The system clock frequency can be selected by the SYSPLLCTRL (Table 68) and the SYSAHBCLKDIV register (Table 49). You can find optimal settings for setting the system PLL by using the set_pll routine in the power API (Table 515).
- Several peripherals use individual peripheral clocks with their own clock dividers. The peripheral clocks can be shut down through the corresponding clock divider registers.
- The power API provides an easy way to optimize power consumption depending on CPU load and performance requirements. See Section 35.3.

4.7.3 Sleep mode

In Sleep mode, the system clock to the ARM Cortex-M3 core is stopped and execution of instructions is suspended until either a reset or an interrupt occurs.

Peripheral functions, if selected to be clocked in the SYSAHBCLKCTRL registers, 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.

As in active mode, the power API provides an easy way to optimize power consumption depending on CPU load and performance requirements in sleep mode. See Section 35.3.

4.7.3.1 Power configuration in Sleep mode

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

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

4.7.3.2 Programming Sleep mode

The following steps must be performed to enter Sleep mode:

1. In the NVIC, enable all interrupts that are needed to wake up the part.
2. Call power API: `pPWRD->power_mode_configure(SLEEP, peripheral);`

   Remark: The `peripheral` parameter is don't care.
3. Issue the ARM Cortex-M3 Wait-For-Interrupt (WFI) instruction.

4.7.3.3 Wake-up from Sleep mode

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

4.7.4 Deep-sleep mode

In Deep-sleep mode, the system clock to the processor is disabled as in Sleep mode. All analog blocks are powered down by default but can be selected to keep running through the power API if needed as wake-up sources. The main clock, and therefore all peripheral clocks, are disabled. The IRC is running, but its output is disabled. The flash is in stand-by mode.

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

4.7.4.1 Power configuration in Deep-sleep mode

Power consumption in Deep-sleep mode is determined by which analog wake-up sources are enabled. Serial peripherals and pin interrupts configured to wake up the part do not contribute to the power consumption. All wake-up events (from analog and serial peripherals) must be enabled in the STARTERP registers and in the NVIC. In addition, each analog block must be explicitly enabled through the power API function power_mode_configure() for wake-up. See Table 84.

4.7.4.2 Programming Deep-sleep mode

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

1. Select wake-up sources and enable all selected wake-up events in the STARTERP registers (Table 76 and Table 77) and in the NVIC.
2. Select the power configuration after wake-up in the PDAWAKECFG (Table 74) register.
3. Select the IRC as the main clock. See Table 40.
4. Call the power API with the parameter `peripheral` set to enable the analog peripherals the serve as wake-up sources (see Table 518 “Bit values for the power_mode_configure peripheral parameter”):
   `pPWRD->power_mode_configure(DEEP_SLEEP, peripheral);`
5. Use the ARM WFI instruction.

4.7.4.3 Wake-up from Deep-sleep mode

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

- Using a signal on one of the eight pin interrupts selected in Table 131. Each pin interrupt must also be enabled in the STARTERP0 register (Table 76) and in the NVIC.
- Using an interrupt from any analog block configured in the power API. Also enable the wake-up sources in the STARTERP registers (Table 76 and Table 77) and the NVIC.
- Using a reset from the $\overline{\text{RESET}}$ pin, or the BOD or WWDT (if enabled in the power API).
- Using a wake-up signal from any of the serial peripherals. Also enable the wake-up sources in the STARTERP registers (Table 76 and Table 77) and the NVIC.
- GPIO group interrupt signal. Interrupt must also be enabled in the STARTERP1 register (Table 76) and in the NVIC.
- RTC alarm signal or wake-up signal. See Section 18.1. Interrupts must also be enabled in the STARTERP1 register (Table 76) and in the NVIC.

4.7.5 Power-down mode

In Power-down mode, the system clock to the processor is disabled as in Sleep mode. All analog blocks are powered down by default but can be selected to keep running if needed for waking up the part. The main clock and therefore all peripheral clocks are disabled except for the clock to the watchdog timer if the watchdog oscillator is selected. The IRC itself and the flash are powered down, decreasing power consumption compared to Deep-sleep mode.

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

4.7.5.1 Power configuration in Power-down mode

Power consumption in power-down mode is determined by which analog wake-up sources are enabled. Serial peripherals and pin interrupts configured to wake up the part do not contribute to the power consumption. All wake-up events (from analog and serial peripherals) must be enabled in the STARTERP registers and in the NVIC. In addition, each analog block must be explicitly enabled through the power API function power_mode_configure() for wake-up. See Table 84.

4.7.5.2 Programming Power-down mode

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

1. Select wake-up sources and enable all related wake-up events in the STARTERP registers (Table 76 and Table 77) and in the NVIC.
2. Select the power configuration after wake-up in the PDAWAKECFG (Table 74) register.
3. Select the IRC as the main clock. See Table 40.
4. Call the power API with the peripheral parameter set to enable the analog wake-up sources: `pPWRD->power_mode_configure(POWER_DOWN, peripheral);`
5. Use the ARM WFI instruction.

4.7.5.3 Wake-up from Power-down mode

The part can wake up from Power-down mode in the following ways:

- Using a signal on one of the eight pin interrupts selected in Table 131. Each pin interrupt must also be enabled in the STARTERP0 register (Table 76) and in the NVIC.
- Using an interrupt from any analog block configured in the power API. Also enable the wake-up sources in the STARTERP registers (Table 76 and Table 77) and the NVIC.
- Using a reset from the $\overline{\text{RESET}}$ pin, or the BOD or WWDT (if enabled in the power API).
- Using a wake-up signal from any of the serial peripherals. Also enable the wake-up sources in the STARTERP registers (Table 76 and Table 77) and the NVIC.
- GPIO group interrupt signal. Interrupt must also be enabled in the STARTERP1 register (Table 76) and in the NVIC.
- RTC alarm signal or wake-up signal. See Section 18.1. Interrupts must also be enabled in the STARTERP1 register (Table 76) and in the NVIC.

4.7.6 Deep power-down mode

In Deep power-down mode, power and clocks are shut off to the entire chip with the exception of the WAKEUP pin and the RTC.

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

All functional pins are tri-stated in Deep power-down mode except for the WAKEUP pin. In this mode, you must pull the $\overline{\text{RESET}}$ pin HIGH externally.

Remark: Setting bit 3 in the PCON register (Table 86) prevents the part from entering Deep-power down mode.

4.7.6.1 Power configuration in Deep power-down mode

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

4.7.6.2 Programming Deep power-down mode using the WAKEUP pin:

The following steps must be performed to enter Deep power-down mode when using the WAKEUP pin for waking up:

1. Pull the WAKEUP pin externally HIGH.
2. Ensure that bit 3 in the PCON register (Table 86) is cleared.
3. Store data to be retained in the general purpose registers (Section 4.6.2).
4. Call the power API: `pPWRD->power_mode_configure(DEEP_POWER_DOWN, peripheral);`

   Remark: The `peripheral` parameter is don't care.
5. Use the ARM WFI instruction.

4.7.6.3 Wake-up from Deep power-down mode using the WAKEUP pin:

Pulling the WAKEUP pin LOW wakes up the part from Deep power-down, and the chip goes through the entire reset process.

1. On the WAKEUP pin, transition from HIGH to LOW.
   - The PMU will turn on the on-chip voltage regulator. When the core voltage reaches the power-on-reset (POR) trip point, a system reset will be triggered and the chip re-boots.
   - All registers except the DPDCTRL and GPREG0 to GPREG3registers will be in their reset state.
2. Once the chip has booted, read the deep power-down flag in the PCON register (Table 86) to verify that the reset was caused by a wake-up event from Deep power-down and was not a cold reset.
3. Clear the deep power-down flag in the PCON register (Table 86).
4. (Optional) Read the stored data in the general purpose registers (Section 4.6.2).
5. Set up the PMU for the next Deep power-down cycle.

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

4.7.6.4 Programming Deep power-down mode using the RTC for wake-up:

The following steps must be performed to enter Deep power-down mode when using the RTC for waking up:

1. Set up the RTC high resolution timer. Write to the RTC VAL register. This starts the high res timer if enabled. Another option is to use the 1 Hz alarm timer.
2. Ensure that bit 3 in the PCON register (Table 86) is cleared.
3. Store data to be retained in the general purpose registers (Section 4.6.2).
4. Call the power API: `pPWRD->power_mode_configure(DEEP_POWER_DOWN, peripheral);`

Remark: The `peripheral` parameter is don't care.

5. Use the ARM WFI instruction.

4.7.6.5 Wake-up from Deep power-down mode using the RTC:

The part goes through the entire reset process when the RTC times out:

1. When the high resolution timer count reaches 0, the following happens:
   - The PMU will turn on the on-chip voltage regulator. When the core voltage reaches the power-on-reset (POR) trip point, a system reset will be triggered and the chip boots.
   - All registers except the DPDCTRL and GPREG0 to GPREG3 registers will be in their reset state.
2. Once the chip has booted, read the deep power-down flag in the PCON register (Table 86) to verify that the reset was caused by a wake-up event from Deep power-down and was not a cold reset.
3. Clear the deep power-down flag in the PCON register (Table 86).
4. (Optional) Read the stored data in the general purpose registers (Section 4.6.2).
5. Set up the PMU for the next Deep power-down cycle.

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

5.1 How to read this chapter

USB drivers and the USB boot option are available on parts LPC1549/48/47 only.

5.2 Features

- 32 kB on-chip boot ROM
- Contains the boot loader with In-System Programming (ISP) facility and the following APIs:
  - In-Application Programming (IAP) of flash memory
  - Power profiles for optimizing power consumption and system performance and for controlling low power modes
  - USART drivers
  - C_CAN drivers
  - I2C drivers
  - DMA drivers
  - SPI drivers

5.3 Pin description

The parts support ISP via the USART0, C_CAN, or USB interfaces. The ISP mode is determined by the state of two pins at boot time:

Table 89. ISP modes

ISP mode	ISP_0	ISP_1	Description
No ISP	HIGH	HIGH	ISP bypassed. Part attempts to boot from flash. If the part is not programmed or contains invalid user code, this mode will enter ISP via USB. **Remark:** If IAP function Reinvoke ISP is called, and both ISP pins are HIGH, the part enters one of the ISP mode depending on the parameter passed with the IAP function. See Table 508.
C_CAN	HIGH	LOW	Part enters ISP via C_CAN.
USB	LOW	HIGH	Part enters ISP via USB.
USART0	LOW	LOW	Part enters ISP via USART0.

The ISP pin assignment is different for each package, so that the fewest functions possible are blocked. No more than four pins must be set aside for entering ISP in any ISP mode. The boot code assigns two ISP pins for each package which are probed when the part boots to determine whether or not to enter ISP mode. Once the ISP mode has been determined, the boot loader configures the necessary serial pins for each package.

Pins which are not configured by the boot loader for the selected boot mode (for example CAN0_RD and CAN0_TD in USART mode) can be assigned to any function through the switch matrix.

Table 90. ISP pin assignments

ISP pin	LQFP48	LQFP64	LQFP100
ISP_0	PIO0_4	PIO1_9	PIO2_5
ISP_1	PIO0_16	PIO1_11	PIO2_4
USART mode			
U0_TXD	PIO0_15	PIO0_18	PIO2_6
U0_RXD	PIO0_14	PIO0_13	PIO2_7
C_CAN mode			
CAN0_TD	PIO0_18	PIO0_31	PIO2_8
CAN0_RD	PIO0_13	PIO0_11	PIO2_9
USB mode			
USB_VBUS (same as ISP_1)	PIO0_16	PIO1_11	PIO2_4

Table 91. Default pin assignments for different ISP modes

Symbol	LQFP48	LQFP64	LQFP100
PIO0_4/ADC0_4	ISP_0	-	-
PIO0_11/ADC1_3	-	CAN0_RD (C_CAN mode only)	-
PIO0_13/ADC1_6	CAN0_RD (C_CAN mode only)	U0_RXD (USART mode only)	-
PIO0_14/ADC1_7/ SCT1_OUT5	U0_RXD (USART mode only)	-	-
PIO0_15/ADC1_8	U0_TXD (USART mode only)	-	-
PIO0_16/ADC1_9	ISP_1. In USB mode also USB_VBUS.	-	-
PIO0_18/SCT0_OUT5	CAN0_TD (C_CAN mode only)	U0_TXD (USART mode only)	-
PIO0_31/ADC0_9	-	CAN0_TD (C_CAN mode only)	-
PIO1_9/ACMP2_I4	-	ISP_0	-
PIO1_11	-	ISP_1. In USB mode also USB_VBUS.	-
PIO2_4	-	-	ISP_1. In USB mode also USB_VBUS.
PIO2_5	-	-	ISP_0
PIO2_6	-	-	U0_TXD (USART mode only)

Table 91. Default pin assignments for different ISP modes *…continued*

Symbol	LQFP48	LQFP64	LQFP100
PIO2_7	-	-	U0_RXD (USART mode only)
PIO2_8	-	-	CAN0_TD (C_CAN mode only)
PIO2_9	-	-	CAN0_RD (C_CAN mode only)

5.4 General description

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

The boot loader code is executed every time the part is powered on or reset (see Figure 7). The loader can execute the ISP command handler or the user application code. See Table 89 “ISP modes” for different boot modes.

The boot loader version can be read by ISP/IAP calls (see Table 492 or Table 506).

Assuming that power supply pins are at their nominal levels when the rising edge on $\overline{\text{RESET}}$ pin is generated, it may take up to 3 ms before the boot pins are sampled and the decision whether to continue with user code or ISP handler is made. If the boot pins are 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, 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.

Remark: In USB ISP mode, an external 12 MHz crystal is required to communicate with the external storage device.

See Chapter 34 “LPC15xx Flash/EEPROM API” for the ISP and IAP commands.

5.4.1 Boot process flowchart

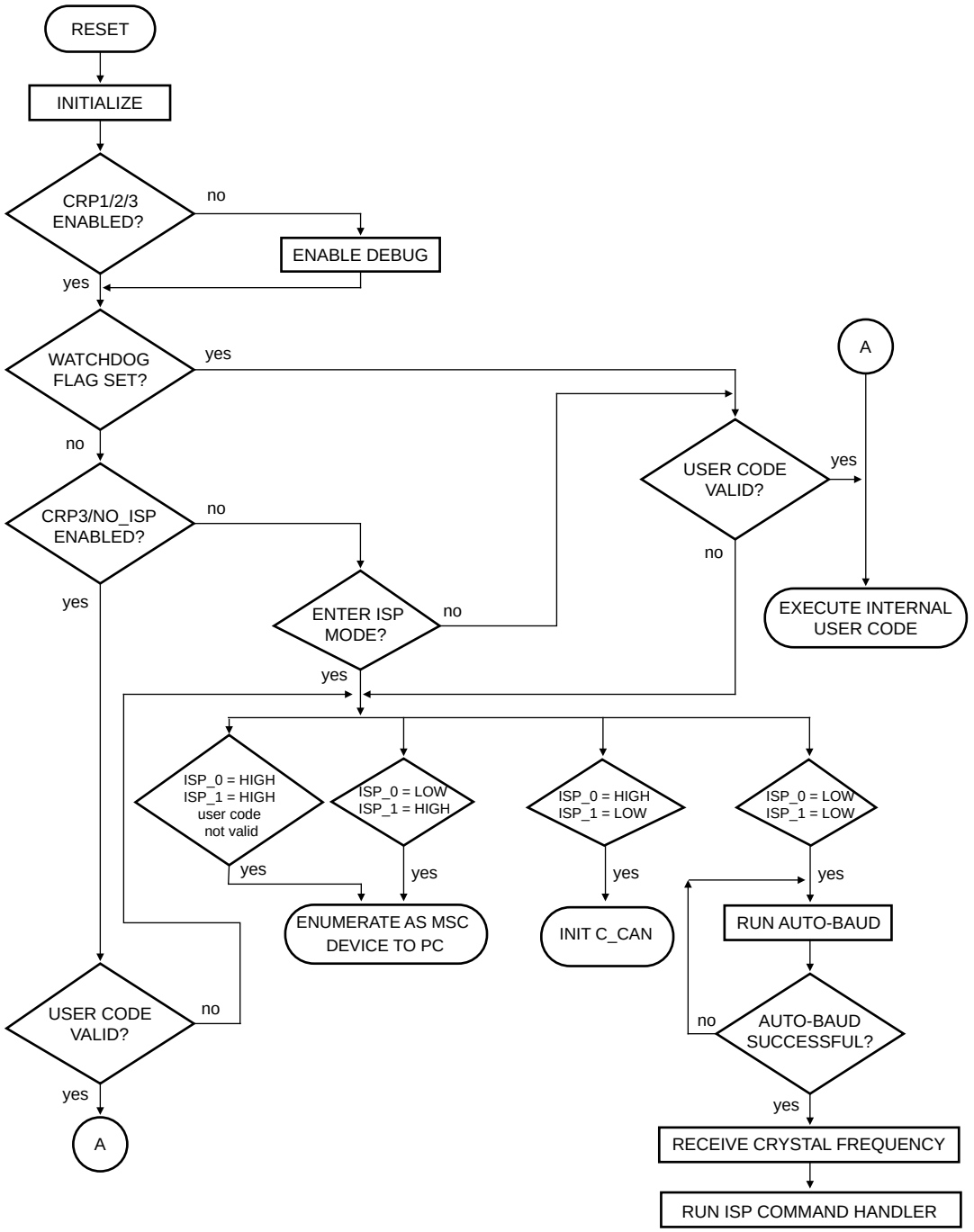


Fig 7. Boot process flowchart

5.4.2 Criterion for valid user code

The reserved ARM Cortex-M3 exception vector location 7 (offset 0x0000 001C in the vector table) should contain the 2's complement of the check-sum of table entries 0 through 6. This causes the checksum of the first 8 table entries to be 0. The boot loader code checksums the first 8 locations in sector 0 of the flash. If the result is 0, then execution control is transferred to the user code.

If the signature is not valid, the part enumerates as a USB MSC device. See Figure 7 "Boot process flowchart" and Section 34.5 "USB communication protocol".

5.4.3 ROM-based APIs

Once the part has booted, the user code can access several APIs located in the boot ROM to access the flash memory, optimize power consumption, and operate the peripherals.

The location of the APIs in boot ROM is shown in Figure 8.

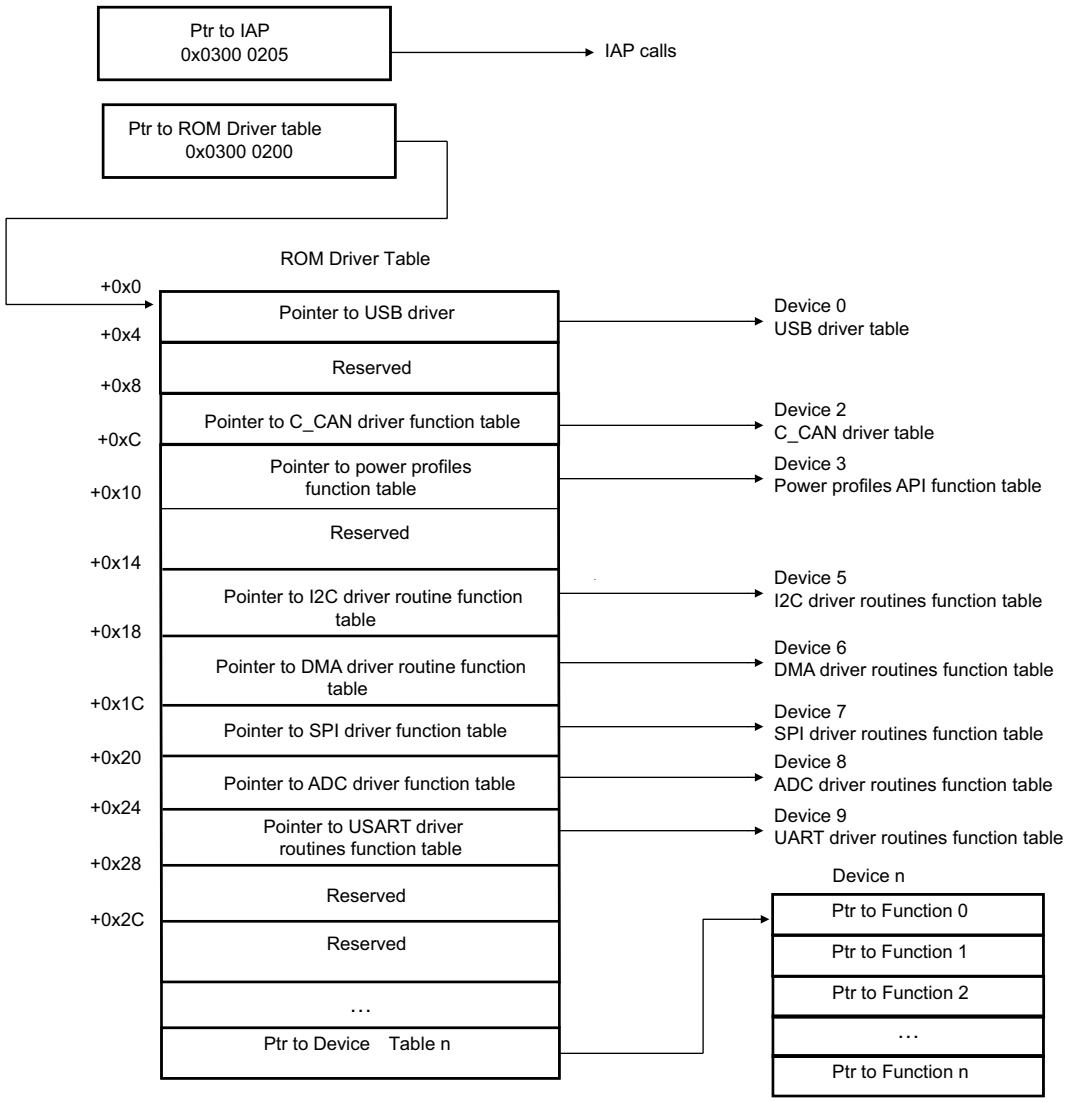


Fig 8. Boot ROM structure

Table 92. API reference

API	Description	Reference
Flash IAP	Flash In-Application programming	Table 500
USB API	USB driver	Section 42.3.1
C_CAN API	C_CAN driver	Table 583
Power profiles API	Configure system clock and power consumption. Control low power modes and wake-up.	Table 514
I2C driver	I2C ROM driver	Table 542
DMA driver	DMA ROM driver	Table 531

Table 92. API reference

API	Description	Reference
SPI driver	SPI ROM driver	Table 575
ADC driver	ADC ROM driver	Table 563
USART driver	USART ROM driver for USART0/1/2	Table 521

```
typedef struct {
     const USBD_API_T *pUSBD;  /*!< USBD API function table base address */
     const uint32_t reserved0; /*!< Reserved */
     const CAND_API_T *pCAND; /*!< C_CAN API function table base address */
     const PWRD_API_T *pPWRD; /*!< Power API function table base address */
     const reserved1;      /*!< reserved */
     const I2CD_API_T *pI2CD;  /*!< I2C driver API function table base address */
     const DMAD_API_T *pDMAD; /*!< DMA driver API function table base address */
     const SPID_API_T *pSPID; /*!< SPI driver API function table base address */
     const ADCD_API_T *pADCD; /*!< ADC driver API function table base address  */
     const UARTD_API_T *pUARTD; /*!< USART  driver API function table base address */
} LPC_ROM_API_T;

#define ROM_DRIVER_BASE (0x03000200UL)
```

6.1 Pin description

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
PIO0_0/ADC0_10/ SCT0_OUT3	1	2	2	[2]	I; PU	IO	**PIO0_0 —** General purpose port 0 input/output 0.
						A	**ADC0_10 —** ADC0 input 10.
						O	**SCT0_OUT3 —** SCTimer0/PWM output 3.
PIO0_1/ADC0_7/ SCT0_OUT4	2	5	6	[2]	I; PU	IO	**PIO0_1 —** General purpose port 0 input/output 1.
						A	**ADC0_7 —** ADC0 input 7.
						O	**SCT0_OUT4 —** SCTimer0/PWM output 4.
PIO0_2/ADC0_6/ SCT1_OUT3	3	6	8	[2]	I; PU	IO	**PIO0_2 —** General purpose port 0 input/output 2.
							ADC0_6 — ADC0 input 6.
						O	**SCT1_OUT3 —** SCTimer1/PWM output 3.
PIO0_3/ADC0_5/ SCT1_OUT4	4	7	10	[2]	I; PU	IO	**PIO0_3 —** General purpose port 0 input/output 3.
						A	**ADC0_5 —** ADC0 input 5.
						O	**SCT1_OUT4 —** SCTimer1/PWM output 4.
PIO0_4/ADC0_4	5	8	13	[2]	I; PU	IO	**PIO0_4 —** General purpose port 0 input/output 4. This is the ISP_0 boot pin for the LQFP48 package.
						A	**ADC0_4 —** ADC0 input 4.
PIO0_5/ADC0_3	6	9	14	[2]	I; PU	IO	**PIO0_5 —** General purpose port 0 input/output 5.
						A	**ADC0_3 —** ADC0 input 3.
PIO0_6/ADC0_2/ SCT2_OUT3	7	10	16	[2]	I; PU	IO	**PIO0_6 —** General purpose port 0 input/output 6.
						A	**ADC0_2 —** ADC0 input 2.
						O	**SCT2_OUT3 —** SCTimer2/PWM output 3.
PIO0_7/ADC0_1	8	11	17	[2]	I; PU	IO	**PIO0_7 —** General purpose port 0 input/output 7.
						A	**ADC0_1 —** ADC0 input 1.
PIO0_8/ADC0_0/TDO	9	12	19	[2]	I; PU	IO	**PIO0_8 —** General purpose port 0 input/output 8. In boundary scan mode: TDO (Test Data Out).
						A	**ADC0_0 —** ADC0 input 0.
PIO0_9/ADC1_1/TDI	12	16	24	[2]	I; PU	IO	**PIO0_9 —** General purpose port 0 input/output 9. In boundary scan mode: TDI (Test Data In).
						A	**ADC1_1 —** ADC1 input 1.
PIO0_10/ADC1_2	15	19	28	[2]	I; PU	IO	**PIO0_10 —** General purpose port 0 input/output 10.
						A	**ADC1_2 —** ADC1 input 2.
PIO0_11/ADC1_3	18	23	33	[2]	I; PU	IO	**PIO0_11 —** General purpose port 0 input/output 11. On the LQFP64 package, this pin is assigned to CAN0_RD in ISP C_CAN mode.
						A	**ADC1_3 —** ADC1 input 3.

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
PIO0_12/DAC_OUT	19	24	35	[3]	I; PU	IO	**PIO0_12 —** General purpose port 0 input/output 12. If this pin is configured as a digital input, the input voltage level must not be higher than V_{DDA}.
						A	**DAC_OUT —** DAC analog output.
PIO0_13/ADC1_6	21	29	43	[2]	I; PU	IO	**PIO0_13 —** General purpose port 0 input/output 13. On the LQFP64 package, this pin is assigned to U0_RXD in ISP USART mode. On the LQFP48 package, this pin is assigned to CAN0_RD in ISP C_CAN mode.
						A	**ADC1_6 —** ADC1 input 6.
PIO0_14/ADC1_7/ SCT1_OUT5	22	30	45	[2]	I; PU	IO	**PIO0_14 —** General purpose port 0 input/output 14. On the LQFP48 package, this pin is assigned to U0_RXD in ISP USART mode.
						A	**ADC1_7 —** ADC1 input 7.
						O	**SCT1_OUT5 —** SCTimer1/PWM output 5.
PIO0_15/ADC1_8	23	31	47	[2]	I; PU	IO	**PIO0_15 —** General purpose port 0 input/output 15. On the LQFP48 package, this pin is assigned to U0_TXD in ISP USART mode.
						A	**ADC1_8 —** ADC1 input 8.
PIO0_16/ADC1_9	24	32	49	[2]	I; PU	IO	**PIO0_16 —** General purpose port 0 input/output 16. On the LQFP48 package, this is the ISP_1 boot pin.
						A	**ADC1_9 —** ADC1 input 9.
PIO0_17/WAKEUP/ TRST	28	39	61	[4]	I; PU	IO	**PIO0_17 —** General purpose port 0 input/output 17. In boundary scan mode: TRST (Test Reset). This pin triggers a wake-up from Deep power-down mode. If you need to wake up from Deep power-down mode via an external pin, do not assign any movable function to this pin. Pull this pin HIGH externally while in Deep power-down mode. Pull this pin LOW to exit Deep power-down mode. A LOW-going pulse as short as 50 ns wakes up the part.
PIO0_18/ SCT0_OUT5	13	17	26	[5]	I; PU	IO	**PIO0_18 —** General purpose port 0 input/output 18. On the LQFP64 package, this pin is assigned to U0_TXD in ISP USART mode. On the LQFP48 package, this pin is assigned to CAN0_TD in ISP C_CAN mode.
						O	**SCT0_OUT5 —** SCTimer0/PWM output 5.
SWCLK/ PIO0_19/TCK	29	40	63	[5]	I; PU	I	**SWCLK —** Serial Wire Clock. SWCLK is enabled by default on this pin. In boundary scan mode: TCK (Test Clock).
						IO	**PIO0_19 —** General purpose port 0 input/output 19.

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
SWDIO/ PIO0_20/SCT1_OUT6/ TMS	33	44	69	[5]	I; PU	I/O	**SWDIO —** Serial Wire Debug I/O. SWDIO is enabled by default on this pin. In boundary scan mode: TMS (Test Mode Select).
						I/O	**PIO0_20 —** General purpose port 0 input/output 20.
						O	**SCT1_OUT6 —** SCTimer1/PWM output 6.
$\overline{\text{RESET}}$/PIO0_21	34	45	71	[6]	I; PU	I	**$\overline{\text{RESET}}$ —** External reset input: A LOW-going pulse as short as 50 ns on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. In deep power-down mode, this pin must be pulled HIGH externally. The $\overline{\text{RESET}}$ pin can be left unconnected or be used as a GPIO or for any movable function if an external $\overline{\text{RESET}}$ function is not needed.
						I/O	**PIO0_21 —** General purpose port 0 input/output 21.
PIO0_22/I2C0_SCL	37	49	78	[7]	IA	IO	**PIO0_22 —** General purpose port 0 input/output 22.
						I/O	**I2C0_SCL —** I^2C-bus clock input/output. High-current sink if I^2C Fast-mode Plus is selected in the I/O configuration register.
PIO0_23/I2C0_SDA	38	50	79	[7]	IA	IO	**PIO0_23 —** General purpose port 0 input/output 23.
						I/O	**I2C0_SDA —** I2C-bus data input/output. High-current sink if I2C Fast-mode Plus is selected in the I/O configuration register.
PIO0_24/SCT0_OUT6	43	58	90	[8]	I; PU	IO	**PIO0_24 —** General purpose port 0 input/output 24. High-current output driver.
						O	**SCT0_OUT6 —** SCTimer0/PWM output 6.
PIO0_25/ACMP0_I4	44	60	93	[2]	I; PU	IO	**PIO0_25 —** General purpose port 0 input/output 25.
						A	**ACMP0_I4 —** Analog comparator 0 input 4.
PIO0_26/ACMP0_I3/ SCT3_OUT3	45	61	95	[2]	I; PU	IO	**PIO0_26 —** General purpose port 0 input/output 26.
						A	**ACMP0_I3 —** Analog comparator 0 input 3.
						O	**SCT3_OUT3 —** SCTimer3/PWM output 3.
PIO0_27/ACMP_I1	46	62	97	[2]	I; PU	IO	**PIO0_27 —** General purpose port 0 input/output 27.
						A	**ACMP_I1 —** Analog comparator common input 1.
PIO0_28/ACMP1_I3	47	63	98	[2]	I; PU	IO	**PIO0_28 —** General purpose port 0 input/output 28.
						A	**ACMP1_I3 —** Analog comparator 1 input 3.
PIO0_29/ACMP2_I3/ SCT2_OUT4	48	64	100	[2]	I; PU	IO	**PIO0_29 —** General purpose port 0 input/output 29.
						A	**ACMP2_I3 —** Analog comparator 2 input 3.
						O	**SCT2_OUT4 —** SCTimer2/PWM output 4.
PIO0_30/ADC0_11	-	1	1	[2]	I; PU	IO	**PIO0_30 —** General purpose port 0 input/output 30.
						A	**ADC0_11 —** ADC0 input 11.

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
PIO0_31/ADC0_9	-	3	3	[2]	I; PU	IO	**PIO0_31 —** General purpose port 0 input/output 31. On the LQFP64 package, this pin is assigned to CAN0_TD in ISP C_CAN mode.
						A	**ADC0_9 —** ADC0 input 9.
PIO1_0/ADC0_8	-	4	5	[2]	I; PU	IO	**PIO1_0 —** General purpose port 1 input/output 0.
						A	**ADC0_8 —** ADC0 input 8.
PIO1_1/ADC1_0	-	15	23	[2]	I; PU	IO	**PIO1_1 —** General purpose port 1 input/output 1.
						A	**ADC1_0 —** ADC1 input 0.
PIO1_2/ADC1_4	-	25	36	[2]	I; PU	IO	**PIO1_2 —** General purpose port 1 input/output 2.
						A	**ADC1_4 —** ADC1 input 4.
PIO1_3/ADC1_5	-	28	41	[2]	I; PU	IO	**PIO1_3 —** General purpose port 1 input/output 3.
						A	**ADC1_5 —** ADC1 input 5.
PIO1_4/ADC1_10	-	33	51	[2]	I; PU	IO	**PIO1_4 —** General purpose port 1 input/output 4.
						A	**ADC1_10 —** ADC1 input 10.
PIO1_5/ADC1_11	-	34	52	[2]	I; PU	IO	**PIO1_5 —** General purpose port 1 input/output 5.
						A	**ADC1_11 —** ADC1 input 11.
PIO1_6/ACMP_I2	-	46	73	[2]	I; PU	IO	**PIO1_6 —** General purpose port 1 input/output 6.
						A	**ACMP_I2 —** Analog comparator common input 2.
PIO1_7/ACMP3_I4	-	51	81	[2]	I; PU	IO	**PIO1_7 —** General purpose port 1 input/output 7.
						A	**ACMP3_I4 —** Analog comparator 3 input 4.
PIO1_8/ACMP3_I3/ SCT3_OUT4	-	53	84	[2]	I; PU	IO	**PIO1_8 —** General purpose port 1 input/output 8.
						A	**ACMP3_I3 —** Analog comparator 3 input 3.
						O	**SCT3_OUT4 —** SCTimer3/PWM output 4.
PIO1_9/ACMP2_I4	-	54	85	[2]	I; PU	IO	**PIO1_9 —** General purpose port 1 input/output 9. On the LQFP64 package, this is the ISP_0 boot pin.
						A	**ACMP2_I4 —** Analog comparator 2 input 4.
PIO1_10/ACMP1_I4	-	59	91	[2]	I; PU	IO	**PIO1_10 —** General purpose port 1 input/output 10.
						A	**ACMP1_I4 —** Analog comparator 1 input 4.
PIO1_11	-	38	58	[5]	I; PU	IO	**PIO1_11 —** General purpose port 1 input/output 11. On the LQFP64 package, this is the ISP_1 boot pin.
PIO1_12	-	-	9	[5]	I; PU	IO	**PIO1_12 —** General purpose port 1 input/output 12.
PIO1_13	-	-	11	[5]	I; PU	IO	**PIO1_13 —** General purpose port 1 input/output 13.
PIO1_14/SCT0_OUT7	-	-	12	[5]	I; PU	IO	**PIO1_14 —** General purpose port 1 input/output 14.
						O	**SCT0_OUT7 —** SCTimer0/PWM output 7.
PIO1_15	-	-	15	[5]	I; PU	IO	**PIO1_15 —** General purpose port 1 input/output 15.
PIO1_16	-	-	18	[5]	I; PU	IO	**PIO1_16 —** General purpose port 1 input/output 16.
PIO1_17/SCT1_OUT7	-	-	20	[5]	I; PU	IO	**PIO1_17 —** General purpose port 1 input/output 17.
						O	**SCT1_OUT7 —** SCTimer1/PWM output 7.
PIO1_18	-	-	25	[5]	I; PU	IO	**PIO1_18 —** General purpose port 1 input/output 18.

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
PIO1_19	-	-	29	[5]	I; PU	IO	**PIO1_19 —** General purpose port 1 input/output 19.
PIO1_20/SCT2_OUT5	-	-	34	[5]	I; PU	IO	**PIO1_20 —** General purpose port 1 input/output 20.
						O	**SCT2_OUT5 —** SCTimer2/PWM output 5.
PIO1_21	-	-	37	[5]	I; PU	IO	**PIO1_21 —** General purpose port 1 input/output 21.
PIO1_22	-	-	38	[5]	I; PU	IO	**PIO1_22 —** General purpose port 1 input/output 22.
PIO1_23	-	-	42	[5]	I; PU	IO	**PIO1_23 —** General purpose port 1 input/output 23.
PIO1_24/SCT3_OUT5	-	-	44	[5]	I; PU	IO	**PIO1_24 —** General purpose port 1 input/output 24.
						O	**SCT3_OUT5 —** SCTimer3/PWM output 5.
PIO1_25	-	-	46	[5]	I; PU	IO	**PIO1_25 —** General purpose port 1 input/output 25.
PIO1_26	-	-	48	[5]	I; PU	IO	**PIO1_26 —** General purpose port 1 input/output 26.
PIO1_27	-	-	50	[5]	I; PU	IO	**PIO1_27 —** General purpose port 1 input/output 27.
PIO1_28	-	-	55	[5]	I; PU	IO	**PIO1_28 —** General purpose port 1 input/output 28.
PIO1_29	-	-	56	[5]	I; PU	IO	**PIO1_29 —** General purpose port 1 input/output 29.
PIO1_30	-	-	59	[5]	I; PU	IO	**PIO1_30 —** General purpose port 1 input/output 30.
PIO1_31	-	-	60	[5]	I; PU	IO	**PIO1_31 —** General purpose port 1 input/output 31.
PIO2_0	-	-	62	[5]	I; PU	IO	**PIO2_0 —** General purpose port 2 input/output 0.
PIO2_1	-	-	64	[5]	I; PU	IO	**PIO2_1 —** General purpose port 2 input/output 1.
PIO2_2	-	-	72	[5]	I; PU	IO	**PIO2_2 —** General purpose port 2 input/output 2.
PIO2_3	-	-	76	[5]	I; PU	IO	**PIO2_3 —** General purpose port 2 input/output 3.
PIO2_4	-	-	77	[5]	I; PU	IO	**PIO2_4 —** General purpose port 2 input/output 4. On the LQFP100 package, this is the ISP_1 boot pin.
PIO2_5	-	-	80	[5]	I; PU	IO	**PIO2_5 —** General purpose port 2 input/output 5. On the LQFP100 package, this is the ISP_0 boot pin.
PIO2_6	-	-	82	[5]	I; PU	IO	**PIO2_6 —** General purpose port 2 input/output 6. On the LQFP100 package, this pin is assigned to U0_TXD in ISP USART mode.
PIO2_7	-	-	86	[5]	I; PU	IO	**PIO2_7 —** General purpose port 2 input/output 7. On the LQFP100 package, this pin is assigned to U0_RXD in ISP USART mode.
PIO2_8	-	-	92	[5]	I; PU	IO	**PIO2_8 —** General purpose port 2 input/output 8. On the LQFP100 package, this pin is assigned to CAN0_TD in ISP C_CAN mode.
PIO2_9	-	-	94	[5]	I; PU	IO	**PIO2_9 —** General purpose port 2 input/output 9. On the LQFP100 package, this pin is assigned to CAN0_RD in ISP C_CAN mode.
PIO2_10	-	-	96	[5]	I; PU	IO	**PIO2_10 —** General purpose port 2 input/output 10.
PIO2_11	-	-	99	[5]	I; PU	IO	**PIO2_11 —** General purpose port 2 input/output 11.
PIO2_12	35	47	74	[5]	I; PU	IO	**PIO2_12 —** General purpose port 2 input/output 12. On parts LPC1519/17/18 only.
PIO2_13	36	48	75	[5]	I; PU	IO	**PIO2_13 —** General purpose port 2 input/output 13. On parts LPC1519/17/18 only.

Table 93. Pin description

Symbol	LQFP48	LQFP64	LQFP100		Reset state[1]	Type	Description
USB_DP	35	47	74	[10]	-	IO	USB bidirectional D+ line. Pad includes internal 33 Ω series termination resistor. On parts LPC1549/48/47 only.
USB_DM	36	48	75	[10]	-	IO	USB bidirectional D− line. Pad includes internal 33 Ω series termination resistor. On parts LPC1549/48/47 only.
RTCXIN	31	42	66	[9]	-		RTC oscillator input. This input should be grounded if the RTC is not used.
RTCXOUT	32	43	67	[9]	-		RTC oscillator output.
XTALIN	26	36	54	[9]	-		Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V.
XTALOUT	25	35	53	[9]	-		Output from the oscillator amplifier.
VBAT	30	41	65		-		Battery supply voltage. If no battery is used, tie VBAT to VDD or to ground.
V_{DDA}	16	20	30		-		Analog supply voltage. V_{DD} and the analog reference voltages VREFP_ADC and VREFP_DAC_VDDCMP must not exceed the voltage level on V_{DDA}. V_{DDA}should typically be the same voltages as V_{DD} but should be isolated to minimize noise and error. V_{DDA} should be tied to V_{DD} if the ADC is not used.
V_{DD}	39, 27, 42	22, 52, 37, 57	4, 32, 70, 83, 57, 89		-		3.3 V supply voltage (2.4 V to 3.6 V). The voltage level on V_{DD} must be equal or lower than the analog supply voltage V_{DDA}.
VREFP_DAC_VDDCMP	14	18	27	[9]	-		DAC positive reference voltage and analog comparator reference voltage. The voltage level on VREFP_DAC_VDDCMP must be equal to or lower than the voltage applied to V_{DDA}.
VREFN	11	14	22		-		ADC and DAC negative voltage reference. If the ADC is not used, tie VREFN to V_{SS}.
VREFP_ADC	10	13	21		-		ADC positive reference voltage. The voltage level on VREFP_ADC must be equal to or lower than the voltage applied to V_{DDA}. If the ADC is not used, tie VREFP_ADC to V_{DD}.
V_{SSA}	17	21	31		-		Analog ground. V_{SSA}should typically be the same voltage as V_{SS} but should be isolated to minimize noise and error. V_{SSA} should be tied to V_{SS} if the ADC is not used.
V_{SS}	41, 20, 40	56, 26, 27, 55	88, 7, 39, 40, 68, 87		-		Ground.

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

[2] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors, configurable hysteresis, and analog input. When configured as analog input, digital section of the pad is disabled and the pin is not 5 V tolerant. This pin includes a 10 ns on/off glitch filter. By default, the glitch filter is turned on.

[3] This pin is not 5 V tolerant due to special analog functionality. When configured for a digital function, this pin is 3 V tolerant and provides standard digital I/O functions with configurable internal pull-up and pull-down resistors and hysteresis. When configured for DAC_OUT, the digital section of the pin is disabled and this pin is a 3 V tolerant analog output. This pin includes a 10 ns on/off glitch filter. By default, the glitch filter is turned on.

[4] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors, and configurable hysteresis. This pin includes a 10 ns on/off glitch filter. By default, the glitch filter is turned on. This pin is powered in deep power-down mode and can wake up the part.

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

[6] 5 V tolerant pad. $\overline{\text{RESET}}$ functionality is not available in Deep power-down mode. Use the WAKEUP pin to reset the chip and wake up from Deep power-down mode. An external pull-up resistor is required on this pin for the Deep power-down mode.

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

[9] Special analog pin.

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

Table 94. Movable functions

Function name	Type	Description
U0_TXD	O	Transmitter output for USART0.
U0_RXD	I	Receiver input for USART0.
$\overline{\text{U0\_RTS}}$	O	Request To Send output for USART0.
$\overline{\text{U0\_CTS}}$	I	Clear To Send input for USART0.
U0_SCLK	I/O	Serial clock input/output for USART0 in synchronous mode.
U1_TXD	O	Transmitter output for USART1.
U1_RXD	I	Receiver input for USART1.
$\overline{\text{U1\_RTS}}$	O	Request To Send output for USART1.
$\overline{\text{U1\_CTS}}$	I	Clear To Send input for USART1.
U1_SCLK	I/O	Serial clock input/output for USART1 in synchronous mode.
U2_TXD	O	Transmitter output for USART2.
U2_RXD	I	Receiver input for USART2.
U2_SCLK	I/O	Serial clock input/output for USART1 in synchronous mode.
SPI0_SCK	I/O	Serial clock for SPI0.
SPI0_MOSI	I/O	Master Out Slave In for SPI0.
SPI0_MISO	I/O	Master In Slave Out for SPI0.
SPI0_SSEL0	I/O	Slave select 0 for SPI0.
SPI0_SSEL1	I/O	Slave select 1 for SPI0.
SPI0_SSEL2	I/O	Slave select 2 for SPI0.
SPI0_SSEL3	I/O	Slave select 3 for SPI0.
SPI1_SCK	I/O	Serial clock for SPI1.
SPI1_MOSI	I/O	Master Out Slave In for SPI1.
SPI1_MISO	I/O	Master In Slave Out for SPI1.
SPI1_SSEL0	I/O	Slave select 0 for SPI1.
SPI1_SSEL1	I/O	Slave select 1 for SPI1.

Table 94. Movable functions *…continued*

Function name	Type	Description
CAN0_TD	O	CAN0 transmit.
CAN0_RD	I	CAN0 receive.
USB_VBUS	I	USB VBUS.
SCT0_OUT0	O	SCTimer0/PWM output 0.
SCT0_OUT1	O	SCTimer0/PWM output 1.
SCT0_OUT2	O	SCTimer0/PWM output 2.
SCT1_OUT0	O	SCTimer1/PWM output 0.
SCT1_OUT1	O	SCTimer1/PWM output 1.
SCT1_OUT2	O	SCTimer1/PWM output 2.
SCT2_OUT0	O	SCTimer2/PWM output 0.
SCT2_OUT1	O	SCTimer2/PWM output 1.
SCT2_OUT2	O	SCTimer2/PWM output 2.
SCT3_OUT0	O	SCTimer3/PWM output 0.
SCT3_OUT1	O	SCTimer3/PWM output 1.
SCT3_OUT2	O	SCTimer3/PWM output 2.
SCT_ABORT0	I	SCT abort 0.
SCT_ABORT1	I	SCT abort 1.
ADC0_PINTRIG0	I	ADC0 external pin trigger input 0.
ADC0_PINTRIG1	I	ADC0 external pin trigger input 1.
ADC1_PINTRIG0	I	ADC1 external pin trigger input 0.
ADC1_PINTRIG1	I	ADC1 external pin trigger input 1.
DAC_PINTRIG	I	DAC external pin trigger input.
DAC_SHUTOFF	I	DAC shut-off external input.
ACMP0_O	O	Analog comparator 0 output.
ACMP1_O	O	Analog comparator 1 output.
ACMP2_O	O	Analog comparator 2 output.
ACMP3_O	O	Analog comparator 3 output.
CLKOUT	O	Clock output.
ROSC	O	Analog comparator ring oscillator output.
ROSC_RESET	I	Analog comparator ring oscillator reset.
USB_FTOGGLE	O	USB frame toggle. Do not assign this function to a pin until a USB device is connected and the first SOF interrupt has been received by the device.
QEI_PHA	I	QEI phase A input.
QEI_PHB	I	QEI phase B input.
QEI_IDX	I	QEI index input.
GPIO_INT_BMAT	O	Output of the pattern match engine.
SWO	O	Serial wire output.

Table 95. Pins connected to the INPUT MUX and SCT IPU

Symbol	LQFP48	LQFP64	LQFP100	Description
PIO0_2/ADC0_6/SCT1_OUT3	3	6	8	SCT0 input mux
PIO0_3/ADC0_5/SCT1_OUT4	4	7	10	SCT0 input mux
PIO0_4/ADC0_4	5	8	13	SCT2 input mux
PIO0_5/ADC0_3	6	9	14	FREQMEAS
PIO0_7/ADC0_1	8	11	17	SCT3 input mux
PIO0_14/ADC1_7/SCT1_OUT5	22	30	45	SCTIPU input SAMPLE_IN_A0
PIO0_15/ADC1_8	23	31	47	SCT1 input mux
PIO0_16/ADC1_9	24	32	49	SCT1 input mux
PIO0_17/WAKEUP/$\overline{\text{TRST}}$	28	39	61	SCT0 input mux
SWCLK/PIO0_19/TCK	29	40	63	FREQMEAS
$\overline{\text{RESET}}$/PIO0_21	34	45	71	SCT1 input mux
PIO0_25/ACMP0_I4	44	60	93	SCTIPU input SAMPLE_IN_A1
PIO0_27/ACMP_I1	46	62	97	SCT2 input mux
PIO0_30/ADC0_11	-	1	1	FREQMEAS
				SCT0 input mux
PIO0_31/ADC0_9	-	3	3	SCT1 input mux
PIO1_4/ADC1_10	-	33	51	SCT1 input mux
PIO1_5/ADC1_11	-	34	52	SCT1 input mux
PIO1_6/ACMP_I2	-	46	73	SCT0 input mux
PIO1_7/ACMP3_I4	-	51	81	SCT0 input mux
PIO1_11	-	38	58	SCT3 input mux
				SCTIPU input SAMPLE_IN_A2
PIO1_12	-	-	9	SCT0 input mux
PIO1_13	-	-	11	SCT0 input mux
PIO1_15	-	-	12	SCT1 input mux
PIO1_16	-	-	18	SCT1 input mux
PIO1_18	-	-	25	SCT2 input mux
PIO1_19	-	-	29	SCT2 input mux
PIO1_21	-	-	37	SCT3 input mux
PIO1_22	-	-	38	SCT3 input mux
PIO1_26	-	-	48	SCTIPU input SAMPLE_IN_A3
PIO1_27	-	-	50	FREQMEAS

7.1 How to read this chapter

The IOCON block is identical for all LPC15xx parts. Registers for pins that are not available on a specific package are reserved.

Table 96. Available pin configuration registers

Package	GPIO Port 0	GPIO Port 1	GPIO Port 2
	Pins/configuration registers available		
LQFP48	PIO0_0 to PIO0_29	-	-
LQFP64	PIO0_0 to PIO0_31	PIO1_0 to PIO1_11	-
LQFP100	PIO0_0 to PIO0_31	PIO1_0 to PIO1_31	PIO2_0 to PIO2_11

7.2 Features

The following electrical properties are configurable for each pin:

- Pull-up/pull-down resistor
- Open-drain mode
- Hysteresis
- Digital filter with programmable time constant
- 10 ns, digital glitch filter on selected pins

The true open-drain pins PIO0_22 and PIO0_23 can be configured for different I2C-bus speeds.

The switch matrix configures a pin for its analog function automatically, when the analog function is enabled in the PINENABLEn registers.

7.3 Basic configuration

Enable the clock to the IOCON in the SYSAHBCLKCTRL0 register (Table 50, bit 13). Once the pins are configured, you can disable the IOCON clock to conserve power.

Each pin has a programmable digital input filter. The base clock for the filter is the output of the IOCONCLKDIV clock divider in the SYSCON block (see Table 54). The base clock can be divided individually for each pin to create the glitch filter constant in each digital pin configuration register.

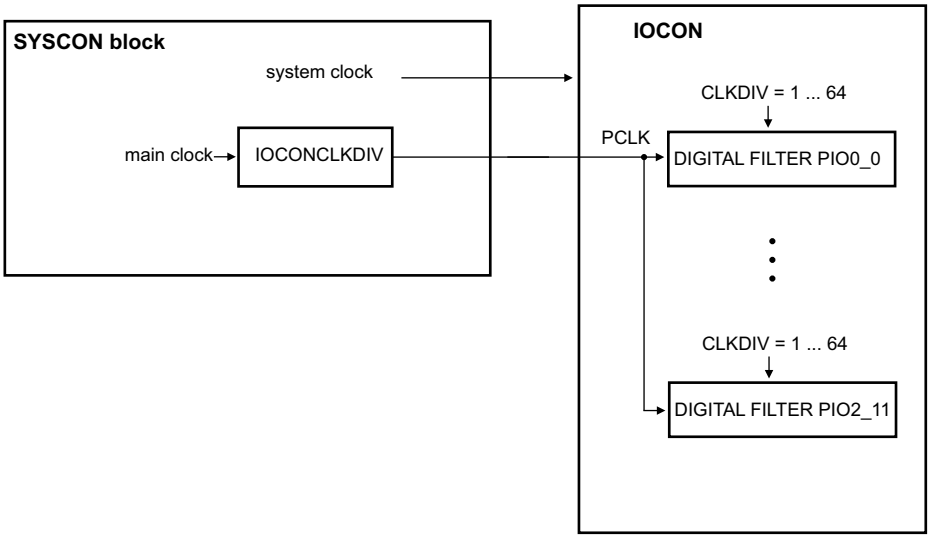


Fig 9. IOCON clocking

7.4 General description

7.4.1 Pin configuration

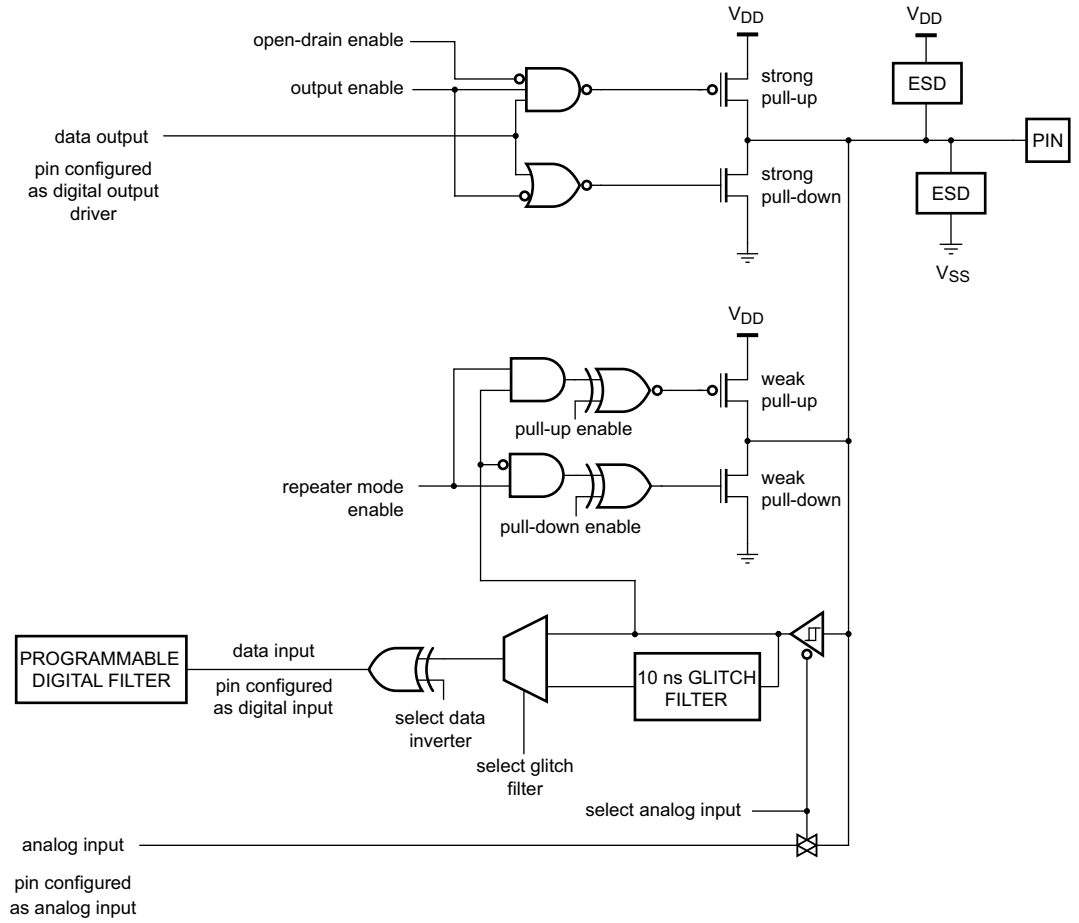


Fig 10. Pin configuration

7.4.2 Pin function

The pin function is determined entirely through the switch matrix. By default one GPIO function is assigned to each pin. The switch matrix can assign all functions from the movable function table to any pin in the IOCON block or enable a special function like an analog input on a specific pin.

Related links:

Section 8.4.2 “Movable functions”

7.4.3 Pin mode

The MODE bit in the IOCON register allows enabling or disabling an on-chip pull-up resistor for each pin. By default all pull-up resistors are enabled except for the I^2C-bus pins PIO0_22 and PIO0_23, which do not have a programmable pull-up resistor.

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

7.4.4 Open-drain mode

An open-drain mode can be enabled for all digital I/O pins that are not the I2C-bus pins. This mode is not a true open-drain mode. The input cannot be pulled up above V_{DD}.

Remark: As opposed to the true open-drain I2C-bus pins, digital pins with configurable open-drain mode are **not** 5 V tolerant when $V_{DD} = 0$.

7.4.5 Analog mode

The switch matrix automatically configures the pin in analog mode whenever an analog input or output is selected as the pin's function through the switch matrix PINENABLE registers.

When using a pin in analog mode, disable the pull-up and pull-down resistors.

7.4.6 I^2C-bus mode

The I^2C-bus pins PIO0_22 and PIO0_23 can be programmed to support a true open-drain mode independently of whether the I2C function is selected or another digital function. If the I^2C function is selected, all three I^2C modes, Standard mode, Fast-mode, and Fast-mode plus, are supported. A digital glitch filter can be configured for all functions. Pins PIO0_22 and PIO0_23 operate as high-current sink drivers (20 mA) independently of the programmed function.

7.4.7 Input glitch filter

Selected pins (see Section 7.5.1 "Digital pin control registers with glitch filter on port 0") provide the option of turning on or off a 10 ns input glitch filter. The glitch filter is turned on by default. The $\overline{\text{RESET}}$ pin has a 20 ns glitch filter (not configurable).

7.4.8 Programmable digital filter

All GPIO pins are equipped with a programmable digital filter. The filter rejects input pulses with a selectable duration of shorter than one, two, or three cycles of a filter clock (S_MODE = 1, 2, or 3). For each individual pin, the filter clock is derived from the main clock using the IOCONCLKDIV register divided by the CLKDIV value (PCLKn). The filter can also be bypassed entirely.

Any input pulses of duration T_{pulse} of either polarity will be rejected if:
$T_{pulse} < T_{PCLKn} \times S\_MODE$

Input pulses of one filter clock cycle longer may also be rejected:

$T_{pulse} = T_{PCLKn} \times (S\_MODE + 1)$

Remark: The filtering effect is accomplished by requiring that the input signal be stable for (S_MODE +1) successive edges of the filter clock before being passed on to the chip. Enabling the filter results in delaying the signal to the internal logic and should be done only if specifically required by an application. For high-speed or time critical functions ensure that the filter is bypassed.

If the delay of the input signal must be minimized, select a faster PCLK and a higher sample mode (S_MODE) to minimize the effect of the potential extra clock cycle.

If the sensitivity to noise spikes must be minimized, select a slower PCLK and lower sample mode.

Related registers and links:

Table 54 “IOCON glitch filter clock divider register (IOCONCLKDIV, address 0x4007 40D4) bit description”

7.5 Register description

Each port pin PIOm_n has one IOCON register assigned to control the pin’s electrical characteristics.

Table 97. Register overview: I/O configuration (base address 0x400F 8000)

Name	Access	Address offset	Description	Reset value	Reference
PIO0_0 to PIO0_17	R/W	0x000 to 0x044	Digital I/O control for port 0 pins PIO0_0 to PIO0_17. With analog function and glitch filter.	0x90	Table 99
PIO0_18 to PIO0_21	R/W	0x048 to 0x054	Digital I/O control for port 0 pins PIO0_18 to PIO0_21. Without glitch filter.	0x90	Table 100
PIO0_22	R/W	0x058	I/O control for open-drain pin PIO0_22. This pin is used for the I2C-bus SCL function.	0x90	Table 101
PIO0_23	R/W	0x05C	I/O control for open-drain pin PIO0_23. This pin is used for the I2C-bus SDA function.	0x90	Table 101
PIO0_24	R/W	0x060	Digital I/O control for port 0 pins PIO0_24. Without glitch filter.	0x90	Table 100
PIO0_25 to PIO0_31	R/W	0x064 to 0x07C	Digital I/O control for port 0 pins PIO0_25 to PIO0_31.With analog function and glitch filter.	0x90	Table 99
PIO1_0 to PIO1_10	R/W	0x080 to 0x0A8	Digital I/O control for port 1 pins PIO1_0 to PIO1_10. With analog function and glitch filter.	0x90	Table 102
PIO1_11 to PIO1_31	R/W	0x0AC to 0x0FC	Digital I/O control for port 1 pins PIO1_11to PIO1_31.Without glitch filter.	0x90	Table 103
PIO2_0 to PIO2_13	R/W	0x100 to 0x134	Digital I/O control for port 2 pins PIO2_0 to PIO2_13. Without glitch filter.	0x90	Table 104

Table 98. Digital I/O configuration register types

Name	Address offset	True open-drain	Analog	Glitch filter on/off	Digital filter	High-drive output	Reference
PIO0_0	0x000	no	yes	yes	yes	no	Table 99
PIO0_1	0x004	no	yes	yes	yes	no	Table 99
PIO0_2	0x008	no	yes	yes	yes	no	Table 99

Table 98. Digital I/O configuration register types

Name	Address offset	True open-drain	Analog	Glitch filter on/off	Digital filter	High-drive output	Reference
PIO0_3	0x00C	no	yes	yes	yes	no	Table 99
PIO0_4	0x010	no	yes	yes	yes	no	Table 99
PIO0_5	0x014	no	yes	yes	yes	no	Table 99
PIO0_6	0x018	no	yes	yes	yes	no	Table 99
PIO0_7	0x01C	no	yes	yes	yes	no	Table 99
PIO0_8	0x020	no	yes	yes	yes	no	Table 99
PIO0_9	0x024	no	yes	yes	yes	no	Table 99
PIO0_10	0x028	no	yes	yes	yes	no	Table 99
PIO0_11	0x02C	no	yes	yes	yes	no	Table 99
PIO0_12	0x030	no	yes	yes	yes	no	Table 99
PIO0_13	0x034	no	yes	yes	yes	no	Table 99
PIO0_14	0x038	no	yes	yes	yes	no	Table 99
PIO0_15	0x03C	no	yes	yes	yes	no	Table 99
PIO0_16	0x040	no	yes	yes	yes	no	Table 99
PIO0_17	0x044	no	yes	yes	yes	no	Table 99
PIO0_18	0x048	no	no	no	yes	no	Table 100
PIO0_19	0x04C	no	no	no	yes	no	Table 100
PIO0_20	0x050	no	no	no	yes	no	Table 100
PIO0_21	0x054	no	no	no	yes	no	Table 100
PIO0_22	0x058	yes	no	no	yes	no	Table 101
PIO0_23	0x05C	yes	no	no	yes	no	Table 101
PIO0_24	0x060	no	no	no	yes	yes	Table 100
PIO0_25	0x064	no	yes	yes	yes	no	Table 99
PIO0_26	0x068	no	yes	yes	yes	no	Table 102
PIO0_27	0x06C	no	yes	yes	yes	no	Table 102
PIO0_28	0x070	no	yes	yes	yes	no	Table 102
PIO0_29	0x074	no	yes	yes	yes	no	Table 102
PIO0_30	0x078	no	yes	yes	yes	no	Table 102
PIO0_31	0x07C	no	yes	yes	yes	no	Table 102
PIO1_0	0x080	no	yes	yes	yes	no	Table 102
PIO1_1	0x084	no	yes	yes	yes	no	Table 102
PIO1_2	0x088	no	yes	yes	yes	no	Table 102
PIO1_3	0x08C	no	yes	yes	yes	no	Table 102
PIO1_4	0x090	no	yes	yes	yes	no	Table 102
PIO1_5	0x094	no	yes	yes	yes	no	Table 102
PIO1_6	0x098	no	yes	yes	yes	no	Table 102
PIO1_7	0x09C	no	yes	yes	yes	no	Table 102
PIO1_8	0x0A0	no	yes	yes	yes	no	Table 102
PIO1_9	0x0A4	no	yes	yes	yes	no	Table 102
PIO1_10	0x0A8	no	yes	yes	yes	no	Table 102

Table 98. Digital I/O configuration register types

Name	Address offset	True open-drain	Analog	Glitch filter on/off	Digital filter	High-drive output	Reference
PIO1_11	0x0AC	no	no	no	yes	no	Table 103
PIO1_12	0x0B0	no	no	no	yes	no	Table 103
PIO1_13	0x0B4	no	no	no	yes	no	Table 103
PIO1_14	0x0B8	no	no	no	yes	no	Table 103
PIO1_15	0x0BC	no	no	no	yes	no	Table 103
PIO1_16	0x0C0	no	no	no	yes	no	Table 103
PIO1_17	0x0C4	no	no	no	yes	no	Table 103
PIO1_18	0x0C8	no	no	no	yes	no	Table 103
PIO1_19	0x0CC	no	no	no	yes	no	Table 103
PIO1_20	0x0D0	no	no	no	yes	no	Table 103
PIO1_21	0x0D4	no	no	no	yes	no	Table 103
PIO1_22	0x0D8	no	no	no	yes	no	Table 103
PIO1_23	0x0DC	no	no	no	yes	no	Table 103
PIO1_24	0x0E0	no	no	no	yes	no	Table 103
PIO1_25	0x0E4	no	no	no	yes	no	Table 103
PIO1_26	0x0E8	no	no	no	yes	no	Table 103
PIO1_27	0x0EC	no	no	no	yes	no	Table 103
PIO1_28	0x0F0	no	no	no	yes	no	Table 103
PIO1_29	0x0F4	no	no	no	yes	no	Table 103
PIO1_30	0x0F8	no	no	no	yes	no	Table 103
PIO1_31	0x0FC	no	no	no	yes	no	Table 103
PIO2_0	0x100	no	yes	yes	yes	no	Table 104
PIO2_1	0x104	no	yes	yes	yes	no	Table 104
PIO2_2	0x108	no	yes	yes	yes	no	Table 104
PIO2_3	0x10C	no	no	no	yes	no	Table 104
PIO2_4	0x110	no	no	no	yes	no	Table 104
PIO2_5	0x114	no	no	no	yes	no	Table 104
PIO2_6	0x118	no	no	no	yes	no	Table 104
PIO2_7	0x11C	no	no	no	yes	no	Table 104
PIO2_8	0x120	no	no	no	yes	no	Table 104
PIO2_9	0x124	no	no	no	yes	no	Table 104
PIO2_10	0x128	no	no	no	yes	no	Table 104
PIO2_11	0x12C	no	no	no	yes	no	Table 104
PIO2_12	0x130	no	no	no	yes	no	Table 104
PIO2_13	0x134	no	no	no	yes	no	Table 104

7.5.1 Digital pin control registers with glitch filter on port 0

The digital pin control registers control the digital properties of all pins except the pins with true open-drain I2C-bus functions. The pin's function is determined by the switch matrix.

If any of the pins is connected to an analog function through the switch matrix PINENABLE registers, the digital portion of the pin is disconnected.

For each pin controlled by these registers, the digital glitch filter is enabled.

Table 99. Digital pin control registers PIO0_0 to PIO0_17 register (PIO0_[0:17], address 0x400F 8000 (PIO0_0) to 0x400F 8044 (PIO0_17) and 0x400F 8064 (PIO0_25) to 0x400F 807C (PIO0_31)) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Only write 0 to these bits.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	-	-	Reserved.	1
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter enabled.	
		1	Filter disabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	

Table 99. Digital pin control registers PIO0_0 to PIO0_17 register (PIO0_[0:17], address 0x400F 8000 (PIO0_0) to 0x400F 8044 (PIO0_17) and 0x400F 8064 (PIO0_25) to 0x400F 807C (PIO0_31)) bit description

Bit	Symbol	Value	Description	Reset value
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	0

7.5.2 Digital pin control registers without glitch filter on port 0

The digital pin control registers control the digital properties of all pins except the pins with true open-drain I2C-bus functions. The pin's function is determined by the switch matrix.

If any of the pins is connected to an analog function through the switch matrix PINENABLE registers, the digital portion of the pin is disconnected.

Pins controlled by these registers do not have a glitch filter.

Table 100. Digital pin control registers PIO0_18 to PIO0_21 register (PIO0_[18:21], address 0x400F 8048 (PIO0_18) to 0x400F 8054 (PIO0_21), and 0x400F 8060 (PIO0_24)) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Only write 0 to these bits.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
9:7	-	-	Reserved.	0x1

Table 100. Digital pin control registers PIO0_18 to PIO0_21 register (PIO0_[18:21], address 0x400F 8048 (PIO0_18) to 0x400F 8054 (PIO0_21), and 0x400F 8060 (PIO0_24)) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	0

7.5.3 Digital pin control registers for open-drain pins PIO0_22/23 on port 0

Table 101. Digital open-drain pin control registers (PIO0_[22:23], address 0x400F 805C (PIO0_22) to 0x400F 8060 (PIO0_23)) bit description

Bit	Symbol	Value	Description	Reset value
5:0	-		Reserved. Only write 0 to these bits.	0x10
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	-		Reserved.	1

Table 101. Digital open-drain pin control registers (PIO0_[22:23], address 0x400F 805C (PIO0_22) to 0x400F 8060 (PIO0_23)) bit description

Bit	Symbol	Value	Description	Reset value
9:8	I2CMODE		Selects I2C mode. Select Standard mode (I2CMODE = 00, default) or Standard I/O functionality (I2CMODE = 01) if the pin function is GPIO (FUNC = 000).	0
		0x0	Standard mode/ Fast-mode I2C.	
		0x1	Standard digital I/O functionality	
		0x2	Fast-mode Plus I2C	
		0x3	Reserved.	
10	-	-	Reserved.	-
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	-

7.5.4 Digital pin control registers with glitch filter on port 1

The digital pin control registers control the digital properties of all pins except the pins with true open-drain I2C-bus functions. The pin's function is determined by the switch matrix.

If any of the pins is connected to an analog function through the switch matrix PINENABLE registers, the digital portion of the pin is disconnected.

For each pin controlled by these registers, the digital glitch filter is enabled.

Table 102. Digital pin control registers PIO1_0 to PIO1_10 register (PIO1_[0:10], address 0x400F 8080 (PIO1_0) to 0x400F 80A8 (PIO1_10)) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Only write 0 to these bits.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	-	-	Reserved.	1
8	FILTR		Selects 10 ns input glitch filter.	0
		0	Filter enabled.	
		1	Filter disabled.	
9	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	

Table 102. Digital pin control registers PIO1_0 to PIO1_10 register (PIO1_[0:10], address 0x400F 8080 (PIO1_0) to 0x400F 80A8 (PIO1_10)) bit description

Bit	Symbol	Value	Description	Reset value
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	0

7.5.5 Digital pin control registers without glitch filter on port 1

The digital pin control registers control the digital properties of all pins except the pins with true open-drain I2C-bus functions. The pin's function is determined by the switch matrix.

If any of the pins is connected to an analog function through the switch matrix PINENABLE registers, the digital portion of the pin is disconnected.

Pins controlled by these registers do not have a glitch filter.

Table 103. Digital pin control registers PIO1_11 to PIO1_31 register (PIO1_[11:31], address 0x400F 80AC (PIO1_11) to 0x400F 80FC (PIO1_31)) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Only write 0 to these bits.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	-	-	Reserved.	1
9:8	-	-	Reserved.	0

Table 103. Digital pin control registers PIO1_11 to PIO1_31 register (PIO1_[11:31], address 0x400F 80AC (PIO1_11) to 0x400F 80FC (PIO1_31)) bit description

Bit	Symbol	Value	Description	Reset value
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	0

7.5.6 Digital pin control registers without glitch filter on port 2

The digital pin control registers control the digital properties of all pins except the pins with true open-drain I2C-bus functions. The pin's function is determined by the switch matrix.

If any of the pins is connected to an analog function through the switch matrix PINENABLE registers, the digital portion of the pin is disconnected.

Pins controlled by these registers do not have a glitch filter.

Table 104. Digital pin control registers PIO2_0 to PIO2_13 register (PIO2_[0:13], address 0x400F 8100 (PIO2_0) to 0x400F 8124 (PIO2_13)) bit description

Bit	Symbol	Value	Description	Reset value
2:0	-		Reserved. Only write 0 to these bits.	0
4:3	MODE		Selects function mode (on-chip pull-up/pull-down resistor control).	0x2
		0x0	Inactive (no pull-down/pull-up resistor enabled).	
		0x1	Pull-down resistor enabled.	
		0x2	Pull-up resistor enabled.	
		0x3	Repeater mode.	

Table 104. Digital pin control registers PIO2_0 to PIO2_13 register (PIO2_[0:13], address 0x400F 8100 (PIO2_0) to 0x400F 8124 (PIO2_13)) bit description

Bit	Symbol	Value	Description	Reset value
5	HYS		Hysteresis.	0
		0	Disable.	
		1	Enable.	
6	INV		Invert input	0
		0	Input not inverted (HIGH on pin reads as 1; LOW on pin reads as 0).	
		1	Input inverted (HIGH on pin reads as 0, LOW on pin reads as 1).	
7	-	-	Reserved.	1
9:8	-	-	Reserved.	0
10	OD		Open-drain mode.	0
		0	Disable.	
		1	Open-drain mode enabled. **Remark:** This is not a true open-drain mode.	
12:11	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	
15:13	CLKDIV		Select peripheral clock divider for input filter sampling clock IOCONCLKDIV. Value 0x7 is reserved.	0
		0x0	PCLK.	
		0x1	PCLK/2.	
		0x2	PCLK/4.	
		0x3	PCLK/8.	
		0x4	PCLK/16.	
		0x5	PCLK/32.	
		0x6	PCLK/64.	
31:16	-	-	Reserved.	0

8.1 How to read this chapter

The switch matrix is available on all parts.

8.2 Features

- Flexible assignment of digital peripheral functions to pins
- Enable/disable of analog functions

8.3 Basic configuration

Once configured, no clocks are needed for the switch matrix to function. The system clock is needed only to write to or read from the pin assignment registers. After the switch matrix is configured, disable the clock to the switch matrix block in the SYSAHBCLKCTRL register.

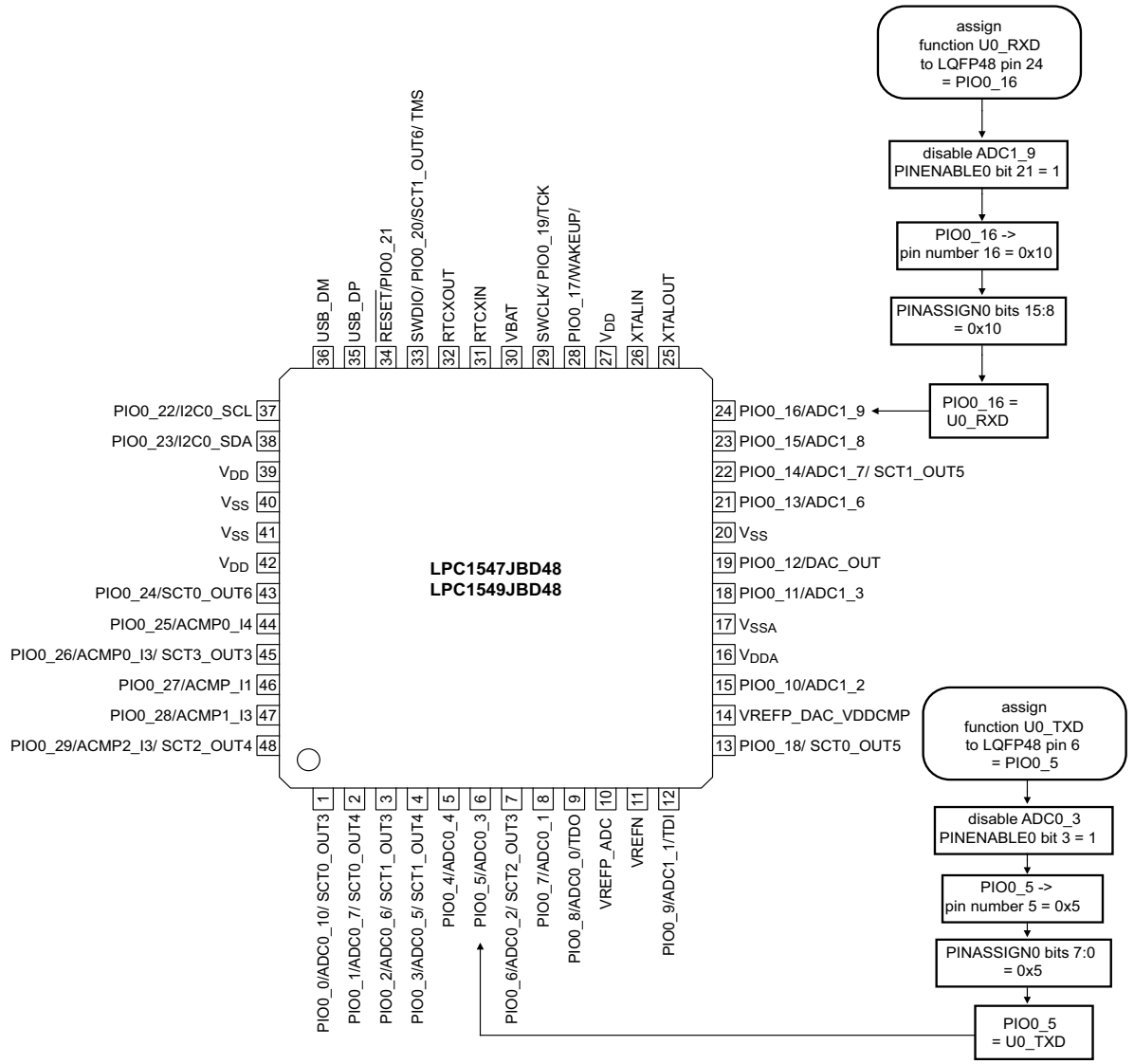
Before activating a peripheral or enabling its interrupt, use the switch matrix to connect the peripheral to external pins.

The serial wire debug pins SWCLK and SWDIO are assigned by default to pins PIO0_19 and PIO0_20. If the user code disables the SWD functions through the switch matrix to use the pins for other functions, the SWD port is disabled.

Remark: For the purpose of programming the pin functions through the switch matrix, every programmable pin (all pins except some special function pins, the power, and the ground pins) is identified in a package-independent way by its GPIO port pin number.

Remark: The switch matrix is only reset by a POR or BOD reset. A hardware reset via the $\overline{\text{RESET}}$ pin or a watchdog timer reset do not reset the switch matrix. Therefore, peripheral functions remain connected to pins through the hardware or watchdog reset and the pins remain input or output as defined by the switch matrix and assume the default state as defined by the peripheral connected to them. See also Section 3.7.1.

8.3.1 Connect an internal signal to a package pin



A pin is identified for the purpose of programming the switch matrix by its default GPIO port pin.

Fig 11. Example: Connect function U0_RXD and U0_TXD to pins

The switch matrix connects all internal signals listed in the table of movable functions through the pin assignment registers to external pins on the package. External pins are identified by their default GPIO pin number PIO0_n. Follow these steps to connect an internal signal FUNC to an external pin. An example of a movable function is the UART transmit signal TXD:

1. Find the function FUNC in the list of movable function in Table 105 or in the data sheet.
2. Use the LPC15xx data sheet to decide which pin x on the LPC15xx package to connect FUNC to.

3. Use the pin description table to find the default GPIO function PIO0_n assigned to package pin x. m is the pin number (0 to 31 for port 0, 32 to 63 for port 1, 64 to 75 for port 2).
4. Locate the pin assignment register for the function FUNC in the switch matrix register description.
5. Disable any special functions on pin PIO0_n in the PINENABLE0 register.
6. Program the pin number m into the bits assigned to FUNC.

FUNC is now connected to pin x on the package.

8.3.2 Enable an analog input or other special function

The switch matrix enables functions that can only be assigned to one pin. Examples are analog inputs, all GPIO pins, and the debug SWD pins.

- If you want to assign a GPIO pin, disable any special function available on this pin in the PINENABLE0 register and do not assign any movable function to it.

  By default, all pins except pins hosting the $\overline{\text{RESET}}$ and serial wire functions are assigned to GPIO.
- For all other functions that are not in the table of movable functions, do the following:
  a. Locate the function in the pin description table in the data sheet. This shows the package pin for this function.
  b. Enable the function in the PINENABLE0/1 registers. All other possible functions on this pins are now disabled.

8.4 General description

The switch matrix connects internal signals (functions) to external pins. Functions are signals coming from or going to a single pin on the package and coming from or going to an on-chip peripheral block. Examples of functions are the GPIOs, the UART transmit output (TXD), or the clock output CLKOUT. Many peripherals have several functions that must be connected to external pins.

Most functions can be assigned through the switch matrix to any external pin that is not a power or ground pin. These functions are called movable functions.

Analog inputs and outputs as well as open-drain I2C functions and a few digital functions such as the SCT outputs are assigned to one particular external pin with the appropriate electrical characteristics and have to be enabled on that pin. These functions are called fixed-pin functions. If a fixed-pin function is not enabled on the assigned pin, it can be replaced by any other movable function.

GPIOs are fixed-pin functions. Each GPIO is assigned to one and only one external pin. By default, all external pins have the GPIO function assigned. External pins are therefore identified by their fixed-pin GPIO function.

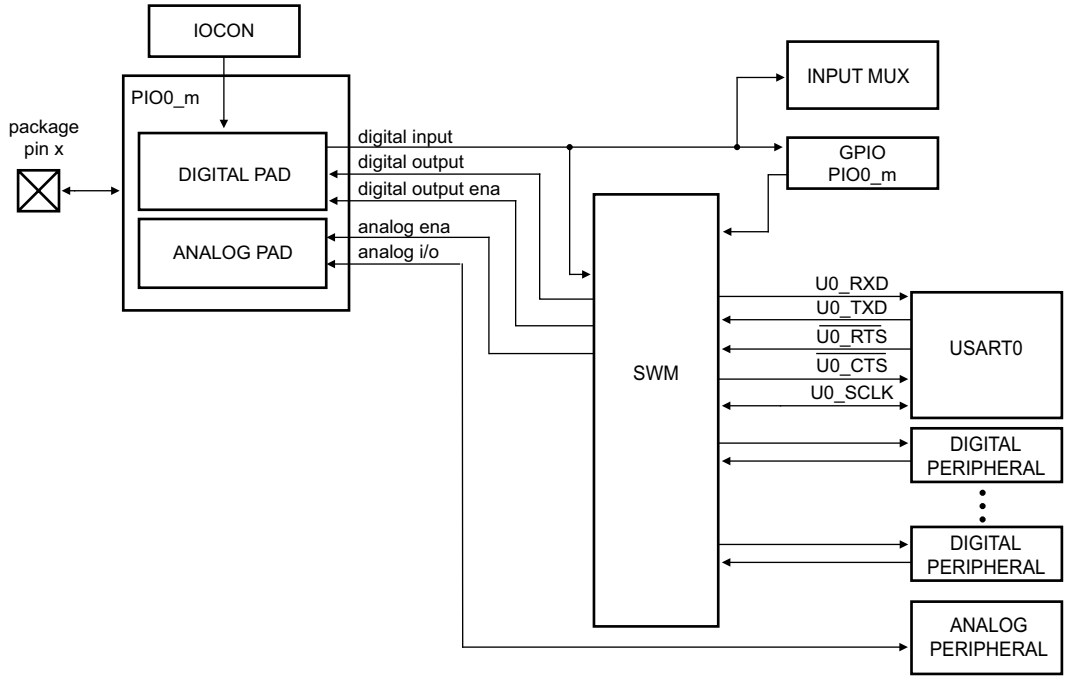


Fig 12. Switch matrix block diagram

8.4.1 Switch matrix register interface

The switch matrix consists of two blocks of pin-assignment registers PINASSIGN and PINENABLE. Every function has an assigned field (1-bit or 8-bit wide) within this bank of registers where you can program the external pin - identified by its GPIO function - you want the function to connect to.

GPIOs range from PIO0_0 to PIO2_11 and, for assignment through the pin-assignment registers, are consecutively numbered 0 to 75 (32 pins for port 0 and 1, 12 pins for port 2).

There are two types of functions which must be assigned to port pins in different ways:

1. **Movable functions** (PINASSIGN0 to 8):

   All movable functions are digital functions. Assign movable functions to pin numbers through the 8 bits of the PINASSIGN register associated with this function. Once the function is assigned a pin PIO0_n, it is connected through this pin to a physical pin on the package.

   Remark: You are allowed to assign only one digital output function to an external pin at any given time.

   Remark: You can assign more than one digital input function to one external pin.

2. **Fixed-pin functions** (PINENABLE0/1):

   Some functions require pins with special characteristics and cannot be moved to other physical pins. Hence these functions are mapped to a fixed port pin. Examples of fixed-pin functions are the oscillator pins or comparator inputs.

   Each fixed-pin function is associated with one bit in the PINENABLE0 register which selects or deselects the function.

- If a fixed-pin function is deselected, any movable function can be assigned to its port and pin.
- If a fixed-pin function is deselected and no movable function is assigned to this pin, the pin is GPIO.
- On reset, all fixed-pin functions are deselected.
- If a fixed-pin function is selected, its assigned pin can not be used for any other function.

8.4.2 Movable functions

Table 105. Movable functions

Function name	Type	Description	SWM Pin assign register	Reference
U0_TXD	O	Transmitter output for USART0.	PINASSIGN0	Table 107
U0_RXD	I	Receiver input for USART0.	PINASSIGN0	Table 107
$\overline{\text{U0\_RTS}}$	O	Request To Send output for USART0.	PINASSIGN0	Table 107
$\overline{\text{U0\_CTS}}$	I	Clear To Send input for USART0.	PINASSIGN0	Table 107
U0_SCLK	I/O	Serial clock input/output for USART0 in synchronous mode.	PINASSIGN1	Table 108
U1_TXD	O	Transmitter output for USART1.	PINASSIGN1	Table 108
U1_RXD	I	Receiver input for USART1.	PINASSIGN1	Table 108
$\overline{\text{U1\_RTS}}$	O	Request To Send output for USART1.	PINASSIGN1	Table 108
$\overline{\text{U1\_CTS}}$	I	Clear To Send input for USART1.	PINASSIGN2	Table 109
U1_SCLK	I/O	Serial clock input/output for USART1 in synchronous mode.	PINASSIGN2	Table 109
U2_TXD	O	Transmitter output for USART2.	PINASSIGN2	Table 109
U2_RXD	I	Receiver input for USART2.	PINASSIGN2	Table 109
U2_SCLK	I/O	Serial clock input/output for USART1 in synchronous mode.	PINASSIGN3	Table 110
SPI0_SCK	I/O	Serial clock for SPI0.	PINASSIGN3	Table 110
SPI0_MOSI	I/O	Master Out Slave In for SPI0.	PINASSIGN3	Table 110
SPI0_MISO	I/O	Master In Slave Out for SPI0.	PINASSIGN3	Table 110
SPI0_SSEL0	I/O	Slave select 0 for SPI0.	PINASSIGN4	Table 111
SPI0_SSEL1	I/O	Slave select 1 for SPI0.	PINASSIGN4	Table 111
SPI0_SSEL2	I/O	Slave select 2 for SPI0.	PINASSIGN4	Table 111
SPI0_SSEL3	I/O	Slave select 3 for SPI0.	PINASSIGN4	Table 111
SPI1_SCK	I/O	Serial clock for SPI1.	PINASSIGN5	Table 112
SPI1_MOSI	I/O	Master Out Slave In for SPI1.	PINASSIGN5	Table 112
SPI1_MISO	I/O	Master In Slave Out for SPI1.	PINASSIGN5	Table 112
SPI1_SSEL0	I/O	Slave select 0 for SPI1.	PINASSIGN5	Table 112
SPI1_SSEL1	I/O	Slave select 1 for SPI1.	PINASSIGN6	Table 113
CAN0_TD	O	CAN0 transmit.	PINASSIGN6	Table 113
CAN0_RD	I	CAN0 receive.	PINASSIGN6	Table 113
USB_VBUS	I	USB VBUS.	PINASSIGN7	Table 114
SCT0_OUT0	O	SCT0 output 0.	PINASSIGN7	Table 114
SCT0_OUT1	O	SCT0 output 1.	PINASSIGN7	Table 114
SCT0_OUT2	O	SCT0 output 2.	PINASSIGN7	Table 114
SCT1_OUT0	O	SCT1 output 0.	PINASSIGN8	Table 115
SCT1_OUT1	O	SCT1 output 1.	PINASSIGN8	Table 115
SCT1_OUT2	O	SCT1 output 2.	PINASSIGN8	Table 115
SCT2_OUT0	O	SCT2 output 0.	PINASSIGN8	Table 115

Table 105. Movable functions *…continued*

Function name	Type	Description	SWM Pin assign register	Reference
SCT2_OUT1	O	SCT2 output 1.	PINASSIGN9	Table 116
SCT2_OUT2	O	SCT2 output 2.	PINASSIGN9	Table 116
SCT3_OUT0	O	SCT3 output 0.	PINASSIGN9	Table 116
SCT3_OUT1	O	SCT3 output 1.	PINASSIGN9	Table 116
SCT3_OUT2	O	SCT3 output 2.	PINASSIGN10	Table 117
SCT_ABORT0	I	SCT abort 0.	PINASSIGN10	Table 117
SCT_ABORT1	I	SCT abort 1.	PINASSIGN10	Table 117
ADC0_PINTRIG0	I	ADC0 external pin trigger input 0.	PINASSIGN10	Table 117
ADC0_PINTRIG1	I	ADC0 external pin trigger input 1.	PINASSIGN11	Table 118
ADC1_PINTRIG0	I	ADC1 external pin trigger input 0.	PINASSIGN11	Table 118
ADC1_PINTRIG1	I	ADC1 external pin trigger input 1.	PINASSIGN11	Table 118
DAC_PINTRIG	I	DAC external pin trigger input.	PINASSIGN11	Table 118
DAC_SHUTOFF	I	DAC shut-off external input.	PINASSIGN12	Table 119
ACMP0_O	O	Analog comparator 0 output.	PINASSIGN12	Table 119
ACMP1_O	O	Analog comparator 1 output.	PINASSIGN12	Table 119
ACMP2_O	O	Analog comparator 2 output.	PINASSIGN12	Table 119
ACMP3_O	O	Analog comparator 3 output.	PINASSIGN13	Table 120
CLKOUT	O	Clock output.	PINASSIGN13	Table 120
ROSC	O	Analog comparator ring oscillator output.	PINASSIGN13	Table 120
ROSC_RESET	I	Analog comparator ring oscillator reset.	PINASSIGN13	Table 120
USB_FTOGGLE	O	USB frame toggle. Do not assign this function to a pin until a USB device is connected and the first SOF interrupt has been received by the device.	PINASSIGN14	Table 121
QEI_PHA	I	QEI phase A input.	PINASSIGN14	Table 121
QEI_PHB	I	QEI phase B input.	PINASSIGN14	Table 121
QEI_IDX	I	QEI index input.	PINASSIGN14	Table 121
GPIO_INT_BMAT	O	Output of the pattern match engine.	PINASSIGN15	Table 122
SWO	O	Serial wire output.	PINASSIGN15	Table 122

8.5 Register description

Table 106. Register overview: Switch matrix (base address 0x4003 8000)

Name	Access	Offset	Description	Reset value	Reference
PINASSIGN0	R/W	0x000	Pin assign register 0. Assign movable functions U0_TXD, U0_RXD, U0_RTS, U0_CTS.	0xFFFF FFFF	Table 107
PINASSIGN1	R/W	0x004	Pin assign register 1. Assign movable functions U0_SCLK, U1_TXD, U1_RXD, U1_RTS.	0xFFFF FFFF	Table 108
PINASSIGN2	R/W	0x008	Pin assign register 2. Assign movable functions U1_CTS, U1_SCLK, U2_TXD, U2_RXD.	0xFFFF FFFF	Table 109

Table 106. Register overview: Switch matrix (base address 0x4003 8000) *…continued*

Name	Access	Offset	Description	Reset value	Reference
PINASSIGN3	R/W	0x00C	Pin assign register 3. Assign movable function U2_SCLK, SPI0_SCK, SPI0_MOSI, SPI0_MISO.	0xFFFF FFFF	Table 110
PINASSIGN4	R/W	0x010	Pin assign register 4. Assign movable functions SPI0_SSEL0, SPI0_SSEL1, SPI0_SSEL2, SPI0_SSEL3.	0xFFFF FFFF	Table 111
PINASSIGN5	R/W	0x014	Pin assign register 5. Assign movable functions SPI1_SCK, SPI1_MOSI, SPI1_MISO, SPI1_SSEL0	0xFFFF FFFF	Table 112
PINASSIGN6	R/W	0x018	Pin assign register 6. Assign movable functions SPI1_SSEL1, CAN0_TD, CAN0_RD.	0xFFFF FFFF	Table 113
PINASSIGN7	R/W	0x01C	Pin assign register 7. Assign movable functions USB_VBUS, SCT0_OUT0, SCT0_OUT1, SCT0_OUT2	0xFFFF FFFF	Table 114
PINASSIGN8	R/W	0x020	Pin assign register 8. Assign movable functions SCT1_OUT0, SCT1_OUT1, SCT1_OUT2, SCT2_OUT0	0xFFFF FFFF	Table 115
PINASSIGN9	R/W	0x024	Pin assign register 9. Assign movable functions SCT2_OUT1, SCT2_OUT2, SCT3_OUT0, SCT3_OUT1	0xFFFF FFFF	Table 116
PINASSIGN10	R/W	0x028	Pin assign register 10. Assign movable functions SCT3_OUT2, SCT_ABORT0, SCT_ABORT1, ADC0_PINTRIG0	0xFFFF FFFF	Table 117
PINASSIGN11	R/W	0x02C	Pin assign register 11. Assign movable functions ADC0_PINTRIG1, ADC1_PINTRIG0, ADC1_PINTRIG1, DAC_PINTRIG	0xFFFF FFFF	Table 118
PINASSIGN12	R/W	0x030	Pin assign register 12. Assign movable functions DAC_SHUTOFF, ACMP0_O, ACMP1_O, ACMP2_O	0xFFFF FFFF	Table 119
PINASSIGN13	R/W	0x034	Pin assign register 13. Assign movable functions ACMP3_O, CLKOUT, ROSC, ROSC_RESET	0xFFFF FFFF	Table 120
PINASSIGN14	R/W	0x038	Pin assign register 14. Assign movable functions USB_FTOGGLE, QEI_PHA, QEI_PHB, QEI_IDX	0xFFFF FFFF	Table 121
PINASSIGN15	R/W	0x03C	Pin assign register 15. Assign movable functions GPIO_INT_BMAT, SWO	0xFFFF FFFF	Table 122
-	-	-	Reserved	-	
PINENABLE0	R/W	0x1C0	Pin enable register 0. Enables fixed-pin functions for ADC0, ADC1, and analog comparator.	0xFFFF FFFF	Table 123
PINENABLE1	R/W	0x1C4	Pin enable register 0. Enables fixed-pin functions for analog comparator, I2C, SCT outputs, RESET, serial wire debug.	0xFF1F FFFF	Table 124

8.5.1 PINASSIGN0

Table 107. Pin assign register 0 (PINASSIGN0, address 0x4003 8000) bit description

Bit	Symbol	Description	Reset value
7:0	UART0_TXD_O	UART0_TXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	UART0_RXD_I	UART0_RXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	UART0_RTS_O	UART0_RTS function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	UART0_CTS_I	UART0_CTS function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.2 PINASSIGN1

Table 108. Pin assign register 1 (PINASSIGN1, address 0x4003 8004) bit description

Bit	Symbol	Description	Reset value
7:0	UART0_SCLK_IO	UART0_SCLK function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	UART1_TXD_O	UART1_TXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	UART1_RXD_I	UART1_RXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	UART1_RTS_O	UART1_RTS function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.3 PINASSIGN2

Table 109. Pin assign register 2 (PINASSIGN2, address 0x4003 8008) bit description

Bit	Symbol	Description	Reset value
7:0	UART1_CTS_I	UART1_CTS function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

Table 109. Pin assign register 2 (PINASSIGN2, address 0x4003 8008) bit description

Bit	Symbol	Description	Reset value
15:8	UART1_SCLK_IO	UART1_SCLK function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	UART2_TXD_O	UART2_TXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	UART2_RXD_I	UART2_RXD function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.4 PINASSIGN3

Table 110. Pin assign register 3 (PINASSIGN3, address 0x4003 800C) bit description

Bit	Symbol	Description	Reset value
7:0	UART2_SCLK_IO	UART2_SCLK function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SPI0_SCK_IO	SPI0_SCK function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0,..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SPI0_MOSI_IO	SPI0_MOSI function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SPI0_MISO_IO	SPI0_MISO function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.5 PINASSIGN4

Table 111. Pin assign register 4 (PINASSIGN4, address 0x4003 8010) bit description

Bit	Symbol	Description	Reset value
7:0	SPI0_SSELSN_0_IO	SPI0_SSELSN_0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SPI0_SSELSN_1_IO	SPI0_SSELSN_1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SPI0_SSELSN_2_IO	SPI0_SSELSN_2 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SPI0_SSELSN_3_IO	SPI0_SSELSN_3 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.6 PINASSIGN5

Table 112. Pin assign register 5 (PINASSIGN5, address 0x4003 8014) bit description

Bit	Symbol	Description	Reset value
7:0	SPI1_SCK_IO	SPI1_SCK function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SPI1_MOSI_IO	SPI1_MOSI function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SPI1_MISO_IO	SPI1_MISO function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SPI1_SSELSN_0_IO	SPI1_SSELSN_0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.7 PINASSIGN6

Table 113. Pin assign register 6 (PINASSIGN6, address 0x4003 8018) bit description

Bit	Symbol	Description	Reset value
7:0	SPI1_SSELSN_1_IO	SPI1_SSELSN_1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	CAN_TD1_O	CAN_TD1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	CAN_RD1_I	CAN_RD1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	-	Reserved,	0xFF

8.5.8 PINASSIGN7

Table 114. Pin assign register 7 (PINASSIGN7, address 0x4003 801C) bit description

Bit	Symbol	Description	Reset value
7:0	USB_VBUS_I	USB_VBUS function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SCT0_OUT0_O	SCT0_OUT0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SCT0_OUT1_O	SCT0_OUT1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SCT0_OUT2_O	SCT0_OUT2 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.9 PINASSIGN8

Table 115. Pin assign register 8 (PINASSIGN8, address 0x4003 8020) bit description

Bit	Symbol	Description	Reset value
7:0	SCT1_OUT0_O	SCT1_OUT0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SCT1_OUT1_O	SCT1_OUT1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SCT1_OUT2_O	SCT1_OUT2 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SCT2_OUT0_O	SCT2_OUT0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.10 PINASSIGN9

Table 116. Pin assign register 9 (PINASSIGN9, address 0x4003 8024) bit description

Bit	Symbol	Description	Reset value
7:0	SCT2_OUT1_O	SCT2_OUT1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SCT2_OUT2_O	SCT2_OUT2 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SCT3_OUT0_O	SCT3_OUT0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	SCT3_OUT1_O	SCT3_OUT1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.11 PINASSIGN10

Table 117. Pin assign register 10 (PINASSIGN10, address 0x4003 8028) bit description

Bit	Symbol	Description	Reset value
7:0	SCT3_OUT2_O	SCT3_OUT2 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

Table 117. Pin assign register 10 (PINASSIGN10, address 0x4003 8028) bit description

Bit	Symbol	Description	Reset value
15:8	SCT_ABORT0_I	SCT_ABORT0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	SCT_ABORT1_I	SCT_ABORT1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	ADC0_PIN_TRIG0_I	ADC0_PIN_TRIG0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.12 PINASSIGN11

Table 118. Pin assign register 11 (PINASSIGN11, address 0x4003 802C) bit description

Bit	Symbol	Description	Reset value
7:0	ADC0_PIN_TRIG1_I	ADC0_PIN_TRIG1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	ADC1_PIN_TRIG0_I	ADC1_PIN_TRIG0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	ADC1_PIN_TRIG1_I	ADC1_PIN_TRIG1 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	DAC_PIN_TRIG_I	DAC_PIN_TRIG function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.13 PINASSIGN12

Table 119. Pin assign register 12 (PINASSIGN12, address 0x4003 8030) bit description

Bit	Symbol	Description	Reset value
7:0	DAC_SHUTOFF_I	DAC_SHUTOFF function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	ACMP0_OUT_O	ACMP0_OUT function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	ACMP1_OUT_O	ACMP1_OUT function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	ACMP2_OUT_O	ACMP2_OUT function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.14 PINASSIGN13

Table 120. Pin assign register 13 (PINASSIGN13, address 0x4003 8034) bit description

Bit	Symbol	Description	Reset value
7:0	ACMP3_OUT_O	ACMP3_OUT function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	CLK_OUT_O	CLK_OUT function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	ROSC0_O	ROSC0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	ROSC_RST0_I	ROSC_RST0 function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.15 PINASSIGN14

Table 121. Pin assign register 14 (PINASSIGN14, address 0x4003 8038) bit description

Bit	Symbol	Description	Reset value
7:0	USB_FRAME_TOG_O	USB_FRAME_TOG function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	QEI0_PHA_I	QEI0_PHA function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	QEI0_PHB_I	QEI0_PHB function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
31:24	QEI0_IDX_I	QEI0_IDX function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF

8.5.16 PINASSIGN15

Table 122. Pin assign register 15 (PINASSIGN15, address 0x4003 803C) bit description

Bit	Symbol	Description	Reset value
7:0	GPIO_INT_BMATCH_O	GPIO_INT_BMATCH function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
15:8	SWO_O	SWO function assignment. The value is the pin number to be assigned to this function. PIO0_0 = 0, ..., PIO1_0 = 32, ..., PIO2_11 = 75.	0xFF
23:16	-	Reserved.	0xFF
31:24	-	Reserved.	0xFF

8.5.17 PINENABLE0

This register enables analog or digital fixed-pin functions that can only be assigned to one pin.

If the fixed-pin function is disabled, this pin is connected to GPIO or any digital pin function that is assigned to it through the PINASSIGN registers. By default the fixed-pin function is deselected and GPIO is assigned to this pin.

Table 123. Pin enable register 0 (PINENABLE0, address 0x4003 81C0) bit description

Bit	Symbol	Value	Description	Reset value
0	ADC0_0		ADC0_0 function enable.	1
		0	Enabled on pin PIO0_8.	
		1	Disabled.	
1	ADC0_1		ADC0_1 function enable.	1
		0	Enabled on pin PIO0_7.	
		1	Disabled.	
2	ADC0_2		ADC0_2 function enable.	1
		0	Enabled on pin PIO0_6.	
		1	Disabled.	
3	ADC0_3		ADC0_3 function enable.	1
		0	Enabled on pin PIO0_5.	
		1	Disabled.	
4	ADC0_4		ADC0_4 function enable.	1
		0	Enabled on pin PIO0_4.	
		1	Disabled.	
5	ADC0_5		ADC0_5 function enable.	1
		0	Enabled on pin PIO0_3.	
		1	Disabled.	
6	ADC0_6		ADC0_6 function enable.	1
		0	Enabled on pin PIO0_2.	
		1	Disabled.	
7	ADC0_7		ADC0_7 function enable.	1
		0	Enabled on pin PIO0_1.	
		1	Disabled.	
8	ADC0_8		ADC0_8 function enable.	1
		0	Enabled on pin PIO1_0.	
		1	Disabled.	
9	ADC0_9		ADC0_9 function enable.	1
		0	Enabled on pin PIO0_31.	
		1	Disabled.	
10	ADC0_10		ADC0_10 function enable.	1
		0	Enabled on pin PIO0_0.	
		1	Disabled.	

Table 123. Pin enable register 0 (PINENABLE0, address 0x4003 81C0) bit description

Bit	Symbol	Value	Description	Reset value
11	ADC0_11		ADC0_11 function enable.	1
		0	Enabled on pin PIO0_30.	
		1	Disabled.	
12	ADC1_0		ADC1_0 function enable.	1
		0	Enabled on pin PIO1_1.	
		1	Disabled.	
13	ADC1_1		ADC1_1 function enable.	1
		0	Enabled on pin PIO0_9.	
		1	Disabled.	
14	ADC1_2		ADC1_2 function enable.	1
		0	Enabled on pin PIO0_10.	
		1	Disabled.	
15	ADC1_3		ADC1_3 function enable.	1
		0	Enabled on pin PIO0_11.	
		1	Disabled.	
16	ADC1_4		ADC1_4 function enable.	1
		0	Enabled on pin PIO1_2.	
		1	Disabled.	
17	ADC1_5		ADC1_5 function enable.	1
		0	Enabled on pin PIO1_3.	
		1	Disabled.	
18	ADC1_6		ADC1_6 function enable.	1
		0	Enabled on pin PIO0_13.	
		1	Disabled.	
19	ADC1_7		ADC1_7 function enable.	1
		0	Enabled on pin PIO0_14.	
		1	Disabled.	
20	ADC1_8		ADC1_8 function enable.	1
		0	Enabled on pin PIO0_15.	
		1	Disabled.	
21	ADC1_9		ADC1_9 function enable.	1
		0	Enabled on pin PIO0_16.	
		1	Disabled.	
22	ADC1_10		ADC1_10 function enable.	1
		0	Enabled on pin PIO1_4.	
		1	Disabled.	
23	ADC1_11		ADC1_11 function enable.	1
		0	Enabled on pin PIO1_5.	
		1	Disabled.	
24	DAC_OUT		DAC_OUT function enable.	1

Table 123. Pin enable register 0 (PINENABLE0, address 0x4003 81C0) bit description

Bit	Symbol	Value	Description	Reset value
		0	Enabled on pin PIO0_12.	
		1	Disabled.	
25	ACMP_I1		ACMP input 1 (common input) function enable.	1
		0	Enabled on pin PIO0_27.	
		1	Disabled.	
26	ACMP_I2		ACMP input 2 (common input) function enable.	1
		0	Enabled on pin PIO1_6.	
		1	Disabled.	
27	ACMP0_I3		Analog comparator 0 input 3 function enable.	1
		0	Enabled on pin PIO0_26.	
		1	Disabled.	
28	ACMP0_I4		Analog comparator 0 input 4 function enable.	1
		0	Enabled on pin PIO0_25.	
		1	Disabled.	
29	ACMP1_I3		Analog comparator 1 input 3 function enable.	1
		0	Enabled on pin PIO0_28.	
		1	Disabled.	
30	ACMP1_I4		Analog comparator 1 input 4 function enable.	1
		0	Enabled on pin PIO1_10.	
		1	Disabled.	
31	ACMP2_I3		Analog comparator 2 input 3 function enable.	1
		0	Enabled on pin PIO0_29.	
		1	Disabled.	

8.5.18 PINENABLE 1

This register enables analog or digital fixed-pin functions that can only be assigned to one pin.

If the fixed-pin function is disabled, this pin is connected to GPIO or any digital pin function that is assigned to it through the PINASSIGN registers. By default the fixed-pin function is deselected and GPIO is assigned to this pin.

Table 124. Pin enable register 1 (PINENABLE1, address 0x4003 81C4) bit description

Bit	Symbol	Value	Description	Reset value
0	ACMP2_I4		Analog comparator 2 input 4 function enable.	1
		0	Enabled on pin PIO1_9.	
		1	Disabled.	
1	ACMP3_I3		Analog comparator 3 input 3 function enable.	1
		0	Enabled on pin PIO1_8.	
		1	Disabled.	
2	ACMP3_I4		Analog comparator 3 input 4 function enable.	1

Table 124. Pin enable register 1 (PINENABLE1, address 0x4003 81C4) bit description

Bit	Symbol	Value	Description	Reset value
		0	Enabled on pin PIO1_7.	
		1	Disabled.	
3	I2C0_SDA		I2C0_SDA function enable.	1
		0	Enabled on pin PIO0_23.	
		1	Disabled.	
4	I2C0_SCL		I2C0_SCL function enable.	1
		0	Enabled on pin PIO0_22.	
		1	Disabled.	
5	SCT0_OUT3		SCT0_OUT3 function enable.	1
		0	Enabled on pin PIO0_0.	
		1	Disabled.	
6	SCT0_OUT4		SCT0_OUT4 function enable.	1
		0	Enabled on pin PIO0_1.	
		1	Disabled.	
7	SCT0_OUT5		SCT0_OUT5 function enable.	1
		0	Enabled on function PIO0_18.	
		1	Disabled.	
8	SCT0_OUT6		SCT0_OUT6 function enable.	1
		0	Enabled on pin PIO0_24.	
		1	Disabled.	
9	SCT0_OUT7		SCT0_OUT7 function enable.	1
		0	Enabled on pin PIO1_14.	
		1	Disabled.	
10	SCT1_OUT3		SCT1_OUT3 function enable.	1
		0	Enabled on pin PIO0_2.	
		1	Disabled.	
11	SCT1_OUT4		SCT1_OUT4 function enable.	1
		0	Enabled on pin PIO0_3.	
		1	Disabled.	
12	SCT1_OUT5		SCT1_OUT5 function enable.	1
		0	Enabled on pin PIO0_14.	
		1	Disabled.	
13	SCT1_OUT6		SCT1_OUT6 function enable.	1
		0	Enabled on pin PIO0_20.	
		1	Disabled.	
14	SCT1_OUT7		SCT1_OUT7 function enable.	1
		0	Enabled on pin PIO1_17.	
		1	Disabled.	
15	SCT2_OUT3		SCT2_OUT3 function enable.	1
		0	Enabled on pin PIO0_6.	

Table 124. Pin enable register 1 (PINENABLE1, address 0x4003 81C4) bit description

Bit	Symbol	Value	Description	Reset value
		1	Disabled.	
16	SCT2_OUT4		SCT2_OUT4 function enable.	1
		0	Enabled on pin PIO0_29.	
		1	Disabled.	
17	SCT2_OUT5		SCT2_OUT5 function enable.	1
		0	Enabled on pin PIO1_20.	
		1	Disabled.	
18	SCT3_OUT3		SCT3_OUT3 function enable.	1
		0	Enabled on pin PIO0_26.	
		1	Disabled.	
19	SCT3_OUT4		SCT3_OUT4 function enable.	1
		0	Enabled on pin PIO1_8.	
		1	Disabled.	
20	SCT3_OUT5		SCT3_OUT5 function enable.	1
		0	Enabled on pin PIO1_24.	
		1	Disabled.	
21	RESETN		$\overline{\text{RESET}}$ function enable.	0
		0	Enabled on pin PIO0_21.	
		1	Disabled.	
22	SWCLK		SWCLK function enable.	0
		0	Enabled on pin PIO0_19.	
		1	Disabled.	
23	SWDIO		SWDIO function enable.	0
		0	Enabled on pin PIO0_20.	
		1	Disabled.	
31:24	-	-	Reserved.	1

9.1 How to read this chapter

Input multiplexing is available for all parts. Depending on the package, not all inputs from external pins may be available. See Table 96 “Available pin configuration registers”.

9.2 Features

- Configures the inputs to the SCTs.
- Configures the inputs to the pin interrupt block and pattern match engine.
- Configures the inputs to the DMA triggers.
- Configures the inputs to the frequency measure function. This function is controlled by the FREQMECTRL register in the SYSCON block.

9.3 Basic configuration

Once set up, no clocks are needed for the input multiplexer to function. The system clock is needed only to write to or read from the INPUT MUX registers. Once the input multiplexer is configured, disable the clock to the INPUT MUX block in the SYSAHBCLKCTRL register.

9.4 Pin description

The input multiplexer has no dedicated pins. However, several external pins can be selected as inputs to the SCT input multiplexer and all digital pins of ports 0 and 1 can be selected as inputs to the pin interrupts. Multiplexer inputs from external pins work independently of any function assigned to the pin through the switch matrix as long as no analog function is enabled.

Table 125. INPUT MUX pin description

Pins	Peripheral	Input mux Reference
PIO0_2, PIO0_3, PIO0_17, PIO0_30, PIO1_6, PIO1_7, PIO1_12, PIO1_13	SCT0	Table 127
PIO0_15, PIO0_16, PIO0_21, PIO0_31, PIO1_4, PIO1_5, PIO1_15, PIO1_16	SCT1	Table 128
P0_4, P0_27, P1_18, P1_19	SCT2	Table 129
PIO0_7, PIO1_11, PIO1_21, PIO1_22	SCT3	Table 130
PIO0_n, PIO1_n (n = 0 to 31)	Pin interrupts 0 to 7	Table 131
PIO0_5, PIO0_19, PIO0_30, PIO1_27	Frequency measure block	Table 134

9.5 General description

The inputs to the four SCTs, to the DMA trigger, to the eight pin interrupts, and to the frequency measure block are multiplexed to multiple input sources. The sources can be external pins, interrupts, or output signals of other peripherals.

The input multiplexing makes it possible to design complex event-driven processes without CPU intervention by connecting peripherals like the SCTs and the ADCs, the SCTs and the analog comparators, or two SCTs internally with each other.

The DMA can use trigger input multiplexing to sequence DMA transactions without the use of interrupt service routines.

9.5.1 SCT input multiplexing

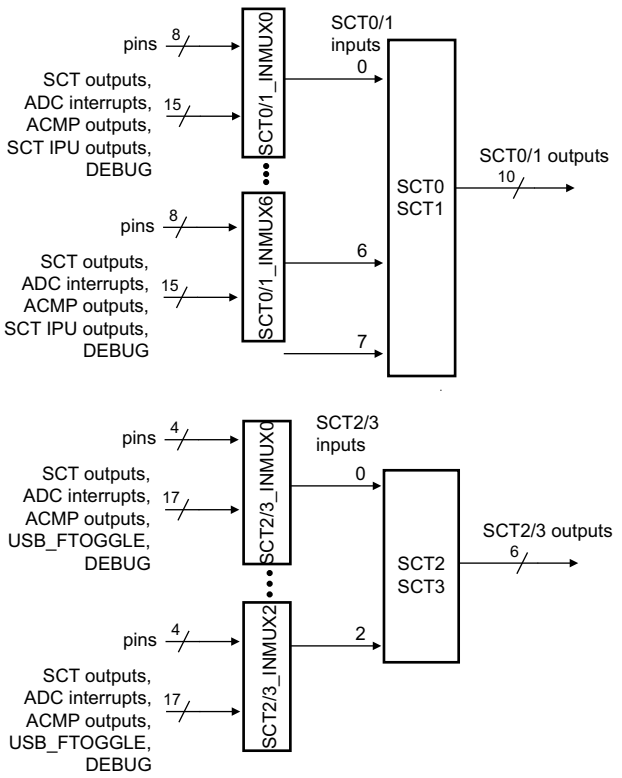


Fig 13. SCT input multiplexing

9.5.2 Pin interrupt input multiplexing

The input mux for the pin interrupts and pattern match engine multiplexes all pins from ports 0 and 1.

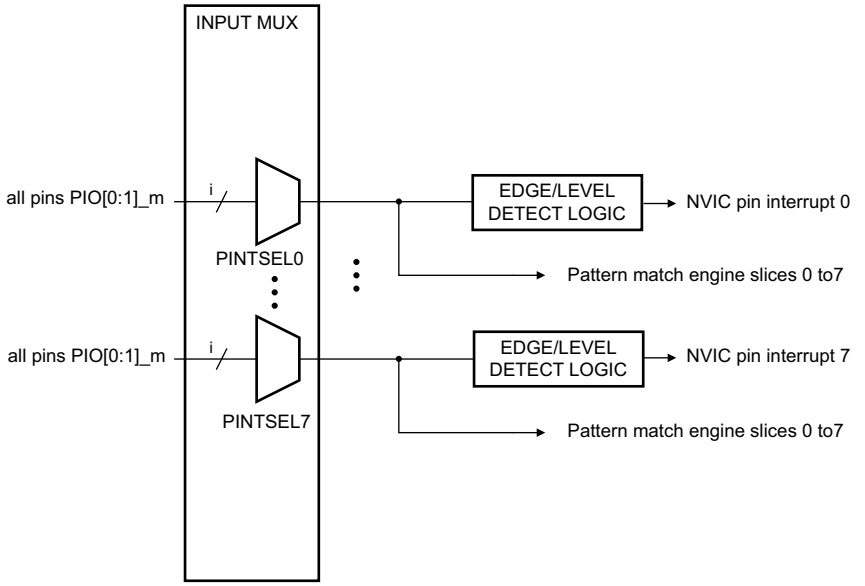


Fig 14. Pin interrupt multiplexing

9.5.3 DMA trigger input multiplexing

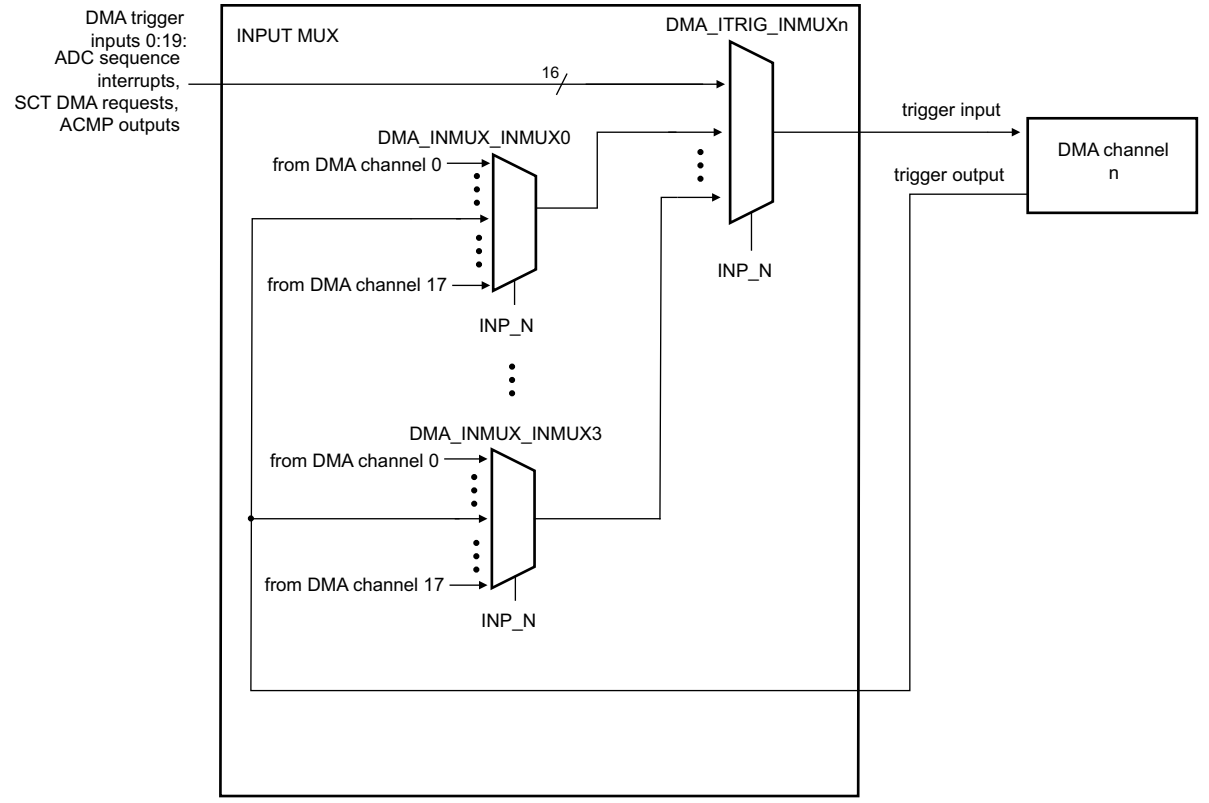


Fig 15. DMA trigger multiplexing

9.6 Register description

All input mux registers reside on word address boundaries. Details of the registers appear in the description of each function.

All address offsets not shown in Table 126 are reserved and should not be written to.

Table 126. Register overview: Input multiplexing (base address 0x4001 4000)

Name	Access	Offset	Description	Reset value	Reference
SCT0_INMUX0	R/W	0x000	Input mux register for SCT0 input 0	0x1F	Table 127
SCT0_INMUX1	R/W	0x004	Input mux register for SCT0 input 1	0x1F	Table 127
SCT0_INMUX2	R/W	0x008	Input mux register for SCT0 input 2	0x1F	Table 127
SCT0_INMUX3	R/W	0x00C	Input mux register for SCT0 input 3	0x1F	Table 127
SCT0_INMUX4	R/W	0x010	Input mux register for SCT0 input 4	0x1F	Table 127
SCT0_INMUX5	R/W	0x014	Input mux register for SCT0 input 5	0x1F	Table 127
SCT0_INMUX6	R/W	0x018	Input mux register for SCT0 input 6	0x1F	Table 127
-	-	0x01C	Reserved.	-	-
SCT1_INMUX0		0x020	Input mux register for SCT1 input 0	0x1F	Table 128
SCT1_INMUX1	R/W	0x024	Input mux register for SCT1 input 1	0x1F	Table 128
SCT1_INMUX2	R/W	0x028	Input mux register for SCT1 input 2	0x1F	Table 128
SCT1_INMUX3	R/W	0x02C	Input mux register for SCT1 input 3	0x1F	Table 128
SCT1_INMUX4	R/W	0x030	Input mux register for SCT1 input 4	0x1F	Table 128
SCT1_INMUX5	R/W	0x034	Input mux register for SCT1 input 5	0x1F	Table 128
SCT1_INMUX6	R/W	0x038	Input mux register for SCT1 input 6	0x1F	Table 128
-	-	0x03C	Reserved.	-	-
SCT2_INMUX0	R/W	0x040	Input mux register for SCT2 input 0	0x1F	Table 129
SCT2_INMUX1	R/W	0x044	Input mux register for SCT2 input 1	0x1F	Table 129
SCT2_INMUX2	R/W	0x048	Input mux register for SCT2 input 2	0x1F	Table 129
-	-	0x04C - 0x05C	Reserved.	-	-
SCT3_INMUX0	R/W	0x060	Input mux register for SCT3 input 0	0x1F	Table 130
SCT3_INMUX1	R/W	0x064	Input mux register for SCT3 input 1	0x1F	Table 130
SCT3_INMUX2	R/W	0x068	Input mux register for SCT3 input 2	0x1F	Table 130
-	-	0x06C - 0x0BC	Reserved.	-	-
PINTSEL0	R/W	0x0C0	Pin interrupt select register 0	0x7F	Table 131
PINTSEL1	R/W	0x0C4	Pin interrupt select register 1	0x7F	Table 131
PINTSEL2	R/W	0x0C8	Pin interrupt select register 2	0x7F	Table 131
PINTSEL3	R/W	0x0CC	Pin interrupt select register 3	0x7F	Table 131
PINTSEL4	R/W	0x0D0	Pin interrupt select register 4	0x7F	Table 131
PINTSEL5	R/W	0x0D4	Pin interrupt select register 5	0x7F	Table 131
PINTSEL6	R/W	0x0D8	Pin interrupt select register 6	0x7F	Table 131

Table 126. Register overview: Input multiplexing (base address 0x4001 4000) *…continued*

Name	Access	Offset	Description	Reset value	Reference
PINTSEL7	R/W	0x0DC	Pin interrupt select register 7	0x7F	Table 131
DMA_ITRIG_INMUX0	R/W	0x0E0	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX1	R/W	0x0E4	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX2	R/W	0x0E8	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX3	R/W	0x0EC	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX4	R/W	0x0F0	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX5	R/W	0x0F4	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX6	R/W	0x0F8	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX7	R/W	0x0FC	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX8	R/W	0x100	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX9	R/W	0x104	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX10	R/W	0x108	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX11	R/W	0x10C	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX12	R/W	0x110	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX13	R/W	0x114	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX14	R/W	0x118	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX15	R/W	0x11C	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132

Table 126. Register overview: Input multiplexing (base address 0x4001 4000) *…continued*

Name	Access	Offset	Description	Reset value	Reference
DMA_ITRIG_INMUX16	R/W	0x120	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_ITRIG_INMUX17	R/W	0x124	Input mux register for trigger inputs 0 to 23 connected to DMA channel 0. Selects from ADC, SCT, ACMP DMA interrupts and DMA requests.	0x1F	Table 132
DMA_INMUX_INMUX0	R/W	0x140	Input mux register for DMA trigger input 20. Selects from 18 DMA trigger outputs.	0x1F	Table 133
DMA_INMUX_INMUX1	R/W	0x144	Input mux register for DMA trigger input 21. Selects from 18 DMA trigger outputs.	0x1F	Table 133
DMA_INMUX_INMUX2	R/W	0x148	Input mux register for DMA trigger input 22. Selects from 18 DMA trigger outputs.	0x1F	Table 133
DMA_INMUX_INMUX3	R/W	0x14C	Input mux register for DMA trigger input 23. Selects from 18 DMA trigger outputs.	0x1F	Table 133
FREQMEAS_REF	R/W	0x160	Clock selection for frequency measurement function reference clock	0xF	Table 134
FREQMEAS_TARGET	R/W	0x164	Clock selection for frequency measurement function target clock	0xF	Table 135

9.6.1 SCT0 Input mux registers 0 to 6

With the SCT0 Input mux registers you can select one input source for each SCT0 input from 23 external and internal sources. (An exception is SCT0 input SCT0_IN7 which is connected to the SCT PLL clock and not multiplexed with any other signals.)

The output of SCT0 Input mux register 0 selects the source for SCT0 input 0, the output of SCT0 Input mux register 1 selects the source for SCT0 input 1, and so forth up to SCT0 Input mux register 6, which selects the input for SCT0 input 6.

The value to be programmed in this register is the Input mux input number ranging from 0 for pin PIO0_2 to 22 for the DEBUG_HALTED signal from the ARM CoreSight debug signal.

Inputs 0 to 7 are directly connected to specific external pins as indicated in the register description table. The same pins can also be connected through the switch matrix as inputs to another peripheral.

See Section 15.3.1 “SCT inputs and outputs” for details.

Table 127. SCT0 Input mux registers 0 to 6 (SCT0_INMUX[0:6], address 0x4001 4000 (SCT0_INMUX0) to 0x4001 4018 (SCT0_INMUX6)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	INP_N		Input number (decimal value) to SCT0 inputs 0 to 6.	0x1F
		0x0	PIO0_2. (external pin)	
		0x1	PIO0_3. (external pin)	
		0x2	PIO0_17. (external pin)	
		0x3	PIO0_30. (external pin)	

Table 127. SCT0 Input mux registers 0 to 6 (SCT0_INMUX[0:6], address 0x4001 4000 (SCT0_INMUX0) to 0x4001 4018 (SCT0_INMUX6)) bit description

Bit	Symbol	Value	Description	Reset value
		0x4	PIO1_6. (external pin)	
		0x5	PIO1_7. (external pin)	
		0x6	PIO1_12. (external pin)	
		0x7	PIO1_13. (external pin)	
		0x8	SCT1_OUT4. (large SCT1 output 4)	
		0x9	SCT2_OUT4. (small SCT2 output 4)	
		0xA	SCT2_OUT5. (small SCT2 output 5)	
		0xB	ADC0_THCMP_IRQ. (ADC0 threshold compare interrupt)	
		0xC	ADC1_THCMP_IRQ	
		0xD	ACMP0_OUT. (One output from each analog comparator)	
		0xE	ACMP1_OUT	
		0xF	ACMP2_OUT	
		0x10	ACMP3_OUT	
		0x11	SCTIPU_ABORT	
		0x12	SCTIPU_SAMPLE0	
		0x13	SCTIPU_SAMPLE1	
		0x14	SCTIPU_SAMPLE2	
		0x15	SCTIPU_SAMPLE3	
		0x16	DEBUG_HALTED. (from ARM Cortex CoreSight Debugger)	
31:5	-		Reserved.	-

9.6.2 SCT1 Input mux registers 0 to 6

With the SCT1 Input mux registers you can select one input source for each SCT1 input from 23 external and internal sources. (An exception is SCT1 input SCT1_IN7 which is connected to the SCT PLL clock and not multiplexed with any other signals.)

The output of SCT1 Input mux register 0 selects the source for SCT1 input 0, the output of SCT1 Input mux register 1 selects the source for SCT1 input 1, and so forth up to SCT1 Input mux register 6, which selects the input for SCT1 input 6.

The value to be programmed in this register is the Input mux input number ranging from 0 for pin PIO0_15 to 22 for the DEBUG_HALTED signal from the ARM CoreSight debug signal.

Inputs 0 to 7 are directly connected to specific external pins as indicated in the register description table. The same pins can also be connected through the switch matrix as inputs to another peripheral.

See Section 15.3.1 “SCT inputs and outputs” for details.

Table 128. SCT1 Input mux registers 0 to 6 (SCT1_INMUX[0:6], address 0x4007 4020 (SCT1_INMUX0) to 0x4001 4038 (SCT1_INMUX6)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	INP_N		Input number (decimal value) to SCT1 inputs 0 to 6.	0x1F
		0x0	PIO0_15. (external pin)	
		0x1	PIO0_16. (external pin)	
		0x2	PIO0_21. (external pin)	
		0x3	PIO0_31. (external pin)	
		0x4	PIO1_4 (external pin)	
		0x5	PIO1_5. (external pin)	
		0x6	PIO1_15. (external pin)	
		0x7	PIO1_16. (external pin)	
		0x8	SCT0_OUT4 (large SCT0 output 4)	
		0x9	SCT3_OUT4 (small SCT3 output 4)	
		0xA	SCT3_OUT5 (small SCT3 output 5)	
		0xB	ADC0_THCMP_IRQ. (ADC0 threshold compare interrupt)	
		0xC	ADC1_THCMP_IRQ	
		0xD	ACMP0_OUT. (One output from each analog comparator)	
		0xE	ACMP1_OUT	
		0xF	ACMP2_OUT	
		0x10	ACMP3_OUT	
		0x11	SCTIPU_ABORT	
		0x12	SCTIPU_SAMPLE0	
		0x13	SCTIPU_SAMPLE1	
		0x14	SCTIPU_SAMPLE2	
		0x15	SCTIPU_SAMPLE3	
		0x16	DEBUG_HALTED. (from ARM Cortex CoreSight Debugger)	
31:5	-		Reserved.	-

9.6.3 SCT2 Input mux registers 0 to 2

With the SCT2 Input mux registers you can select one input source for each SCT2 input from 21 external and internal sources.

The output of SCT2 Input mux register 0 selects the source for SCT2 input 0, the output of SCT2 Input mux register 1 selects the source for SCT2 input 1, and SCT2 Input mux register 2, selects the input for SCT2 input 2.

The value to be programmed in this register is the Input mux input number ranging from 0 for pin PIO0_4 to 20 for the DEBUG_HALTED signal from the ARM CoreSight debug signal.

Inputs 0 to 3 are directly connected to specific external pins as indicated in the register description table. The same pins can also be connected through the switch matrix as inputs to another peripheral.

See Section 16.3.1 “SCT inputs and outputs” for details.

Table 129. SCT2 Input mux registers 0 to 2 (SCT2_INMUX[0:2], address 0x4001 4040 (SCT2_INMUX0) to 0x4001 4048 (SCT2_INMUX2)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	INP_N		Input number (decimal value) to SCT2 inputs 0 to 2.	0x1F
		0x0	PIO0_4. (external pin)	
		0x1	PIO0_27. (external pin)	
		0x2	PIO1_18. (external pin)	
		0x3	PIO1_19. (external pin)	
		0x4	SCT0_OUT4	
		0x5	SCT0_OUT5	
		0x6	SCT0_OUT7	
		0x7	SCT0_OUT8	
		0x8	ADC0_THCMP_IRQ	
		0x9	ADC1_THCMP_IRQ	
		0xA	ACMP0_OUT (One output from each analog comparator)	
		0xB	ACMP1_OUT	
		0xC	ACMP2_OUT	
		0xD	ACMP3_OUT	
		0xE	SCTIPU_ABORT	
		0xF	SCTIPU_SAMPLE0	
		0x10	SCTIPU_SAMPLE1	
		0x11	SCTIPU_SAMPLE2	
		0x12	SCTIPU_SAMPLE3	
		0x13	USB_FRAME_TOGGLE	
		0x14	DEBUG_HALTED	
31:5	-		Reserved.	-

9.6.4 SCT3 Input mux registers 0 to 2

With the SCT3 Input mux registers you can select one input source for each SCT3 input from 21 external and internal sources.

The output of SCT3 Input mux register 0 selects the source for SCT3 input 0, the output of SCT3 Input mux register 1 selects the source for SCT3 input 1, and SCT3 Input mux register 2, selects the input for SCT3 input 2.

The value to be programmed in this register is the Input mux input number ranging from 0 for pin PIO0_7 to 20 for the DEBUG_HALTED signal from the ARM CoreSight debug signal.

Inputs 0 to 3 are directly connected to specific external pins as indicated in the register description table. The same pins can also be connected through the switch matrix as inputs to another peripheral.

See Section 16.3.1 “SCT inputs and outputs” for details.

Table 130. SCT3 Input mux registers 0 to 2 (SCT3_INMUX[0:2], address 0x4001 4060 (SCT3_INMUX0) to 0x4001 4068 (SCT3_INMUX2)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	INP_N		Input number (decimal value) to SCT3 inputs 0 to 2.	0x1F
		0x0	PIO0_7. (external pin)	
		0x1	PIO1_11. (external pin)	
		0x2	PIO1_21. (external pin)	
		0x3	PIO1_22. (external pin)	
		0x4	SCT1_OUT4	
		0x5	SCT1_OUT5	
		0x6	SCT1_OUT7	
		0x7	SCT1_OUT8	
		0x8	ADC0_THCMP_IRQ	
		0x9	ADC1_THCMP_IRQ	
		0xA	ACMP0_OUT	
		0xB	ACMP1_OUT	
		0xC	ACMP2_OUT	
		0xD	ACMP3_OUT	
		0xE	SCTIPU_ABORT3	
		0xF	SCTIPU_SAMPLE0	
		0x10	SCTIPU_SAMPLE1	
		0x11	SCTIPU_SAMPLE2	
		0x12	SCTIPU_SAMPLE3	
		0x13	USB_FRAME_TOGGLE	
		0x14	DEBUG_HALTED	
31:5	-		Reserved.	-

9.6.5 Pin interrupt select registers

Each of these 8 registers selects one pin from all digital pins on GPIO ports 0 and 1 as the source of a pin interrupt or as the input to the pattern match engine. To select a pin for any of the eight pin interrupts or pattern match engine inputs, write the GPIO port pin number as 0 to 31 for pins PIO0_0 to PIO0_31 to the INTPIN bits. Port 1 pins correspond to pin numbers 32 to 63. For example, setting INTPIN to 0x5 in PINTSEL0 selects pin PIO0_5 for pin interrupt 0.

Remark: The GPIO port pin number serves to identify the pin to the PINTSEL register. Any digital function, including GPIO, can be assigned to this pin through the switch matrix.

Each of the 8 pin interrupts must be enabled in the NVIC using interrupt slots # 7 to 14 (see Table 2).

To use the selected pins for pin interrupts or the pattern match engine, see Section 12.5.2 “Pattern match engine”.

Table 131. Pin interrupt select registers (PINTSEL[0:7], address 0x4001 40C0 (PINTSEL0) to 0x4001 40DC (PINTSEL7)) bit description

Bit	Symbol	Description	Reset value
7:0	INTPIN	Pin number select for pin interrupt or pattern match engine input. (PIO0_0 to PIO1_31 correspond to numbers 0 to 63).	0x7F
31:8	-	Reserved	-

9.6.6 DMA input trigger input mux registers 0 to 17

With the DMA input trigger input mux registers you can select one trigger input for each of the 18 DMA channels from 20 internal sources.

By default, none of the triggers are selected.

Table 132. DMA input trigger Input mux registers 0 to 17 (DMA_ITRIG_INMUX[0:17], address 0x4001 40E0 (DMA_ITRIG_INMUX0) to 0x4001 4124 (DMA_ITRIG_INMUX17)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	INP		Trigger input number (decimal value) for DMA channel n (n = 0 to 17).	0x1F
		0x0	ADC0_SEQA_IRQ	
		0x1	ADC0_SEQB_IRQ	
		0x2	ADC1_SEQA_IRQ	
		0x3	ADC1_SEQB_IRQ	
		0x4	SCT0_DMA0	
		0x5	SCT0_DMA1	
		0x6	SCT1_DMA0	
		0x7	SCT1_DMA1	
		0x8	SCT2_DMA0	
		0x9	SCT2_DMA1	
		0xA	SCT3_DMA0	
		0xB	SCT3_DMA1	
		0xC	ACMP0_OUT. (One output from each analog comparator)	
		0xD	ACMP1_OUT	
		0xE	ACMP2_OUT	
		0xF	ACMP3_OUT	
		0x10	DMA trigger mux 0. (DMA_INMUX_INMUX0).	
		0x11	DMA trigger mux 1. (DMA_INMUX_INMUX1)	
		0x12	DMA trigger mux 2. (DMA_INMUX_INMUX2).	
		0x13	DMA trigger mux 3. (DMA_INMUX_INMUX3).	
31:5	-		Reserved.	-

9.6.7 DMA trigger input mux input mux registers 0 to 3

This register provides a multiplexer for inputs 16 to 19 of each DMA trigger input mux register DMA_ITRIG_INMUX. These inputs can be selected from the 18 trigger outputs generated by the DMA (one trigger output per channel).

By default, none of the triggers are selected.

Table 133. DMA input trigger input mux input mux registers 0 to 3 (DMA_INMUX_INMUX[0:3], address 0x4001 4140 (DMA_INMUX_INMUX0) to 0x4001 414C (DMA_INMUX_INMUX3)) bit description

Bit	Symbol	Description	Reset value
4:0	INP	DMA trigger output number (decimal value) for DMA channel n (n = 0 to 17).	0x1F
31:5	-	Reserved.	-

9.6.8 Frequency measure function reference clock select register

This register selects a clock for the reference clock of the frequency measure function.

By default, no clock is selected.

Table 134. Frequency measure function frequency clock select register (FREQMEAS_REF, address 0x4001 4160) bit description

Bit	Symbol	Value	Description	Reset value
3:0	CLKIN		Clock source number (decimal value) for frequency measure function target clock.	0xF
		0x0	System oscillator (MAIN_OSC)	
		0x1	IRC	
		0x2	WDOSC	
		0x3	32KHZOSC	
		0x4	USB_FTOGGLE	
		0x5	PIO0_5. (external pin)	
		0x6	PIO0_19. (external pin)	
		0x7	PIO0_30. (external pin)	
		0x8	PIO1_27. (external pin)	
31:4	-		Reserved.	-

9.6.9 Frequency measure function target clock select register

This register selects a clock for the target clock of the frequency measure function.

By default, no clock is selected.

Table 135. Frequency measure function target clock select register (FREQMEAS_TARGET, address 0x4001 4164) bit description

Bit	Symbol	Value	Description	Reset value
3:0	CLKIN		Clock source number (decimal value) for frequency measure function target clock.	0xF
		0x0	System oscillator (MAIN_OSC)	
		0x1	IRC	
		0x2	WDOSC	
		0x3	32KHZOSC	
		0x4	USB_FTOGGLE	
		0x5	PIO0_5. (external pin)	
		0x6	PIO0_19. (external pin)	
		0x7	PIO0_30. (external pin)	
		0x8	PIO1_27. (external pin)	
31:4	-		Reserved.	-

10.1 How to read this chapter

All GPIO registers refer to 32 pins on each port. Depending on the package type, not all pins are available, and the corresponding bits in the GPIO registers are reserved (see Table 136).

Table 136. GPIO pins available

Package	GPIO Port 0	GPIO Port 1	GPIO Port 2
LQFP48	PIO0_0 to PIO0_29	-	-
LQFP64	PIO0_0 to PIO0_31	PIO1_0 to PIO1_11	-
LQFP100	PIO0_0 to PIO0_31	PIO1_0 to PIO1_31	PIO2_0 to PIO2_11

10.2 Basic configuration

For the GPIO port registers, enable the clock to each GPIO port in the SYSAHBCLKCTRL0 register (Table 50, bit 14 to 16).

10.3 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.

10.4 General description

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.

The GPIOs can be used as external interrupts together with the pin interrupt and group interrupt blocks, see Section 12.2 and Section 11.2.

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

10.5 Register description

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

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

Remark: 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.

Table 137. Register overview: GPIO port (base address 0x1C00 0000)

Name	Access	Address offset	Description	Reset value	Width	Reference
B0 to B31	R/W	0x0000 to 0x001F	Byte pin registers port 0; pins PIO0_0 to PIO0_31	ext	byte (8 bit)	Table 138
B32 to B63	R/W	0x0020 to 0x003F	Byte pin registers port 1	ext	byte (8 bit)	Table 138
B64 to B75	R/W	0x0040 to 0x004B	Byte pin registers port 2	ext	byte (8 bit)	Table 138
W0 to W31	R/W	0x1000 to 0x107C	Word pin registers port 0	ext	word (32 bit)	Table 139
W32 to W63	R/W	0x1080 to 0x10FC	Word pin registers port 1	ext	word (32 bit)	Table 139
W64 to W75	R/W	0x1100 to 0x112C	Word pin registers port 2	ext	word (32 bit)	Table 139
DIR0	R/W	0x2000	Direction registers port 0	0	word (32 bit)	Table 140
DIR1	R/W	0x2004	Direction registers port 1	0	word (32 bit)	Table 140
DIR2	R/W	0x2008	Direction registers port 2	0	word (32 bit)	Table 140
MASK0	R/W	0x2080	Mask register port 0	0	word (32 bit)	Table 141
MASK1	R/W	0x2084	Mask register port 1	0	word (32 bit)	Table 141
MASK2	R/W	0x2088	Mask register port 2	0	word (32 bit)	Table 141
PIN0	R/W	0x2100	Port pin register port 0	ext	word (32 bit)	Table 142
PIN1	R/W	0x2104	Port pin register port 1	ext	word (32 bit)	Table 142
PIN2	R/W	0x2108	Port pin register port 2	ext	word (32 bit)	Table 142
MPIN0	R/W	0x2180	Masked port register port 0	ext	word (32 bit)	Table 143
MPIN1	R/W	0x2184	Masked port register port 1	ext	word (32 bit)	Table 143
MPIN2	R/W	0x2188	Masked port register port 2	ext	word (32 bit)	Table 143
SET0	R/W	0x2200	Write: Set register for port 0 Read: output bits for port 0	0	word (32 bit)	Table 144
SET1	R/W	0x2204	Write: Set register for port 1 Read: output bits for port 1	0	word (32 bit)	Table 144
SET2	R/W	0x2208	Write: Set register for port 2 Read: output bits for port 2	0	word (32 bit)	Table 144
CLR0	WO	0x2280	Clear port 0	NA	word (32 bit)	Table 145
CLR1	WO	0x2284	Clear port 1	NA	word (32 bit)	Table 145
CLR2	WO	0x2288	Clear port 2	NA	word (32 bit)	Table 145
NOT0	WO	0x2300	Toggle port 0	NA	word (32 bit)	Table 146
NOT1	WO	0x2304	Toggle port 1	NA	word (32 bit)	Table 146
NOT2	WO	0x2308	Toggle port 2	NA	word (32 bit)	Table 146

10.5.1 GPIO port byte pin registers

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

Table 138. GPIO port byte pin registers (B[0:B75], addresses 0x1C00 0000 (B0) to 0x1C00 004B (B75)) bit description

Bit	Symbol	Description	Reset value	Access
0	PBYTE	Read: state of the pin PIOm_n, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as 0. One register for each port pin: m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port 2. Write: loads the pin's output bit.	ext	R/W
7:1		Reserved (0 on read, ignored on write)	0	-

10.5.2 GPIO port word pin registers

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

Table 139. GPIO port word pin registers (W[0:75], addresses 0x1C00 1000 (W0) to 0x1C00 112C (W75)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PWORD	Read 0: pin PIOm_n is LOW. Write 0: clear output bit. Read 0xFFFF FFFF: pin PIOm_n is HIGH. Write any value 0x0000 0001 to 0xFFFF FFFF: set output bit. **Remark:** Only 0 or 0xFFFF FFFF can be read. Writing any value other than 0 will set the output bit. One register for each port pin: m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port2.	ext	R/W

10.5.3 GPIO port direction registers

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

Table 140. GPIO direction port register (DIR[0:2], address 0x1C00 2000 (DIR0) to 0x1C00 2008 (DIR2)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	DIRP	Selects pin direction for pin PIOm_n (bit 0 = PIOm_0, bit 1 = PIOm_1, ..., bit 31 = PIOm_31). m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port2. 0 = input. 1 = output.	0	R/W

10.5.4 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 141. GPIO mask port register (MASK[0:2], address 0x1C00 2080 (MASK0) to 0x1C00 2088 (MASK2)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MASKP	Controls which bits corresponding to PIOm_n are active in the MPORT register (bit 0 = PIOm_0, bit 1 = PIOm_1, ..., bit 31 = PIOm_31). m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port2. 0 = Read MPORT: pin state; write MPORT: load output bit. 1 = Read MPORT: 0; write MPORT: output bit not affected.	0	R/W

10.5.5 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 142. GPIO port pin register (PIN[0:2], address 0x1C00 2100 (PIN0) to 0x1C00 2108 (PIN2)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	PORT	Reads pin states or loads output bits (bit 0 = PIOm_0, bit 1 = PIOm_1, ..., bit 31 = PIOm_31). m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port2. 0 = Read: pin is low; write: clear output bit. 1 = Read: pin is high; write: set output bit.	ext	R/W

10.5.6 GPIO masked port pin registers

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

Table 143. GPIO masked port pin register (MPIN[0:2], address 0x1C00 2180 (MPIN0) to 0x1C00 2188 (MPIN2)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	MPORTP	Masked port register (bit 0 = PIOm_0, bit 1 =PIOm_1, ..., bit 31 = PIOm_31). m = port 0 to 2; n = pin 0 to 31 for port 0 and 1 and pin 0 to 11 for port2. 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

10.5.7 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 144. GPIO set port register (SET[0:2], address 0x1C00 2200 (SET0) to 0x1C00 2208 (SET2)) bit description

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

10.5.8 GPIO port clear registers

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

Table 145. GPIO clear port register (CLR[0:2], 0x1C00 2280 (CLR0) to 0x1C00 2288 (CLR2)) bit description

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

10.5.9 GPIO port toggle registers

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

Table 146. GPIO toggle port register (NOT[0:2], address 0x1C00 2300 (NOT0) to 0x1C00 2308 (NOT2)) bit description

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

10.6 Functional description

10.6.1 Reading pin state

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

- The state of a single pin can be read with 7 high-order zeros from a Byte Pin register.
- The state of a single pin can be read in all bits of a byte, halfword, or word from a Word Pin register.
- The state of multiple pins in a port can be read as a byte, halfword, or word from a PORT register.

- The state of a selected subset of the pins in a port can be read from a Masked Port (MPORT) register. Pins having a 1 in the port's Mask register will read as 0 from its MPORT register.

10.6.2 GPIO output

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

1. The pin must be selected for GPIO operation in the switch matrix (this is the default), and
2. the pin must be selected for output by a 1 in its port's DIR register.

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

There are seven ways to change GPIO output bits:

- Writing to a Byte Pin register loads the output bit from the least significant bit.
- Writing to a Word Pin register loads the output bit with the OR of all of the bits written. (This feature follows the definition of truth of a multi-bit value in programming languages.)
- Writing to a port's PORT register loads the output bits of all the pins written to.
- Writing to a port's MPORT register loads the output bits of pins identified by zeros in corresponding positions of the port's MASK register.
- Writing ones to a port's SET register sets output bits.
- Writing ones to a port's CLR register clears output bits.
- Writing ones to a port's NOT register toggles/complements/inverts output bits.

The state of a port's output bits can be read from its SET register. Reading any of the registers described in 10.6.1 returns the state of pins, regardless of their direction or alternate functions.

10.6.3 Masked I/O

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

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

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

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

10.6.4 Recommended practices

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

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

11.1 How to read this chapter

11.2 Features

- The inputs from any number of digital 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 grouped interrupts can wake up the part from sleep, deep-sleep or power-down modes.

11.3 Basic configuration

For the group interrupt feature, enable the clock to both the GROUP0 and GROUP1 register interfaces in the SYSAHBCLKCTRL0 register ((Table 50, bit 19). The group interrupt wake-up feature is enabled in the STARTERP0 register (Table 76).

The GINT block reads the input from the pin directly bypassing the switch matrix.

Make sure that no analog function is selected on pins that are input to the group interrupts. Selecting an analog function in the switch matrix PINENABLE registers disables the digital pad and the digital signal is tied to 0.

11.4 General description

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.

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

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

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

11.5 Register description

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

Table 147. Register overview: GROUP0 interrupt (base address 0x400A 8000 (GINT0) and 0x400A C000 (GINT1))

Name	Access	Address offset	Description	Reset value	Reference
CTRL	R/W	0x000	GPIO grouped interrupt control register	0	Table 148
PORT_POL0	R/W	0x020	GPIO grouped interrupt port 0 polarity register	0xFFFF FFFF	Table 149
PORT_POL1	R/W	0x024	GPIO grouped interrupt port 1 polarity register	0xFFFF FFFF	Table 149
PORT_POL2	R/W	0x028	GPIO grouped interrupt port 2 polarity register	0xFFFF FFFF	Table 149
PORT_ENA0	R/W	0x040	GPIO grouped interrupt port 0 enable register	0	Table 150
PORT_ENA1	R/W	0x044	GPIO grouped interrupt port 1 enable register	0	Table 150
PORT_ENA2	R/W	0x048	GPIO grouped interrupt port 2 enable register	0	Table 150

11.5.1 Grouped interrupt control register

Table 148. GPIO grouped interrupt control register (CTRL, addresses 0x400A 8000 (GINT0) and 0x400A C000 (GINT1)) 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

11.5.2 GPIO grouped interrupt port polarity registers

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

Each register PORT_POLm controls the polarity of pins in port m.

Table 149. GPIO grouped interrupt port polarity registers (PORT_POL[0:2], addresses 0x400A 8020 (PORT_POL0) to 0x400A 8028 (PORT_POL2) (GINT0) and 0x400A C020 (PORT_POL0) to 0x400A C028 (PORT_POL2) (GINT1)) bit description

Bit	Symbol	Description	Reset value	Access
31:0	POL	Configure pin polarity of port m pins for group interrupt. Bit n corresponds to pin PIOm_n of port m. 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	-

11.5.3 GPIO grouped interrupt port enable registers

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

Each register PORT_ENm enables pins in port m.

Table 150. GPIO grouped interrupt port enable registers (PORT_ENA[0:2], addresses 0x400A 8040 (PORT_ENA0) to 0x400A 8048 (PORT_ENA2) (GINT0) and 0x400A C040 (PORT_ENA0) to 0x400A C048 (PORT_ENA2) (GINT1)) bit description

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

11.6 Functional description

Any subset of the pins in each port can be selected to contribute to a common group interrupt (GINT) and can be enabled to wake the part up from Deep-sleep mode or Power-down mode.

An interrupt can be requested for each port, based on any selected subset of pins within each port. The pins that contribute to each port interrupt are selected by 1s in the port's Enable register, and an interrupt polarity can be selected for each pin in the port's Polarity register. The level on each pin is exclusive-ORed with its polarity bit, and the result is ANDed with its enable bit. 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, or it can be edge-detected by the wake-up interrupt logic (see Table 76).

12.1 How to read this chapter

The pin interrupt generator and the pattern match engine are available on all LPC15xx parts.

12.2 Features

- Pin interrupts
  - Up to eight pins can be selected from all GPIO pins on ports 0 and 1 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.
- Pattern match engine
  - Up to 8 pins can be selected from all digital pins on ports 0 and 1 to contribute to a boolean expression. The boolean expression consists of specified levels and/or transitions on various combinations of these pins.
  - Each bit slice minterm (product term) comprising the specified boolean expression can generate its own, dedicated interrupt request.
  - Any occurrence of a pattern match can be programmed to also generate an RXEV notification to the ARM CPU. The RXEV signal can be connected to a pin.
  - Pattern match can be used, in conjunction with software, to create complex state machines based on pin inputs.

12.3 Basic configuration

- Pin interrupts:
  - Select up to eight external interrupt pins from all digital port pins on ports 0 and 1 in the INMUX block (Table 131). The pin selection process is the same for pin interrupts and the pattern match engine. The two features are mutually exclusive.
  - Enable the clock to the pin interrupt register block in the SYSAHBCLKCTRL0 register (Table 50, bit 18).
  - If you want to use the pin interrupts to wake up the part from deep-sleep mode or power-down mode, enable the pin interrupt wake-up feature in the STARTERP0 register (Table 76).
  - Each selected pin interrupt is assigned to one interrupt in the NVIC (interrupts #7 to #14 for pin interrupts 0 to 7).
- Pattern match engine:
  - Select up to eight external pins from all digital port pins on ports 0 and 1 in the Input mux block (Table 131). The pin selection process is the same for pin interrupts and the pattern match engine. The two features are mutually exclusive.

- Enable the clock to the pin interrupt register block in the SYSAHBCLKCTRL register (Table 50, bit 18).
- Each bit slice of the pattern match engine is assigned to one interrupt in the NVIC (interrupts #7 to #14 for pin interrupts 0 to 7).
- The combined interrupt from all slices or slice combinations can be connected to the ARM RXEV request and to pin function GPIO_INT_BMAT through the switch matrix movable function register (PINASSIGN15, Table 122).

12.3.1 Configure pins as pin interrupts or as inputs to the pattern match engine

Follow these steps to configure pins as pin interrupts:

1. Determine the pins that serve as pin interrupts on the LPC15xx package. See the data sheet for determining the GPIO port pin number associated with the package pin.
2. For each pin interrupt, program the GPIO port pin number from ports 0 and 1 into one of the eight PINTSEL registers in the Input mux block.

   Remark: The port pin number serves to identify the pin to the PINTSEL register. Any function, including GPIO, can be assigned to this pin through the switch matrix.
3. Enable each pin interrupt in the NVIC.

Once the pin interrupts or pattern match inputs are configured, you can set up the pin interrupt detection levels or the pattern match boolean expression.

See Section 9.6.5 “Pin interrupt select registers” in the Input mux block for the PINTSEL registers.

Remark: The inputs to the Pin interrupt select registers bypass the switch matrix. They do not have to be selected as GPIO in the switch matrix. Make sure that no analog function is selected on pins that are input to the PINSELECT registers.

12.4 Pin description

The inputs to the pin interrupt and pattern match engine are determined by the pin interrupt select registers in the Input mux. See Section 9.6.5 “Pin interrupt select registers”.

The pattern match engine output is assigned to an external pin through the switch matrix.

See Section 8.3.1 “Connect an internal signal to a package pin” for the steps that you need to follow to assign the GPIO pattern match function to a pin on the LPC15xx package.

Table 151. GPIO pin interrupt/pattern match pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
GPIO_INT_BMAT	O	external to pin	any pin	PINASSIGN15	Table 122	GPIO pattern match output
8 pin interrupts	I	external to pin	any pin	PINSEL[0:7]	Table 131	External pin interrupts

12.5 General description

Pins with configurable functions can serve as external interrupts or inputs to the pattern match engine. You can configure up to eight pins total using the PINTSEL registers in the Input mux block for these features.

12.5.1 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 131). 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.

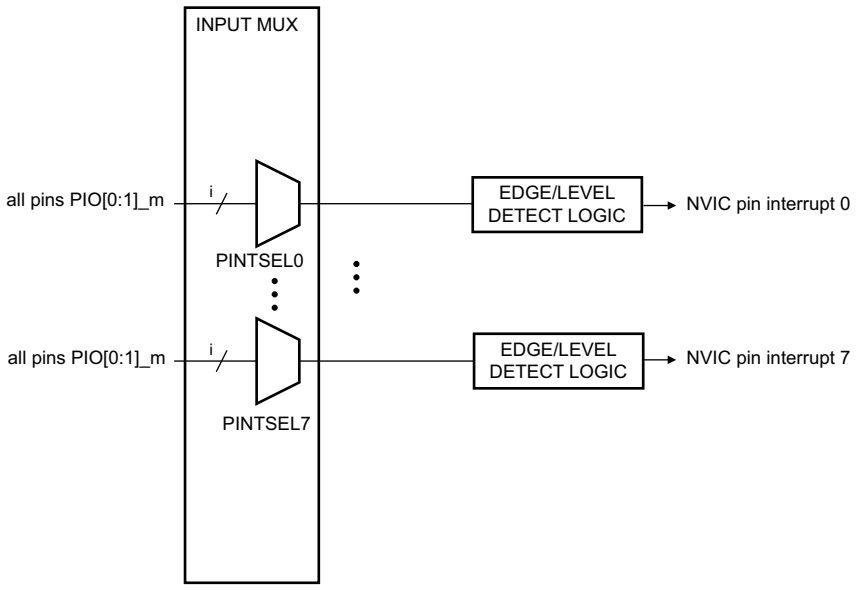


Fig 16. Pin interrupt connections

12.5.2 Pattern match engine

The pattern match feature allows complex boolean expressions to be constructed from the same set of eight GPIO pins that were selected for the GPIO pin interrupts. Each term in the boolean expression is implemented as one slice of the pattern match engine. A slice consists of an input selector and a detect logic that monitors the selected input continuously and creates a HIGH output if the input qualifies as detected, that is as true. Several terms can be combined to a minterm and a pin interrupt is asserted when the minterm evaluates as true.

The detect logic of each slice can detect the following events on the selected input:

- Edge with memory (sticky): A rising edge, a falling edge, or a rising or falling edge that is detected at any time after the edge-detection mechanism has been cleared. The input qualifies as detected (the detect logic output remains HIGH) until the pattern match engine detect logic is cleared again.

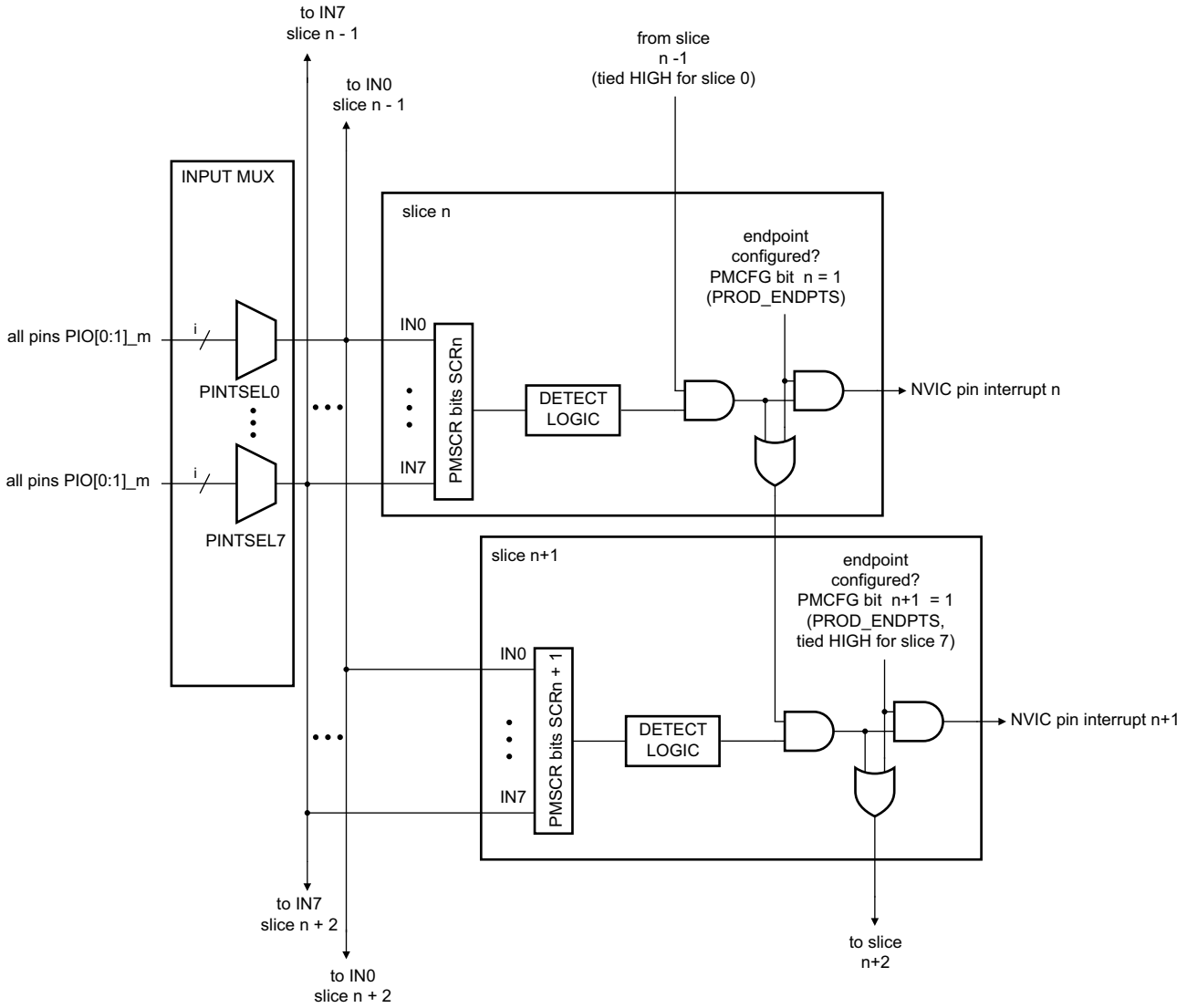
- Event (non-sticky): Every time an edge (rising or falling) is detected, the detect logic output for this pin goes HIGH. This bit is cleared after one clock cycle, and the detect logic can detect another edge,
- Level: A HIGH or LOW level on the selected input.

Figure 18 shows the details of the edge detection logic for each slice.

You can combine a sticky event with non-sticky events to create a pin interrupt whenever a rising or falling edge occurs after a qualifying edge event.

You can create a time window during which rising or falling edges can create a pin interrupt by combining a level detect with an event detect. See Section 12.7.3 for details.

The connections between the pins and the pattern match engine are shown in Figure 17. All pins that are inputs to the pattern match engine are selected in the SYSCON block and can be GPIO port pins or other pin function depending on the switch matrix configuration.



See Figure 18 for the detect logic block.

Fig 17. Pattern match engine connections

The pattern match logic continuously monitors the eight inputs and generates interrupts when any one or more minterms (product terms) of the specified boolean expression is matched. A separate interrupt request is generated for each individual minterm.

In addition, the pattern match module can be enabled to generate a Receive Event (RXEV) output to the ARM core when the entire boolean expression is true (i.e. when any minterm is matched).

The RXEV output is also be routed to GPIO_INT_BMAT pin. This allows the GPIO module to provide a rudimentary programmable logic capability employing up to eight inputs and one output.

The pattern match function utilizes the same eight interrupt request lines as the pin interrupts so these two features are mutually exclusive as far as interrupt generation is concerned. A control bit is provided to select whether interrupt requests are generated in response to the standard pin interrupts or to pattern matches. Note that, if the pin interrupts are selected, the RXEV request to the CPU can still be enabled for pattern matches.

Remark: Pattern matching cannot be used to wake the part up from power-down modes. Pin interrupts must be selected in order to use the GPIO for wake-up.

The pattern match module is constructed of eight bit-slice elements. Each bit slice is programmed to represent one component of one minterm (product term) within the boolean expression. The interrupt request associated with the last bit slice for a particular minterm will be asserted whenever that minterm is matched.
(See bit slice drawing Figure 18).

The pattern match capability can be used to create complex software state machines. Each minterm (and its corresponding individual interrupt) represents a different transition event to a new state. Software can then establish the new set of conditions (that is a new boolean expression) that will cause a transition out of the current state.

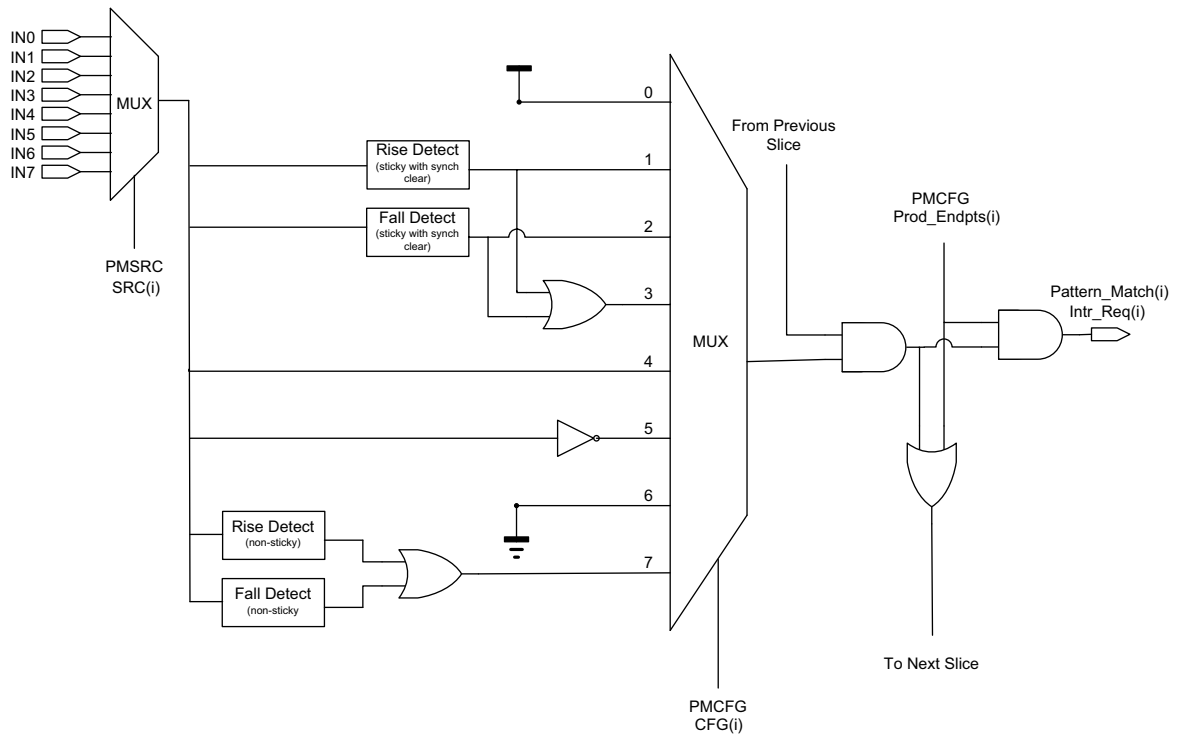


Fig 18. Pattern match bit slice with detect logic

12.5.2.1 Example

Assume the expression: (IN0)~(IN1)(IN3)^ + (IN1)(IN2) + (IN0)~(IN3)~(IN4) is specified through the registers PMSRC (Table 164) and PMCFG (Table 165). Each term in the boolean expression, (IN0), ~(IN1), (IN3)^, etc., represents one bit slice of the pattern match engine.

- In the first minterm (IN0)~(IN1)(IN3)^, bit slice 0 monitors for a high-level on input (IN0), bit slice 1 monitors for a low level on input (IN1) and bit slice 2 monitors for a rising-edge on input (IN3). If this combination is detected, that is if all three terms are true, the interrupt associated with bit slice 2 will be asserted.
- In the second minterm (IN1)(IN2), bit slice 3 monitors input (IN1) for a high level, bit slice 4 monitors input (IN2) for a high level. If this combination is detected, the interrupt associated with bit slice 4 will be asserted.
- In the third minterm (IN0)~(IN3)~(IN4), bit slice 5 monitors input (IN0) for a high level, bit slice 6 monitors input (IN3) for a low level, and bit slice 7 monitors input (IN4) for a low level. If this combination is detected, the interrupt associated with bit slice 7 will be asserted.
- The ORed result of all three minterms asserts the RXEV request to the CPU and the GPIO_INT_BMAT output. That is, if any of the three terms are true, the output is asserted.

Related links:

Section 12.7.2

12.6 Register description

Table 152. Register overview: Pin interrupts/pattern match engine (base address: 0x400A 4000)

Name	Access	Address offset	Description	Reset value	Reference
ISEL	R/W	0x000	Pin Interrupt Mode register	0	Table 153
IENR	R/W	0x004	Pin interrupt level or rising edge interrupt enable register	0	Table 154
SIENR	WO	0x008	Pin interrupt level or rising edge interrupt set register	NA	Table 155
CIENR	WO	0x00C	Pin interrupt level (rising edge interrupt) clear register	NA	Table 156
IENF	R/W	0x010	Pin interrupt active level or falling edge interrupt enable register	0	Table 157
SIENF	WO	0x014	Pin interrupt active level or falling edge interrupt set register	NA	Table 158
CIENF	WO	0x018	Pin interrupt active level or falling edge interrupt clear register	NA	Table 159
RISE	R/W	0x01C	Pin interrupt rising edge register	0	Table 160
FALL	R/W	0x020	Pin interrupt falling edge register	0	Table 161
IST	R/W	0x024	Pin interrupt status register	0	Table 162
PMCTRL	R/W	0x028	Pattern match interrupt control register	0	Table 163
PMSRC	R/W	0x02C	Pattern match interrupt bit-slice source register	0	Table 164
PMCFG	R/W	0x030	Pattern match interrupt bit slice configuration register	0	Table 165

12.6.1 Pin interrupt mode register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), one bit in the ISEL register determines whether the interrupt is edge or level sensitive.

Table 153. Pin interrupt mode register (ISEL, address 0x400A 4000) 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.	-	-

12.6.2 Pin interrupt level or rising edge interrupt enable register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), 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 154. Pin interrupt level or rising edge interrupt enable register (IENR, address 0x400A 4004) 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.	-	-

12.6.3 Pin interrupt level or rising edge interrupt set register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), 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 155. Pin interrupt level or rising edge interrupt set register (SIENR, address 0x400A 4008) 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.	-	-

12.6.4 Pin interrupt level or rising edge interrupt clear register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), 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 156. Pin interrupt level or rising edge interrupt clear register (CIENR, address 0x400A 400C) 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.	-	-

12.6.5 Pin interrupt active level or falling edge interrupt enable register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), one bit in the IENF register enables the falling edge interrupt or the configures the level sensitivity depending on the pin interrupt mode configured in the ISEL register:

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

Table 157. Pin interrupt active level or falling edge interrupt enable register (IENF, address 0x400A 4010) 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.	-	-

12.6.6 Pin interrupt active level or falling edge interrupt set register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), 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 158. Pin interrupt active level or falling edge interrupt set register (SIENF, address 0x400A 4014) 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.	-	-

12.6.7 Pin interrupt active level or falling edge interrupt clear register

For each of the 8 pin interrupts selected in the PINTSELn registers (see Table 131), 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 159. Pin interrupt active level or falling edge interrupt clear register (CIENF, address 0x400A 4018) bit description

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

12.6.8 Pin interrupt rising edge register

This register contains ones for pin interrupts selected in the PINTSELn registers (see Table 131) 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 160. Pin interrupt rising edge register (RISE, address 0x400A 401C) 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.	-	-

12.6.9 Pin interrupt falling edge register

This register contains ones for pin interrupts selected in the PINTSELn registers (see Table 131) 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 161. Pin interrupt falling edge register (FALL, address 0x400A 4020) 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.	-	-

12.6.10 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 162. Pin interrupt status register (IST, address 0x400A 4024) 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.	-	-

12.6.11 Pattern Match Interrupt Control Register

The pattern match control register contains one bit to select pattern-match interrupt generation (as opposed to pin interrupts which share the same interrupt request lines), and another to enable the RXEV output to the cpu. This register also allows the current state of any pattern matches to be read.

If the pattern match feature is not used (either for interrupt generation or for RXEV assertion) bits SEL_PMATCH and ENA_RXEV of this register should be left at 0 to conserve power.

Remark: Set up the pattern-match configuration in the PMSRC and PMCFG registers before writing to this register to enable (or re-enable) the pattern-match functionality. This eliminates the possibility of spurious interrupts as the feature is being enabled.

Table 163. Pattern match interrupt control register (PMCTRL, address 0x400A 4028) bit description

Bit	Symbol	Value	Description	Reset value
0	SEL_PMATCH		Specifies whether the 8 pin interrupts are controlled by the pin interrupt function or by the pattern match function.	0
		0	Pin interrupt. Interrupts are driven in response to the standard pin interrupt function	
		1	Pattern match. Interrupts are driven in response to pattern matches.	

Table 163. Pattern match interrupt control register (PMCTRL, address 0x400A 4028) bit description

Bit	Symbol	Value	Description	Reset value
1	ENA_RXEV		Enables the RXEV output to the ARM cpu and/or to a GPIO output when the specified boolean expression evaluates to true.	0
		0	Disabled. RXEV output to the cpu is disabled.	
		1	Enabled. RXEV output to the cpu is enabled.	
23:2	-		Reserved. Do not write 1s to unused bits.	0
31:24	PMAT	-	This field displays the current state of pattern matches. A 1 in any bit of this field indicates that the corresponding product term is matched by the current state of the appropriate inputs.	0x0

12.6.12 Pattern Match Interrupt Bit-Slice Source register

The bit-slice source register specifies the input source for each of the eight pattern match bit slices.

Each of the possible eight inputs is selected in the pin interrupt select registers, see Table 131. Input 0 corresponds to the pin selected in the PINTSEL0 register, input 1 corresponds to the pin selected in the PINTSEL1 register, and so forth.

Remark: Writing any value to either the PMCFG register or the PMSRC register, or disabling the pattern-match feature (by clearing both the SEL_PMATCH and ENA_RXEV bits in the PMCTRL register to zeros) will erase all edge-detect history.

Table 164. Pattern match bit-slice source register (PMSRC, address 0x4008 402C) bit description

Bit	Symbol	Value	Description	Reset value
7:0	Reserved		Software should not write 1s to unused bits.	0
10:8	SRC0		Selects the input source for bit slice 0	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	

Table 164. Pattern match bit-slice source register (PMSRC, address 0x4008 402C) bit description

Bit	Symbol	Value	Description	Reset value
13:11	SRC1		Selects the input source for bit slice 1	0
		0x0	Input 0. Selects pin interrupt input 0 as the source to bit slice 1.	
		0x1	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x2	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x3	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x4	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x5	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x6	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x7	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
16:14	SRC2		Selects the input source for bit slice 2	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	

Table 164. Pattern match bit-slice source register (PMSRC, address 0x4008 402C) bit description

Bit	Symbol	Value	Description	Reset value
19:17	SRC3		Selects the input source for bit slice 3	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	
22:20	SRC4		Selects the input source for bit slice 4	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	

Table 164. Pattern match bit-slice source register (PMSRC, address 0x4008 402C) bit description

Bit	Symbol	Value	Description	Reset value
25:23	SRC5		Selects the input source for bit slice 5	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	
28:26	SRC6		Selects the input source for bit slice 6	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	

Table 164. Pattern match bit-slice source register (PMSRC, address 0x4008 402C) bit description

Bit	Symbol	Value	Description	Reset value
31:29	SRC7		Selects the input source for bit slice 7	0
		0x0	Input 0. Selects the pin selected in the PINTSEL0 register as the source to bit slice 0.	
		0x1	Input 1. Selects the pin selected in the PINTSEL1 register as the source to bit slice 0.	
		0x2	Input 2. Selects the pin selected in the PINTSEL2 register as the source to bit slice 0.	
		0x3	Input 3. Selects the pin selected in the PINTSEL3 register as the source to bit slice 0.	
		0x4	Input 4. Selects the pin selected in the PINTSEL4 register as the source to bit slice 0.	
		0x5	Input 5. Selects the pin selected in the PINTSEL5 register as the source to bit slice 0.	
		0x6	Input 6. Selects the pin selected in the PINTSEL6 register as the source to bit slice 0.	
		0x7	Input 7. Selects the pin selected in the PINTSEL7 register as the source to bit slice 0.	

12.6.13 Pattern Match Interrupt Bit Slice Configuration register

The bit-slice configuration register configures the detect logic and contains bits to select from among eight alternative conditions for each bit slice that cause that bit slice to contribute to a pattern match. The seven LSBs of this register specify which bit-slices are the end-points of product terms in the boolean expression (i.e. where OR terms are to be inserted in the expression).

Two types of edge detection on each input are possible:

- Sticky: A rising edge, a falling edge, or a rising or falling edge that is detected at any time after the edge-detection mechanism has been cleared. The input qualifies as detected (the detect logic output remains HIGH) until the pattern match engine detect logic is cleared again.
- Non-sticky: Every time an edge (rising or falling) is detected, the detect logic output for this pin goes HIGH. This bit is cleared after one clock cycle, and the edge detect logic can detect another edge,

Remark: To clear the pattern match engine detect logic, write any value to either the PMCFG register or the PMSRC register, or disable the pattern-match feature (by clearing both the SEL_PMATCH and ENA_RXEV bits in the PMCTRL register to zeros). This will erase all edge-detect history.

To select whether a slice marks the final component in a minterm of the boolean expression, write a 1 in the corresponding PROD_ENPTSn bit. Setting a term as the final component has two effects:

1. The interrupt request associated with this bit slice will be asserted whenever a match to that product term is detected.
2. The next bit slice will start a new, independent product term in the boolean expression (i.e. an OR will be inserted in the boolean expression following the element controlled by this bit slice).

Table 165. Pattern match bit slice configuration register (PMCFG, address 0x400A 4030) bit description

Bit	Symbol	Value	Description	Reset value
0	PROD_ENDPTS0		Determines whether slice 0 is an endpoint.	0
		0	No effect. Slice 0 is not an endpoint.	
		1	endpoint. Slice 0 is the endpoint of a product term (minterm). Pin interrupt 0 in the NVIC is raised if the minterm evaluates as true.	
1	PROD_ENDPTS1		Determines whether slice 1 is an endpoint.	0
		0	No effect. Slice 1 is not an endpoint.	
		1	endpoint. Slice 1 is the endpoint of a product term (minterm). Pin interrupt 1 in the NVIC is raised if the minterm evaluates as true.	
2	PROD_ENDPTS2		Determines whether slice 2 is an endpoint.	0
		0	No effect. Slice 2 is not an endpoint.	
		1	endpoint. Slice 2 is the endpoint of a product term (minterm). Pin interrupt 2 in the NVIC is raised if the minterm evaluates as true.	
3	PROD_ENDPTS3		Determines whether slice 3 is an endpoint.	0
		0	No effect. Slice 3 is not an endpoint.	
		1	endpoint. Slice 3 is the endpoint of a product term (minterm). Pin interrupt 3 in the NVIC is raised if the minterm evaluates as true.	
4	PROD_ENDPTS4		Determines whether slice 4 is an endpoint.	0
		0	No effect. Slice 4 is not an endpoint.	
		1	endpoint. Slice 4 is the endpoint of a product term (minterm). Pin interrupt 4 in the NVIC is raised if the minterm evaluates as true.	
5	PROD_ENDPTS5		Determines whether slice 5 is an endpoint.	0
		0	No effect. Slice 5 is not an endpoint.	
		1	endpoint. Slice 5 is the endpoint of a product term (minterm). Pin interrupt 5 in the NVIC is raised if the minterm evaluates as true.	
6	PROD_ENDPTS6		Determines whether slice 6 is an endpoint.	0
		0	No effect. Slice 6 is not an endpoint.	
		1	endpoint. Slice 6 is the endpoint of a product term (minterm). Pin interrupt 6 in the NVIC is raised if the minterm evaluates as true.	
7	-		Reserved. Bit slice 7 is automatically considered a product end point.	0

Table 165. Pattern match bit slice configuration register (PMCFG, address 0x400A 4030) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
10:8	CFG0		Specifies the match contribution condition for bit slice 0.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	
13:11	CFG1		Specifies the match contribution condition for bit slice 1.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	

Table 165. Pattern match bit slice configuration register (PMCFG, address 0x400A 4030) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
16:14	CFG2		Specifies the match contribution condition for bit slice 2.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	
19:17	CFG3		Specifies the match contribution condition for bit slice 3.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	

Table 165. Pattern match bit slice configuration register (PMCFG, address 0x400A 4030) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
22:20	CFG4		Specifies the match contribution condition for bit slice 4.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	
25:23	CFG5		Specifies the match contribution condition for bit slice 5.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	

Table 165. Pattern match bit slice configuration register (PMCFG, address 0x400A 4030) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
28:26	CFG6		Specifies the match contribution condition for bit slice 6.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	
31:29	CFG7		Specifies the match contribution condition for bit slice 7.	0b000
		0x0	Constant HIGH. This bit slice always contributes to a product term match.	
		0x1	Sticky rising edge. Match occurs if a rising edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x2	Sticky falling edge. Match occurs if a falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x3	Sticky rising or falling edge. Match occurs if either a rising or falling edge on the specified input has occurred since the last time the edge detection for this bit slice was cleared. This bit is only cleared when the PMCFG or the PMSRC registers are written to.	
		0x4	High level. Match (for this bit slice) occurs when there is a high level on the input specified for this bit slice in the PMSRC register.	
		0x5	Low level. Match occurs when there is a low level on the specified input.	
		0x6	Constant 0. This bit slice never contributes to a match (should be used to disable any unused bit slices).	
		0x7	Event. Non-sticky rising or falling edge. Match occurs on an event - i.e. when either a rising or falling edge is first detected on the specified input (this is a non-sticky version of value 0x3). This bit is cleared after one clock cycle.	

12.7 Functional description

12.7.1 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 166.

Table 166. 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.
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.

12.7.2 Pattern Match engine example

Suppose the desired boolean pattern to be matched is:
(IN1) + (IN1 * IN2) + (~IN2 * ~IN3 * IN6fe) + (IN5 * IN7ev)

with:

IN6fe = (sticky) falling-edge on input 6

IN7ev = (non-sticky) event (rising or falling edge) on input 7

Each individual term in the expression shown above is controlled by one bit-slice. To specify this expression, program the pattern match bit slice source and configuration register fields as follows:

- PMSRC register (Table 164):
  - Since bit slice 5 will be used to detect a sticky event on input 6, you can write a 1 to the SRC5 bits to clear any pre-existing edge detects on bit slice 5.
  - SRC0: 001 - select input 1 for bit slice 0
  - SRC1: 001 - select input 1 for bit slice 1
  - SRC2: 010 - select input 2 for bit slice 2
  - SRC3: 010 - select input 2 for bit slice 3
  - SRC4: 011 - select input 3 for bit slice 4
  - SRC5: 110 - select input 6 for bit slice 5
  - SRC6: 101 - select input 5 for bit slice 6

- SRC7: 111 - select input 7 for bit slice 7

- PMCFG register (Table 165):
  - PROD_ENDPTS0 = 1
  - PROD_ENDPTS02 = 1
  - PROD_ENDPTS5 = 1
  - All other slices are not product term endpoints and their PROD_ENDPTS bits are 0. Slice 7 is always a product term endpoint and does not have a register bit associated with it.
  - = 0100101 - bit slices 0, 2, 5, and 7 are product-term endpoints. (Bit slice 7 is an endpoint by default - no associated register bit).
  - CFG0: 000 - high level on the selected input (input 1) for bit slice 0
  - CFG1: 000 - high level on the selected input (input 1) for bit slice 1
  - CFG2: 000 - high level on the selected input (input 2) for bit slice 2
  - CFG3: 101 - low level on the selected input (input 2) for bit slice 3
  - CFG4: 101 - low level on the selected input (input 3) for bit slice 4
  - CFG5: 010 - (sticky) falling edge on the selected input (input 6) for bit slice 5
  - CFG6: 000 - high level on the selected input (input 5) for bit slice 6
  - CFG7: 111 - event (any edge, non-sticky) on the selected input (input 7) for bit slice 7
- PMCTRL register (Table 163):
  - Bit0: Setting this bit will select pattern matches to generate the pin interrupts in place of the normal pin interrupt mechanism.

    For this example, pin interrupt 0 will be asserted when a match is detected on the first product term (which, in this case, is just a high level on input 1).

    Pin interrupt 2 will be asserted in response to a match on the second product term.

    Pin interrupt 5 will be asserted when there is a match on the third product term.

    Pin interrupt 7 will be asserted on a match on the last term.
  - Bit1: Setting this bit will cause the RxEv signal to the ARM CPU to be asserted whenever a match occurs on ANY of the product terms in the expression. Otherwise, the RXEV line will not be used.
  - Bit31:24: At any given time, bits 0, 2, 5 and/or 7 may be high if the corresponding product terms are currently matching.
  - The remaining bits will always be low.

12.7.3 Pattern match engine edge detect examples

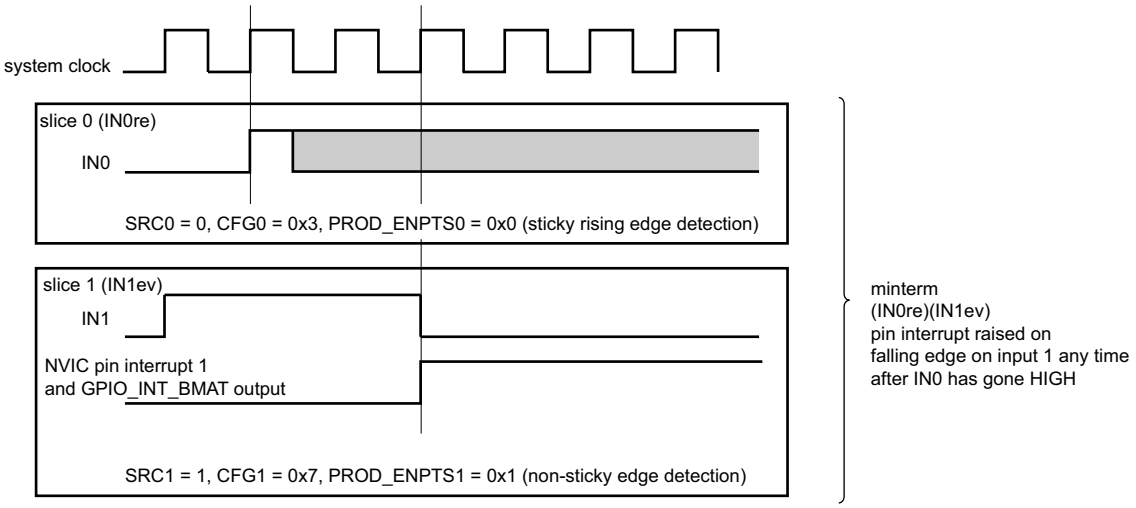


Figure shows pattern match functionality only and accurate timing is not implied. Inputs (INn) are shown synchronized to the system clock for simplicity.

Fig 19. Pattern match engine examples: sticky edge detect

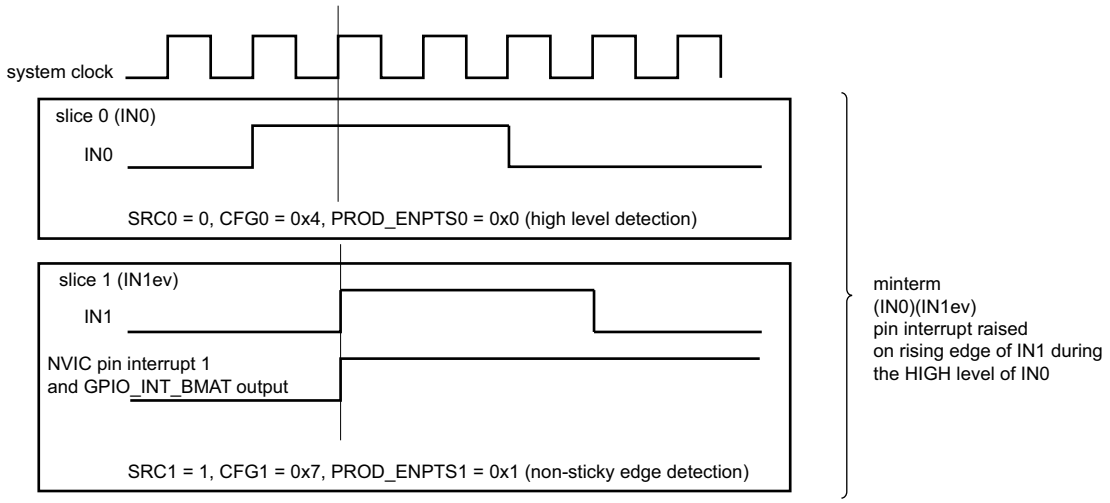


Figure shows pattern match functionality only and accurate timing is not implied. Inputs (INn) are shown synchronized to the system clock for simplicity.

Fig 20. Pattern match engine examples: Windowed non-sticky edge detect evaluates as true

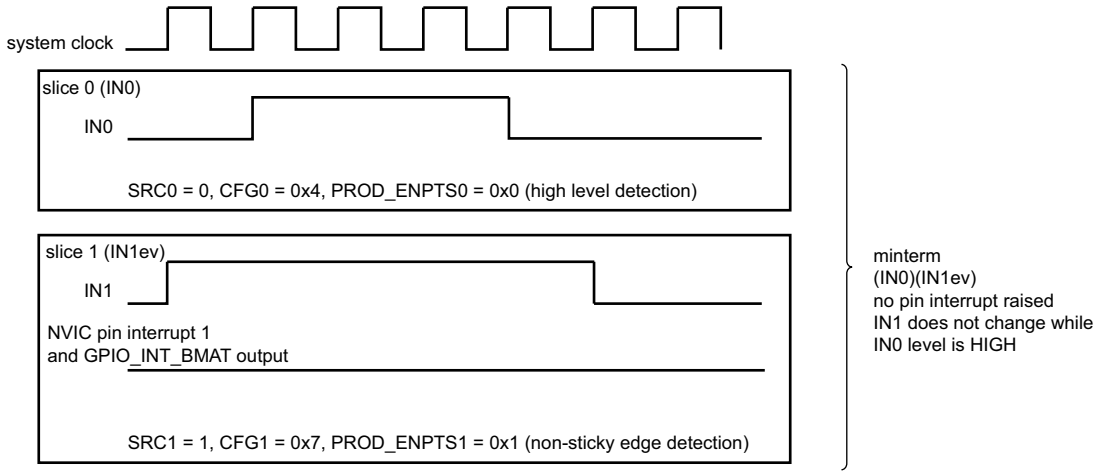


Figure shows pattern match functionality only and accurate timing is not implied. Inputs (INn) are shown synchronized to the system clock for simplicity.

Fig 21. Pattern match engine examples: Windowed non-sticky edge detect evaluates as false

13.1 How to read this chapter

The DMA controller is available on all parts. For API support see Chapter 37 “LPC15xx DMA API ROM driver routines”.

13.2 Features

- 18 channels supported with 14 channels connected to peripheral requests of the USART, SPI, I2C, and DAC peripherals. Four channels have no DMA request connected.
- DMA operations can be triggered by on- or off-chip events. Each DMA channel can select one trigger input from 20 sources. Trigger sources are the ADC interrupt, the analog comparator outputs, and the SCT DMA request lines.
- Priority is user selectable for each channel (up to eight priority levels).
- Continuous priority arbitration.
- Address cache with four entries (each entry is a pair of addresses).
- Efficient use of data bus.
- Supports single transfers up to 1,024 words.
- Address increment options allow packing and/or unpacking data.

13.3 Basic configuration

Configure the DMA as follows:

- Use the SYSAHBCLKCTRL0 register (Table 50) to enable the clock to the DMA registers interface.
- Clear the DMA peripheral reset using the PRESETCTRL0 register (Table 35).
- The DMA interrupt is connected to slot #4 in the NVIC.
- Each DMA channel has one DMA request line associated and can also select one of 20 input triggers through the input mux registers DMA_ITRIG_INMUX[0:17].
- Trigger outputs are connected to DMA_INMUX_INMUX[0:3] as inputs to DMA triggers.

For details on the trigger input and output multiplexing, see Section 9.5.3 “DMA trigger input multiplexing”.

13.3.1 Hardware triggers

Each DMA channel can use one trigger that is independent of the request input for this channel. The trigger input is selected in the DMA_ITRIG_INMUX registers. There are 20 possible internal trigger sources for each channel with each trigger signal issued by the output of a peripheral. In addition, the DMA trigger output can be routed to the trigger input of another channel through the trigger input multiplexing. See Section 9.5.3 “DMA trigger input multiplexing”.

See Table 167 for the connection of input muxes to DMA channels.

See Table 132 for a list of possible trigger input sources.

13.3.2 Trigger outputs

Each channel of the DMA controller provides a trigger output. This allows the possibility of using the trigger outputs as a trigger source to a different channel in order to support complex transfers on selected peripherals. This kind of transfer can, for example, use more than one peripheral DMA request. An example use would be to input data to a holding buffer from one peripheral, and then output the data to another peripheral, with both transfers being paced by the appropriate peripheral DMA request. This kind of operation is called “chained operation” or “channel chaining”.

13.3.3 DMA requests

DMA requests are directly connected to the peripherals. Each channel supports one DMA request line and one trigger input which is multiplexed to many possible input sources.

For each trigger mux DMA_ITRIG_INMUXn, the following sources are supported:

- ADC0 sequence A interrupt ADC0_SEQA_IRQ
- ADC0 sequence B interrupt ADC0_SEQB_IRQ
- ADC1 sequence A interrupt ADC1_SEQA_IRQ
- ADC1 sequence B interrupt ADC1_SEQB_IRQ
- SCT0 DMA request 0 SCT0_DMA0
- SCT0 DMA request 1 SCT0_DMA1
- SCT1 DMA request 0 SCT1_DMA0
- SCT1 DMA request 1 SCT1_DMA1
- SCT2 DMA request 0 SCT2_DMA0
- SCT2 DMA request 1 SCT2_DMA1
- SCT3 DMA request 0 SCT3_DMA0
- SCT3 DMA request 1 SCT3_DMA1
- Output from analog comparator 0 ACMP0_OUT
- Output from analog comparator 1 ACMP1_OUT
- Output from analog comparator 2 ACMP2_OUT
- Output from analog comparator 3 ACMP3_OUT
- Four choices of one of the DMA output triggers

Table 167. DMA requests

DMA channel #	Request input	DMA trigger mux
0	USART0_RX_DMA	DMA_ITRIG_INMUX0
1	USART0_TX_DMA	DMA_ITRIG_INMUX1
2	USART1_RX_DMA	DMA_ITRIG_INMUX2
3	USART1_TX_DMA	DMA_ITRIG_INMUX3

Table 167. DMA requests

DMA channel #	Request input	DMA trigger mux
4	USART2_RX_DMA	DMA_ITRIG_INMUX4
5	USART2_TX_DMA	DMA_ITRIG_INMUX5
6	SPI0_RX_DMA	DMA_ITRIG_INMUX6
7	SPI0_TX_DMA	DMA_ITRIG_INMUX7
8	SPI1_RX_DMA	DMA_ITRIG_INMUX8
9	SPI1_TX_DMA	DMA_ITRIG_INMUX9
10	I2C0_SLV_DMA	DMA_ITRIG_INMUX10
11	I2C0_MST_DMA	DMA_ITRIG_INMUX11
12	I2C0_MONITOR_DMA	DMA_ITRIG_INMUX12
13	DAC_IRQ	DMA_ITRIG_INMUX13
14	-	DMA_ITRIG_INMUX14
15	-	DMA_ITRIG_INMUX15
16	-	DMA_ITRIG_INMUX16
17	-	DMA_ITRIG_INMUX17

13.3.4 DMA in sleep mode

The DMA can operate and access all SRAM blocks in sleep mode.

13.4 Pin description

The DMA controller has no configurable pins.

13.5 General description

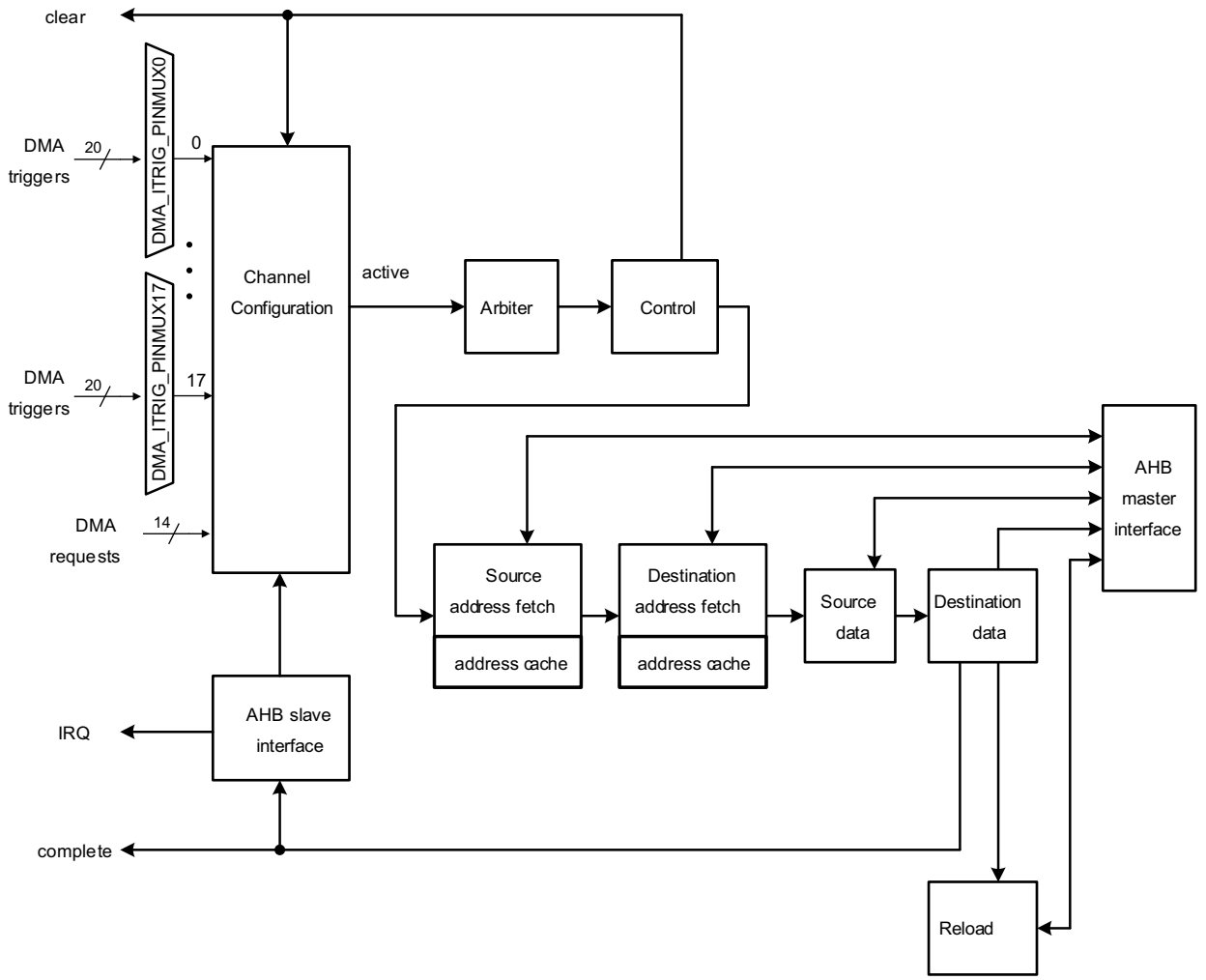


Fig 22. DMA block diagram

13.5.1 DMA requests and triggers

An operation on a DMA channel can be initiated by either a DMA request or a trigger event. DMA requests come from peripherals and specifically indicate when a peripheral either needs input data to be read from it, or that output data may be sent to it. DMA requests are created by the UART, SPI, and I2C peripherals.

A trigger initiates a DMA operation and can be a signal from an unrelated peripheral. Peripherals that generate triggers are the SCTs, the ADCs, and the analog comparators. In addition, the DMA triggers also create an trigger output that can trigger DMA transactions on another channel. Triggers can be used to send a character or a string to a UART or other serial output at a fixed time interval or when an event occurs.

A DMA channel using a trigger can respond by moving data from any memory address to any other memory address. This can include fixed peripheral data registers, or incrementing through RAM buffers. The amount of data moved by a single trigger event

can range from a single transfer to many transfers. A transfer that is started by a trigger can still be paced using the channel's DMA request. This allows sending a string to a serial peripheral, for instance, without overrunning the peripheral's transmit buffer.

Each trigger input to the DMA has a corresponding output that can be used as a trigger input to another channel. The trigger outputs appear in the trigger source list for each channel and can be selected through the DMA_INMUX registers as inputs to other channels.

13.5.2 DMA Modes

The DMA controller doesn't really have separate operating modes, but there are ways of using the DMA controller that have commonly used terminology in the industry.

Once the DMA controller is set up for operation, using any specific DMA channel requires initializing the registers associated with that channel (see Table 167), and supplying at least the channel descriptor, which is located somewhere in memory, typically in on-chip SRAM (see Section 13.6.3). The channel descriptor is shown in Table 168.

Table 168: Channel descriptor

Offset	Description
+ 0x0	Reserved
+ 0x4	Source data end address
+ 0x8	Destination end address
+ 0xC	Link to next descriptor

The source and destination end addresses, as well as the link to the next descriptor are just memory addresses that can point to any valid address on the device. The starting address for both source and destination data is the specified end address minus the transfer length (XFERCOUNT * the address increment as defined by SRCINC and DSTINC). The link to the next descriptor is used only if it is a linked transfer.

After the channel has had a sufficient number of DMA requests and/or triggers, depending on its configuration, the initial descriptor will be exhausted. At that point, if the transfer configuration directs it, the channel descriptor will be reloaded with data from memory pointed to by the "Link to next descriptor" entry of the initial channel descriptor. Descriptors loaded in this manner look slightly different the channel descriptor, as shown in Table 169. The difference is that a new transfer configuration is specified in the reload descriptor instead of being written to the XFERCFG register for that channel.

This process repeats as each descriptor is exhausted as long as reload is selected in the transfer configuration for each new descriptor.

Table 169: Reload descriptors

Offset	Description
+ 0x0	Transfer configuration.

Table 169: Reload descriptors

Offset	Description
+ 0x4	Source end address. This points to the address of the last entry of the source address range if the address is incremented. The address to be used in the transfer is calculated from the end address, data width, and transfer size.
+ 0x8	Destination end address. This points to the address of the last entry of the destination address range if the address is incremented. The address to be used in the transfer is calculated from the end address, data width, and transfer size.
+ 0xC	Link to next descriptor. If used, this address must be aligned to a multiple of 16 bytes (i.e., the size of a descriptor).

13.5.3 Single buffer

This generally applies to memory to memory moves, and peripheral DMA that occurs only occasionally and is set up for each transfer. For this kind of operation, only the initial channel descriptor shown in Table 170 is needed.

Table 170: Channel descriptor for a single transfer

Offset	Description
+ 0x0	Reserved
+ 0x4	Source data end address
+ 0x8	Destination data end address
+ 0xC	(not used)

This case is identified by the Reload bit in the XFERCFG register = 0. When the DMA channel receives a DMA request or trigger (depending on how it is configured), it performs one or more transfers as configured, then stops. Once the channel descriptor is exhausted, additional DMA requests or triggers will have no effect until the channel configuration is updated by software.

13.5.4 Ping-Pong

Ping-pong is a special case of a linked transfer. It is described separately because it is typically used more frequently than more complicated versions of linked transfers.

A ping-pong transfer uses two buffers alternately. At any one time, one buffer is being loaded or unloaded by DMA operations. The other buffer has the opposite operation being handled by software, readying the buffer for use when the buffer currently being used by the DMA controller is full or empty. Table 171 shows an example of descriptors for ping-pong from a peripheral to two buffers in memory.

Table 171: Example descriptors for ping-pong operation: peripheral to buffer

Channel Descriptor		Descriptor B		Descriptor A	
+ 0x0	(not used)	+ 0x0	Buffer B transfer configuration	+ 0x0	Buffer A transfer configuration
+ 0x4	Peripheral data end address	+ 0x4	Peripheral data end address	+ 0x4	Peripheral data end address
+ 0x8	Buffer A memory end address	+ 0x8	Buffer B memory end address	+ 0x8	Buffer A memory end address
+ 0xC	Address of descriptor B	+ 0xC	Address of descriptor A	+ 0xC	Address of descriptor B

In this example, the channel descriptor is used first, with a first buffer in memory called buffer A. The configuration of the DMA channel must have been set to indicate a reload. Similarly, both descriptor A and descriptor B must also specify reload. When the channel descriptor is exhausted, descriptor B is loaded using the link to descriptor B, and a transfer interrupt informs the CPU that buffer A is available.

Descriptor B is then used until it is also exhausted, when descriptor A is loaded using the link to descriptor A contained in descriptor B. Then a transfer interrupt informs the CPU that buffer B is available for processing. The process repeats when descriptor A is exhausted, alternately using each of the 2 memory buffers.

13.5.5 Linked transfers (linked list)

A linked transfer can use any number of descriptors to define a complicated transfer. This can be configured such that a single transfer, a portion of a transfer, one whole descriptor, or an entire structure of links can be initiated by a single DMA request or trigger.

An example of a linked transfer could start out like the example for a ping-pong transfer (Table 171). The difference would be that descriptor B would not link back to descriptor A, but would continue on to another different descriptor. This could continue as long as desired, and can be ended anywhere, or linked back to any point to repeat a sequence of descriptors. Of course, any descriptor not currently in use can be altered by software as well.

13.5.6 Address alignment for data transfers

Transfers of 16 bit width require an address alignment to a multiple of 2 bytes. Transfers of 32 bit width require an address alignment to a multiple of 4 bytes. Transfers of 8 bit width can be at any address.

13.6 Register description

The DMA registers are grouped into DMA control, interrupt and status registers and DMA channel registers. DMA transfers are controlled by a set of three registers per channel, the CFG[0:20], CTRLSTAT[0:20], and XFERCFG[0:20] registers.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

Table 172. Register overview: DMA controller (base address 0x1C00 4000)

Name	Access	Address offset	Description	Reset Value	Reference
Global control and status registers					
CTRL	R/W	0x000	DMA control.	0	Table 173
INTSTAT	RO	0x004	Interrupt status.	0	Table 174
SRAMBASE	R/W	0x008	SRAM address of the channel configuration table.	0	Table 175
Shared registers					
ENABLESET0	RO/W1	0x020	Channel Enable read and Set for all DMA channels.	0	Table 177
ENABLECLR0	W1	0x028	Channel Enable Clear for all DMA channels.	NA	Table 178
ACTIVE0	RO	0x030	Channel Active status for all DMA channels.	0	Table 179
BUSY0	RO	0x038	Channel Busy status for all DMA channels.	0	Table 180
ERRINT0	RO/W1	0x040	Error Interrupt status for all DMA channels.	0	Table 181
INTENSET0	RO/W1	0x048	Interrupt Enable read and Set for all DMA channels.	0	Table 182
INTENCLR0	W1	0x050	Interrupt Enable Clear for all DMA channels.	NA	Table 183
INTA0	RO/W1	0x058	Interrupt A status for all DMA channels.	0	Table 184
INTB0	RO/W1	0x060	Interrupt B status for all DMA channels.	0	Table 185
SETVALID0	W1	0x068	Set ValidPending control bits for all DMA channels.	NA	Table 186
SETTRIG0	W1	0x070	Set Trigger control bits for all DMA channels.	NA	Table 187
ABORT0	W1	0x078	Channel Abort control for all DMA channels.	NA	Table 188
Channel0 registers					
CFG0	R/W	0x400	Configuration register for DMA channel 0.		Table 189
CTLSTAT0	RO	0x404	Control and status register for DMA channel 0.		Table 191
XFERCFG0	R/W	0x408	Transfer configuration register for DMA channel 0.		Table 192
Channel1 registers					
CFG1	R/W	0x410	Configuration register for DMA channel 1.		Table 189
CTLSTAT1	RO	0x414	Control and status register for DMA channel 1.		Table 191
XFERCFG1	R/W	0x418	Transfer configuration register for DMA channel 1.		Table 192
Channel2 registers					
CFG2	R/W	0x420	Configuration register for DMA channel 2.		Table 189
CTLSTAT2	RO	0x424	Control and status register for DMA channel 2.		Table 191
XFERCFG2	R/W	0x428	Transfer configuration register for DMA channel 2.		Table 192
Channel3 registers					
CFG3	R/W	0x430	Configuration register for DMA channel 3.		Table 189
CTLSTAT3	RO	0x434	Control and status register for DMA channel 3.		Table 191
XFERCFG3	R/W	0x438	Transfer configuration register for DMA channel 3.		Table 192
Channel4 registers					
CFG4	R/W	0x440	Configuration register for DMA channel 4.		Table 189
CTLSTAT4	RO	0x444	Control and status register for DMA channel 4.		Table 191
XFERCFG4	R/W	0x448	Transfer configuration register for DMA channel 4.		Table 192
Channel5 registers					
CFG5	R/W	0x450	Configuration register for DMA channel 5.		Table 189
CTLSTAT5	RO	0x454	Control and status register for DMA channel 5.		Table 191

Table 172. Register overview: DMA controller (base address 0x1C00 4000)

Name	Access	Address offset	Description	Reset Value	Reference
XFERCFG5	R/W	0x458	Transfer configuration register for DMA channel 5.		Table 192
Channel6 registers					
CFG6	R/W	0x460	Configuration register for DMA channel 6.		Table 189
CTLSTAT6	RO	0x464	Control and status register for DMA channel 6.		Table 191
XFERCFG6	R/W	0x468	Transfer configuration register for DMA channel 6.		Table 192
Channel7 registers					
CFG7	R/W	0x470	Configuration register for DMA channel 7.		Table 189
CTLSTAT7	RO	0x474	Control and status register for DMA channel 7.		Table 191
XFERCFG7	R/W	0x478	Transfer configuration register for DMA channel 7.		Table 192
Channel8 registers					
CFG8	R/W	0x480	Configuration register for DMA channel 8.		Table 189
CTLSTAT8	RO	0x484	Control and status register for DMA channel 8.		Table 191
XFERCFG8	R/W	0x488	Transfer configuration register for DMA channel 8.		Table 192
Channel9 registers					
CFG9	R/W	0x490	Configuration register for DMA channel 9.		Table 189
CTLSTAT9	RO	0x494	Control and status register for DMA channel 9.		Table 191
XFERCFG9	R/W	0x498	Transfer configuration register for DMA channel 9.		Table 192
Channel10 registers					
CFG10	R/W	0x4A0	Configuration register for DMA channel 10.		Table 189
CTLSTAT10	RO	0x4A4	Control and status register for DMA channel 10.		Table 191
XFERCFG10	R/W	0x4A8	Transfer configuration register for DMA channel 10.		Table 192
Channel11 registers					
CFG11	R/W	0x4B0	Configuration register for DMA channel 11.		Table 189
CTLSTAT11	RO	0x4B4	Control and status register for DMA channel 11.		Table 191
XFERCFG11	R/W	0x4B8	Transfer configuration register for DMA channel 11.		Table 192
Channel12 registers					
CFG12	R/W	0x4C0	Configuration register for DMA channel 12.		Table 189
CTLSTAT12	RO	0x4C4	Control and status register for DMA channel 12.		Table 191
XFERCFG12	R/W	0x4C8	Transfer configuration register for DMA channel 12.		Table 192
Channel13 registers					
CFG13	R/W	0x4D0	Configuration register for DMA channel 13.		Table 189
CTLSTAT13	RO	0x4D4	Control and status register for DMA channel 13.		Table 191
XFERCFG13	R/W	0x4D8	Transfer configuration register for DMA channel 13.		Table 192
Channel14 registers					
CFG14	R/W	0x4E0	Configuration register for DMA channel 14.		Table 189
CTLSTAT14	RO	0x4E4	Control and status register for DMA channel 14.		Table 191
XFERCFG14	R/W	0x4E8	Transfer configuration register for DMA channel 14.		Table 192
Channel15 registers					
CFG15	R/W	0x4F0	Configuration register for DMA channel 15.		Table 189
CTLSTAT15	RO	0x4F4	Control and status register for DMA channel 15.		Table 191

Table 172. Register overview: DMA controller (base address 0x1C00 4000)

Name	Access	Address offset	Description	Reset Value	Reference
XFERCFG15	R/W	0x4F8	Transfer configuration register for DMA channel 15.		Table 192
Channel16 registers					
CFG16	R/W	0x500	Configuration register for DMA channel 16.		Table 189
CTLSTAT16	RO	0x504	Control and status register for DMA channel 16.		Table 191
XFERCFG16	R/W	0x508	Transfer configuration register for DMA channel 16.		Table 192
Channel17 registers					
CFG17	R/W	0x510	Configuration register for DMA channel 17.		Table 189
CTLSTAT17	RO	0x514	Control and status register for DMA channel 17.		Table 191
XFERCFG17	R/W	0x518	Transfer configuration register for DMA channel 17.		Table 192

13.6.1 Control register

The CTRL register contains global the control bit for a enabling the DMA controller.

Table 173. Control register (CTRL, address 0x1C00 4000) bit description

Bit	Symbol	Value	Description	Reset value
0	ENABLE		DMA controller master enable.	0
		0	Disabled. The DMA controller is disabled. This clears any triggers that were asserted at the point when disabled, but does not prevent re-triggering when the DMA controller is re-enabled.	
		1	Enabled. The DMA controller is enabled.	
31:1	-		Reserved. Read value is undefined, only zero should be written.	NA

13.6.2 Interrupt Status register

The Read-Only INTSTAT register provides an overview of DMA status. This allows quick determination of whether any enabled interrupts are pending. Details of which channels are involved are found in the interrupt type specific registers.

Table 174. Interrupt Status register (INTSTAT, address 0x1C00 4004) bit description

Bit	Symbol	Value	Description	Reset value
0	-		Reserved. Read value is undefined, only zero should be written.	NA
1	ACTIVEINT		Summarizes whether any enabled interrupts are pending.	0
		0	Not pending. No enabled interrupts are pending.	
		1	Pending. At least one enabled interrupt is pending.	
2	ACTIVEERRINT		Summarizes whether any error interrupts are pending.	0
		0	Not pending. No error interrupts are pending.	
		1	Pending. At least one error interrupt is pending.	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

13.6.3 SRAM Base address register

The SRAMBASE register must be configured with an address (preferably in on-chip SRAM) where DMA descriptors will be stored. Software must set up the descriptors for those DMA channels that will be used in the application.

Table 175. SRAM Base address register (SRAMBASE, address 0x1C00 4008) bit description

Bit	Symbol	Description	Reset value
8:0	-	Reserved. Read value is undefined, only zero should be written.	NA
31:9	OFFSET	Address bits 31:9 of the beginning of the DMA descriptor table. For 18 channels, the table must begin on a 512 byte boundary.	0

Each DMA channel has an entry for the channel descriptor in the SRAM table. The values for each channel start at the address offsets found in Table 176. Only the descriptors for channels defined at extraction are used. The contents of each channel descriptor are described in Table 168.

Table 176. Channel descriptor map

Descriptor	Table offset
Channel descriptor for DMA channel 0	0x000
Channel descriptor for DMA channel 1	0x010
Channel descriptor for DMA channel 2	0x020
Channel descriptor for DMA channel 3	0x030
Channel descriptor for DMA channel 4	0x040
Channel descriptor for DMA channel 5	0x050
Channel descriptor for DMA channel 6	0x060
Channel descriptor for DMA channel 7	0x070
Channel descriptor for DMA channel 8	0x080
Channel descriptor for DMA channel 9	0x090
Channel descriptor for DMA channel 10	0x0A0
Channel descriptor for DMA channel 11	0x0B0
Channel descriptor for DMA channel 12	0x0C0
Channel descriptor for DMA channel 13	0x0D0
Channel descriptor for DMA channel 14	0x0E0
Channel descriptor for DMA channel 15	0x0F0
Channel descriptor for DMA channel 16	0x100
Channel descriptor for DMA channel 17	0x110

13.6.4 Enable read and Set registers

The ENABLESET0 register determines whether each DMA channel is enabled or disabled. Disabling a DMA channel does not reset the channel in any way. A channel can be paused and restarted by clearing, then setting the Enable bit for that channel.

Reading ENABLESET0 provides the current state of all of the DMA channels represented by that register. Writing a 1 to a bit position in ENABLESET0 that corresponds to an implemented DMA channel sets the bit, enabling the related DMA channel. Writing a 0 to any bit has no effect. Enables are cleared by writing to ENABLECLR0.

Table 177. Enable read and Set register 0 (ENABLESET0, address 0x1C00 4020) bit description

Bit	Symbol	Description	Reset value
17:0	ENA	Enable for DMA channels 17:0. Bit n enables or disables DMA channel n. 0 = disabled. 1 = enabled.	0
31:18	-	Reserved.	-

13.6.5 Enable Clear register

The ENABLECLR0 register is used to clear the channel enable bits in ENABLESET0. This register is write-only.

Table 178. Enable Clear register 0 (ENABLECLR0, address 0x1C00 4028) bit description

Bit	Symbol	Description	Reset value
17:0	CLR	Writing ones to this register clears the corresponding bits in ENABLESET0. Bit n clears the channel enable bit n.	NA
31:18	-	Reserved.	-

13.6.6 Active status register

The ACTIVE0 register indicates which DMA channels are active at the point when the read occurs. The register is read-only.

A DMA channel is considered active when a DMA operation has been started but not yet fully completed. The Active status will persist from a DMA operation being started, until the pipeline is empty after end of the last descriptor (when there is no reload). An active channel may be aborted by software by setting the appropriate bit in one of the Abort register (see Section 13.6.15).

Table 179. Active status register 0 (ACTIVE0, address 0x1C00 4030) bit description

Bit	Symbol	Description	Reset value
17:0	ACT	Active flag for DMA channel n. Bit n corresponds to DMA channel n. 0 = not active. 1 = active.	0
31:18	-	Reserved.	-

13.6.7 Busy status register

The BUSY0 register indicates which DMA channels is busy at the point when the read occurs. This registers is read-only.

A DMA channel is considered busy when there is any operation related to that channel in the DMA controller's internal pipeline. This information can be used after a DMA channel is disabled by software (but still active), allowing confirmation that there are no remaining operations in progress for that channel.

Table 180. Busy status register 0 (BUSY0, address 0x1C00 4038) bit description

Bit	Symbol	Description	Reset value
17:0	BSY	Busy flag for DMA channel n. Bit n corresponds to DMA channel n. 0 = not busy. 1 = busy.	0
31:18	-	Reserved.	-

13.6.8 Error Interrupt register

The ERRINT0 register contains flags for each DMA channel's Error Interrupt. Any pending interrupt flag in the register will be reflected on the DMA interrupt output.

Reading the registers provides the current state of all DMA channel error interrupts. Writing a 1 to a bit position in ERRINT0 that corresponds to an implemented DMA channel clears the bit, removing the interrupt for the related DMA channel. Writing a 0 to any bit has no effect.

Table 181. Error Interrupt register 0 (ERRINT0, address 0x1C00 4040) bit description

Bit	Symbol	Description	Reset value
17:0	ERR	Error Interrupt flag for DMA channel n. Bit n corresponds to DMA channel n. 0 = error interrupt is not active. 1 = error interrupt is active.	0
31:18	-	Reserved.	-

13.6.9 Interrupt Enable read and Set register

The INTENSET0 register controls whether the individual Interrupts for DMA channels contribute to the DMA interrupt output.

Reading the registers provides the current state of all DMA channel interrupt enables. Writing a 1 to a bit position in INTENSET0 that corresponds to an implemented DMA channel sets the bit, enabling the interrupt for the related DMA channel. Writing a 0 to any bit has no effect. Interrupt enables are cleared by writing to INTENCLR0.

Table 182. Interrupt Enable read and Set register 0 (INTENSET0, address 0x1C00 4048) bit description

Bit	Symbol	Description	Reset value
17: 0	INTEN	Interrupt Enable read and set for DMA channel n. Bit n corresponds to DMA channel n. 0 = interrupt for DMA channel is disabled. 1 = interrupt for DMA channel is enabled.	0
31:18	-	Reserved.	-

13.6.10 Interrupt Enable Clear register

The INTENCLR0 register is used to clear interrupt enable bits in INTENSET0. The register is write-only.

Table 183. Interrupt Enable Clear register 0 (INTENCLR0, address 0x1C00 4050) bit description

Bit	Symbol	Description	Reset value
17:0	CLR	Writing ones to this register clears corresponding bits in the INTENSET0. Bit n corresponds to DMA channel n.	NA
31:18	-	Reserved.	-

13.6.11 Interrupt A register

The IntA0 register contains the interrupt A status for each DMA channel. The status will be set when the SETINTA bit is 1 in the transfer configuration for a channel, when the descriptor becomes exhausted. Writing a 1 to a bit in these registers clears the related INTA flag. Writing 0 has no effect. Any interrupt pending status in the registers will be reflected on the DMA interrupt output if it is enabled in the related INTENSET register.

Table 184. Interrupt A register 0 (INTA0, address 0x1C00 4058) bit description

Bit	Symbol	Description	Reset value
17:0	IA	Interrupt A status for DMA channel n. Bit n corresponds to DMA channel n. 0 = the DMA channel interrupt A is not active. 1 = the DMA channel interrupt A is active.	0
31:18	-	Reserved.	-

13.6.12 Interrupt B register

The INTB0 register contains the interrupt B status for each DMA channel. The status will be set when the SETINTB bit is 1 in the transfer configuration for a channel, when the descriptor becomes exhausted. Writing a 1 to a bit in the register clears the related INTB flag. Writing 0 has no effect. Any interrupt pending status in these registers will be reflected on the DMA interrupt output if it is enabled in the INTENSET register.

Table 185. Interrupt B register 0 (INTB0, address 0x1C00 4060) bit description

Bit	Symbol	Description	Reset value
17:0	IB	Interrupt B status for DMA channel n. Bit n corresponds to DMA channel n. 0 = the DMA channel interrupt B is not active. 1 = the DMA channel interrupt B is active.	0
31:18	-	Reserved.	-

13.6.13 Set Valid register

The SETVALID0 register allows setting the Valid bit in the CTRLSTAT register for one or more DMA channels. See Section 13.6.17 for a description of the VALID bit.

The CFGVALID and SV (set valid) bits allow more direct DMA block timing control by software. Each Channel Descriptor, in a sequence of descriptors, can be validated by either the setting of the CFGVALID bit or by setting the channel's SETVALID flag. Normally, the CFGVALID bit is set. This tells the DMA that the Channel Descriptor is active and can be executed. The DMA will continue sequencing through descriptor blocks whose CFGVALID bit are set without further software intervention. Leaving a CFGVALID bit set to 0 allows the DMA sequence to pause at the Descriptor until software triggers the continuation. If, during DMA transmission, a Channel Descriptor is found with CFGVALID set to 0, the DMA checks for a previously buffered SETVALID0 setting for the channel. If found, the DMA will set the descriptor valid, clear the SV setting, and resume processing the descriptor. Otherwise, the DMA pauses until the channels SETVALID0 bit is set.

Table 186. Set Valid 0 register (SETVALID0, address 0x1C00 4068) bit description

Bit	Symbol	Description	Reset value
17:0	SV	SETVALID control for DMA channel n. Bit n corresponds to DMA channel n. 0 = no effect. 1 = sets the VALIDPENDING control bit for DMA channel n.	NA
31:18	-	Reserved.	-

13.6.14 Set Trigger register

The SETTRIG0 register allows setting the TRIG bit in the CTRLSTAT register for one or more DMA channel. See Section 13.6.17 for a description of the TRIG bit, and Section 13.5.1 for a general description of triggering.

Table 187. Set Trigger 0 register (SETTRIG0, address 0x1C00 4070) bit description

Bit	Symbol	Description	Reset value
17:0	TRIG	Set Trigger control bit for DMA channel 0. Bit n corresponds to DMA channel n. 0 = no effect. 1 = sets the TRIG bit for DMA channel n.	NA
31:18	-	Reserved.	-

13.6.15 Abort registers

The Abort0 register allows aborting operation of a DMA channel if needed. To abort a selected channel, the channel should first be disabled by clearing the corresponding Enable bit by writing a 1 to the proper bit ENABLECLR. Then wait until the channel is no longer busy by checking the corresponding bit in BUSY. Finally, write a 1 to the proper bit of ABORT. This prevents the channel from restarting an incomplete operation when it is enabled again.

Table 188. Abort 0 register (ABORT0, address 0x1C00 4078) bit description

Bit	Symbol	Description	Reset value
17:0	ABORTCTRL	Abort control for DMA channel 0. Bit n corresponds to DMA channel n. 0 = no effect. 1 = aborts DMA operations on channel n.	NA
31:18	-	Reserved.	-

13.6.16 Channel configuration registers

The CFGn register contains various configuration options for DMA channel n.

See Table 190 for a summary of trigger options.

Table 189. Configuration registers for channel 0 to 17 (CFG[0:17], addresses 0x1C00 4400 (CFG0) to address 0x1C00 4510 (CFG17)) bit description

Bit	Symbol	Value	Description	Reset value
0	PERIPHREQEN		Peripheral request Enable. If a DMA channel is used to perform a memory-to-memory move, any peripheral DMA request associated with that channel can be disabled to prevent any interaction between the peripheral and the DMA controller.	0
		0	Disabled. Peripheral DMA requests are disabled.	
		1	Enabled. Peripheral DMA requests are enabled.	
1	HWTRIGEN		Hardware Triggering Enable for this channel.	0
		0	Disabled. Hardware triggering is not used.	
		1	Enabled. Use hardware triggering.	
3:2	-		Reserved. Read value is undefined, only zero should be written.	NA
4	TRIGPOL		Trigger Polarity. Selects the polarity of a hardware trigger for this channel.	0
		0	Active low - falling edge. Hardware trigger is active low or falling edge triggered, based on TRIGTYPE.	
		1	Active high - rising edge. Hardware trigger is active high or rising edge triggered, based on TRIGTYPE.	
5	TRIGTYPE		Trigger Type. Selects hardware trigger as edge triggered or level triggered.	0
		0	Edge. Hardware trigger is edge triggered. Transfers will be initiated and completed, as specified for a single trigger.	
		1	Level. Hardware trigger is level triggered. Note that when level triggering without burst (BURSTPOWER = 0) is selected, only hardware triggers should be used on that channel. Transfers continue as long as the trigger level is asserted. Once the trigger is de-asserted, the transfer will be paused until the trigger is, again, asserted. However, the transfer will not be paused until any remaining transfers within the current BURSTPOWER length are completed.	
6	TRIGBURST		Trigger Burst. Selects whether hardware triggers cause a single or burst transfer.	0
		0	Single transfer. Hardware trigger causes a single transfer.	
		1	Burst transfer. When the trigger for this channel is set to edge triggered, a hardware trigger causes a burst transfer, as defined by BURSTPOWER. When the trigger for this channel is set to level triggered, a hardware trigger causes transfers to continue as long as the trigger is asserted, unless the transfer is complete.	
7	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 189. Configuration registers for channel 0 to 17 (CFG[0:17], addresses 0x1C00 4400 (CFG0) to address 0x1C00 4510 (CFG17)) bit description

Bit	Symbol	Value	Description	Reset value
11:8	BURSTPOWER		Burst Power is used in two ways. It always selects the address wrap size when SRCBURSTWRAP and/or DSTBURSTWRAP modes are selected (see descriptions elsewhere in this register). When the TRIGBURST field elsewhere in this register = 1, Burst Power selects how many transfers are performed for each DMA trigger. This can be used, for example, with peripherals that contain a FIFO that can initiate a DMA operation when the FIFO reaches a certain level. 0000: Burst size = 1 (2^0). 0001: Burst size = 2 (2^1). 0010: Burst size = 4 (2^2). ... 1010: Burst size = 1024 (2^{10}). This corresponds to the maximum supported transfer count. others: not supported. The total transfer length as defined in the XFERCOUNT bits in the XFERCFG register must be an even multiple of the burst size.	0
13:12	-		Reserved. Read value is undefined, only zero should be written.	NA
14	SRCBURSTWRAP		Source Burst Wrap. When enabled, the source data address for the DMA is “wrapped”, meaning that the source address range for each burst will be the same. As an example, this could be used to read several sequential registers from a peripheral for each DMA burst, reading the same registers again for each burst.	0
		0	Disabled. Source burst wrapping is not enabled for this DMA channel.	
		1	Enabled. Source burst wrapping is enabled for this DMA channel.	
15	DSTBURSTWRAP		Destination Burst Wrap. When enabled, the destination data address for the DMA is “wrapped”, meaning that the destination address range for each burst will be the same. As an example, this could be used to write several sequential registers to a peripheral for each DMA burst, writing the same registers again for each burst.	0
		0	Disabled. Destination burst wrapping is not enabled for this DMA channel.	
		1	Enabled. Destination burst wrapping is enabled for this DMA channel.	
18:16	CHPRIORITY		Priority of this channel when multiple DMA requests are pending. Eight priority levels are supported. 0x0 = highest priority. 0x7 = lowest priority.	0
31:19	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 190. Trigger setting summary

TrigBurst	TrigType	TrigPol	Description
0	0	0	Hardware DMA trigger is falling edge sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP.
0	0	1	Hardware DMA trigger is rising edge sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP.
0	1	0	Hardware DMA trigger is low level sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP.
0	1	1	Hardware DMA trigger is high level sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP.
1	0	0	Hardware DMA trigger is falling edge sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP, and also determines how much data is transferred for each trigger.
1	0	1	Hardware DMA trigger is rising edge sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP, and also determines how much data is transferred for each trigger.
1	1	0	Hardware DMA trigger is low level sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP, and also determines how much data is transferred for each trigger.
1	1	1	Hardware DMA trigger is high level sensitive. The BURSTPOWER field controls address wrapping if enabled via SRCBURSTWRAP and/or DSTBURSTWRAP, and also determines how much data is transferred for each trigger.

13.6.17 Channel control and status registers

The CTLSTATn register provides status flags specific to DMA channel n.

Table 191. Control and Status registers for channel 0 to 17 (CTLSTAT[0:17], 0x1C00 4404 (CTLSTAT0) to address 0x1C00 4514 (CTLSTAT17)) bit description

Bit	Symbol	Value	Description	Reset value
0	VALIDPENDING		Valid pending flag for this channel. This bit is set when a 1 is written to the corresponding bit in the related SETVALID register when CFGVALID = 1 for the same channel.	0
		0	No effect on DMA operation.	
		1	Valid pending.	
1	-		Reserved. Read value is undefined, only zero should be written.	NA
2	TRIG		Trigger flag. Indicates that the trigger for this channel is currently set. This bit is cleared at the end of an entire transfer or upon reload when CLRTRIG = 1.	0
		0	Not triggered. The trigger for this DMA channel is not set. DMA operations will not be carried out.	
		1	Triggered. The trigger for this DMA channel is set. DMA operations will be carried out.	
31:3	-		Reserved. Read value is undefined, only zero should be written.	NA

13.6.18 Channel transfer configuration registers

The XFERCFGn register contains transfer related configuration information for DMA channel n. Using the Reload bit, this register can optionally be automatically reloaded when the current settings are exhausted (the full transfer count has been completed), allowing linked transfers with more than one descriptor to be performed.

See "Trigger operation" for details on trigger operation.

Table 192. Transfer Configuration registers for channel 0 to 17 (XFERCFG[0:17], addresses 0x1C00 4408 (XFERCFG0) to 0x1C00 4518 (XFERCFG17)) bit description

Bit	Symbol	Value	Description	Reset Value
0	CFGVALID		Configuration Valid flag. This bit indicates whether the current channel descriptor is valid and can potentially be acted upon, if all other activation criteria are fulfilled.	0
		0	Not valid. The channel descriptor is not considered valid until validated by an associated SETVALID0 setting.	
		1	Valid. The current channel descriptor is considered valid.	
1	RELOAD		Indicates whether the channel's control structure will be reloaded when the current descriptor is exhausted. Reloading allows ping-pong and linked transfers.	0
		0	Disabled. Do not reload the channels' control structure when the current descriptor is exhausted.	
		1	Enabled. Reload the channels' control structure when the current descriptor is exhausted.	
2	SWTRIG		Software Trigger.	0
		0	When written by software, the trigger for this channel is not set. A new trigger, as defined by the HWTRIGEN, TRIGPOL, and TRIGTYPE will be needed to start the channel.	
		1	When written by software, the trigger for this channel is set immediately. This feature should not be used with level triggering when TRIGBURST = 0.	
3	CLRTRIG		Clear Trigger.	0
		0	Not cleared. The trigger is not cleared when this descriptor is exhausted. If there is a reload, the next descriptor will be started.	
		1	Cleared. The trigger is cleared when this descriptor is exhausted.	
4	SETINTA		Set Interrupt flag A for this channel. There is no hardware distinction between interrupt A and B. They can be used by software to assist with more complex descriptor usage. By convention, interrupt A may be used when only one interrupt flag is needed.	0
		0	No effect.	
		1	Set. The INTA flag for this channel will be set when the current descriptor is exhausted.	
5	SETINTB		Set Interrupt flag B for this channel. There is no hardware distinction between interrupt A and B. They can be used by software to assist with more complex descriptor usage. By convention, interrupt A may be used when only one interrupt flag is needed.	0
		0	No effect.	
		1	Set. The INTB flag for this channel will be set when the current descriptor is exhausted.	
7:6	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 192. Transfer Configuration registers for channel 0 to 17 (XFERCFG[0:17], addresses 0x1C00 4408 (XFERCFG0) to 0x1C00 4518 (XFERCFG17)) bit description

Bit	Symbol	Value	Description	Reset Value
9:8	WIDTH		Transfer width used for this DMA channel.	0
		0x0	8-bit transfers are performed (8-bit source reads and destination writes).	
		0x1	16-bit transfers are performed (16-bit source reads and destination writes).	
		0x2	32-bit transfers are performed (32-bit source reads and destination writes).	
		0x3	Reserved setting, do not use.	
11:10	-		Reserved. Read value is undefined, only zero should be written.	NA
13:12	SRCINC		Determines whether the source address is incremented for each DMA transfer.	0
		0x0	No increment. The source address is not incremented for each transfer. This is the usual case when the source is a peripheral device.	
		0x1	1 x width. The source address is incremented by the amount specified by Width for each transfer. This is the usual case when the source is memory.	
		0x2	2 x width. The source address is incremented by 2 times the amount specified by Width for each transfer.	
		0x3	4 x width. The source address is incremented by 4 times the amount specified by Width for each transfer.	
15:14	DSTINC		Determines whether the destination address is incremented for each DMA transfer.	0
		0x0	No increment. The destination address is not incremented for each transfer. This is the usual case when the destination is a peripheral device.	
		0x1	1 x width. The destination address is incremented by the amount specified by Width for each transfer. This is the usual case when the destination is memory.	
		0x2	2 x width. The destination address is incremented by 2 times the amount specified by Width for each transfer.	
		0x3	4 x width. The destination address is incremented by 4 times the amount specified by Width for each transfer.	
25:16	XFERCOUNT		Total number of transfers to be performed, minus 1 encoded. The number of bytes transferred is: (XFERCOUNT + 1) x data width (as defined by the WIDTH field). **Remark:** The DMA controller uses this bit field during transfer to count down. Hence, it cannot be used by software to read back the size of the transfer, for instance, in an interrupt handler. 0x0 = a total of 1 transfer will be performed. 0x1 = a total of 2 transfers will be performed. ... 0x3FF = a total of 1,024 transfers will be performed.	0
31:26	-		Reserved. Read value is undefined, only zero should be written.	NA

13.7 Functional description

13.7.1 Trigger operation

A trigger of some kind is always needed to start a transfer on a DMA channel. This can be a hardware or software trigger, and can be used in several ways.

If a channel is configured with the SWTRIG bit equal to 0, the channel can be later triggered either by hardware or software. Software triggering is accomplished by writing a 1 to the appropriate bit in the SETTRIG register. Hardware triggering requires setup of the HWTRIGEN, TRIGPOL, TRIGTYPE, and TRIGBURST fields in the CFG register for the related channel. When a channel is initially set up, the SWTRIG bit in the XFERCFG register can be set, causing the transfer to begin immediately.

Once triggered, transfer on a channel will be paced by DMA requests if the PERIPHREQEN bit in the related CFG register is set. Otherwise, the transfer will proceed at full speed.

The TRIG bit in the CTLSTAT register can be cleared at the end of a transfer, determined by the value CLRTRIG (bit 0) in the XFERCFG register. When a 1 is found in CLRTRIG, the trigger is cleared when the descriptor is exhausted.

14.1 How to read this chapter

The SCTIPU is available on all parts.

14.2 Features

The SCTIPU pre-processes inputs to the State-Configurable Timers (SCT).

- Four outputs created from a selection of input transitions. Each output can be used as abort input to the SCTs or for any other application which requires a collection of multiple SCT inputs to trigger an identical SCT response.
- Four registers to indicate which specific input sources caused the abort input to the SCTs.
- Four additional outputs which can be sampled at certain times and latched at others before being routed to SCT inputs.
- Nine abort inputs. Any combination of the abort inputs can trigger the dedicated abort input of each SCT.

14.3 Basic configuration

Configure the SCTIPU as follows:

- Use the SYSAHBCLKCTRL1 register (Table 51) to enable the clock to the SCTIPU registers interface.
- Clear the SCTIPU peripheral reset using the PRESETCTRL1 register (Table 36).
- Select the SCTIPU sample inputs from external pins and the comparator outputs. See Table 193 and Figure 24.
- Select the abort inputs from the SCT_ABORT pins via the switch matrix, the SCT0 output SCT0_OUT9, the ADC threshold compare interrupts, or the comparator outputs. See Table 197.
- SCTIPU outputs are connected to the SCT inputs and selected through the SCT input mux registers. See Table 193.

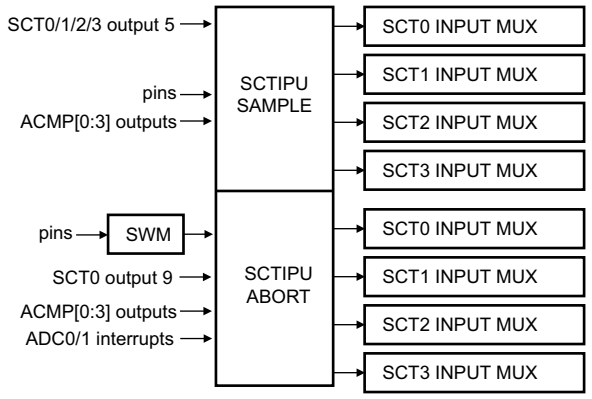


Fig 23. SCTIPU connections

14.3.1 SCTIPU to SCT connections

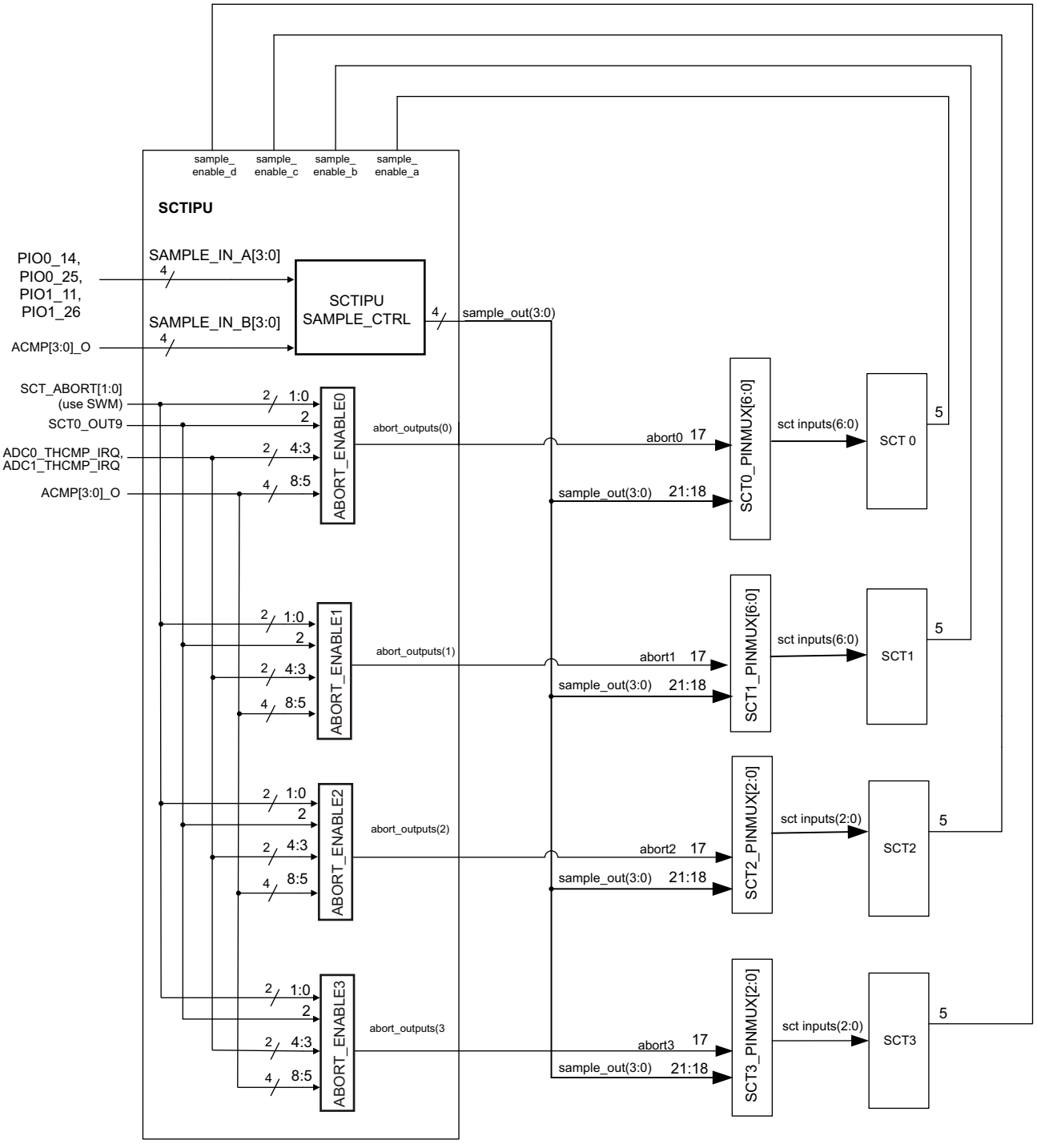


Fig 24. SCTIPU to SCT connections

14.4 Pin description

Table 193. SCTIPU sample sub-module pin description

Function	I/O	Type	Connect to	Use register	Reference	Description
SAMPLE_IN_A0	I	external to pin	PIO0_14	SAMPLE_CTRL	Table 196	Input source for channel 0 to be routed to output channel 0.
SAMPLE_IN_B0	I	internal	ACMP0_O out	SAMPLE_CTRL	Table 196	Input source for channel 0 to be routed to output channel 0.
SAMPLE_IN_A1	I	external to pin	PIO0_25	SAMPLE_CTRL	Table 196	Input source for channel 1 to be routed to output channel 1.
SAMPLE_IN_B1	I	internal	ACMP1_O out	SAMPLE_CTRL	Table 196	Input source for channel 1 to be routed to output channel 1.
SAMPLE_IN_A2	I	external to pin	PIO1_11	SAMPLE_CTRL	Table 196	Input source for channel 2 to be routed to output channel 2.
SAMPLE_IN_B2	I	internal	ACMP2_O out	SAMPLE_CTRL	Table 196	Input source for channel 2 to be routed to output channel 2.
SAMPLE_IN_A3	I	external to pin	PIO1_26	SAMPLE_CTRL	Table 196	Input source for channel 3 to be routed to output channel 3.
SAMPLE_IN_B3	I	internal	ACMP3_O out	SAMPLE_CTRL	Table 196	Input source for channel 3 to be routed to output channel 3.
SAMPLE_ENABLE_A	I	internal	SCT0_OUT5	-	-	Latch/sample enable control input.
SAMPLE_ENABLE_B	I	internal	SCT1_OUT5	-	-	Latch/sample enable control input.
SAMPLE_ENABLE_C	I	internal	SCT2_OUT5	-	-	Latch/sample enable control input.
SAMPLE_ENABLE_D	I	internal	SCT3_OUT5	-	-	Latch/sample enable control input.
SAMPLE_OUT0	O	internal	SCT0_INMUX[0:6], SCT1_INMUX[0:6], SCT2_INMUX[0:2], SCT3_INMUX[0:2]	SCTn_INMUXm	Table 127, Table 128, Table 129, Table 130	Latched/sampled output channel 0.

Table 193. SCTIPU sample sub-module pin description

Function	I/O	Type	Connect to	Use register	Reference	Description
SAMPLE_OUT2	O	internal	SCT0_INMUX[0:6], SCT1_INMUX[0:6], SCT2_INMUX[0:2], SCT3_INMUX[0:2]	SCTn_INMUXm	Table 127, Table 128, Table 129, Table 130	Latched/sampled output channel 1.
SAMPLE_OUT3	O	internal	SCT0_INMUX[0:6], SCT1_INMUX[0:6], SCT2_INMUX[0:2], SCT3_INMUX[0:2]	SCTn_INMUXm	Table 127, Table 128, Table 129, Table 130	Latched/sampled output channel 2
SAMPLE_OUT4	O	internal	SCT0_INMUX[0:6], SCT1_INMUX[0:6], SCT2_INMUX[0:2], SCT3_INMUX[0:2]	SCTn_INMUXm	Table 127, Table 128, Table 129, Table 130	Latched/sampled output channel 3.

Table 194. SCTIPU abort sub-module pin description

Function	I/O	Type	Connected to	Use register	Reference	Description
ABORT_IN[0:8]	I	internal or external to pins	Any combination of comparator outputs, ADC threshold compare outputs, SCT outputs, and abort pin inputs through switch matrix PINASSIGN10.	Table 197, Table 198	-	Inputs to abort channels 0 to 3.
ABORT_OUT0	O	internal	SCT0_INMUX[0:6]	SCT0_INMUXm	Table 127	SCTIPU abort output 0.
ABORT_OUT1	O	internal	SCT1_INMUX[0:6],	SCT1_INMUXm	Table 128	SCTIPU abort output 1.
ABORT_OUT2	O	internal	SCT2_INMUX[0:2]	SCT2_INMUXm	Table 129	SCTIPU abort output 2.
ABORT_OUT3	O	internal	SCT3_INMUX[0:2]	SCT3_INMUXm	Table 130	SCTIPU abort output 3.

14.5 General description

The SCTIPU is a companion block to the State Configurable Timers (SCTs), useful predominantly when the SCTs are employed in control systems such as motor or power control. The SCTIPU performs certain pre-processing on a selection of input signals before they are passed-on as inputs to the various SCTs. The SCTIPU itself is comprised of a SAMPLE sub-module and four ABORT sub-modules.

14.5.1 Abort sub-modules

The ABORT sub-module is essentially an AND-OR structure. Nine input sources are shared among the four ABORT sub-modules. Each of these modules has a 9-bit enable register which can be programmed to enable one or more of the nine inputs to contribute to the ORed output from that particular sub-module. The four outputs are then routed to inputs on the four SCTs.

This module is useful whenever a collection of multiple input sources trigger exactly the same response in an SCT. Combining these input sources into a single SCT input eliminates the need to waste multiple inputs and tie-up multiple events in order to generate the same SCT reaction. A typical use of these inputs is to trigger SCT events which implement an emergency shut-down procedures (hence the name ABORT).

14.5.2 Sample sub-module

This sub-module provides a set of latches which allow four SCT inputs to be sampled at certain times and latched at others. Each of the four outputs from this module have two alternative input sources and four alternative latch/sample-enable signals. It is also possible to override latching on any of the four outputs and force its latch into transparent mode.

The primary purpose of this module is to block signals from being passed-on to the SCT inputs during periods when they are known to be invalid (eg. during switching). The four latch/sample-enable inputs are driven by SCT outputs.

14.6 Register description

Table 195. Register overview: SCTIPU (base address 0x400B 8000)

Name	Access	Address offset	Description	Reset value	Reference
SAMPLE_CTRL	R/W	0x000	SCTIPU sample control register. Contains the input mux selects, latch/sample-enable mux selects, and sample override bits for the SAMPLE module.	0	Table 196
ABORT_ENABLE0	R/W	0x020	SCTIPU abort enable 0 register: Selects which input source contributes to ORed Abort Output 0.	0	
ABORT_SOURCE0	R/W	0x024	SCTIPU abort source 0 register: Status register indicating which input source caused abort output 0.	0	
ABORT_ENABLE1	R/W	0x040	SCTIPU abort enable 1 register: Selects which input source contributes to ORed Abort Output 1.	0	
ABORT_SOURCE1	R/W	0x044	SCTIPU abort source 1 register: Status register indicating which input source caused abort output 1.	0	
ABORT_ENABLE2	R/W	0x060	SCTIPU abort enable 2 register: Selects which input source contributes to ORed Abort Output 2.	0	
ABORT_SOURCE2	R/W	0x064	SCTIPU abort source 2 register: Status register indicating which input source caused abort output 2.	0	
ABORT_ENABLE3	R/W	0x080	SCTIPU abort enable 3 register: Selects which input source contributes to ORed Abort Output 1.	0	
ABORT_SOURCE3	R/W	0x084	SCTIPU abort source 3 register: Status register indicating which input source caused abort output 3.	0	

14.6.1 SCTIPU Sample control register

Table 196: SCTIPU Sample control register (SAMPLE_CTRL, address 0x400B 8000) bit description

Bit	Symbol	Value	Description	Reset value
0	IN0SEL		Select SCTIPU input source for output channel 0.	0
		0	SAMPE_IN_A0. Select input SAMPLE_IN_A0.	
		1	SAMPE_IN_B0. Select input SAMPLE_IN_B0.	
1	IN1SEL		Select SCTIPU input source for output channel 1.	0
		0	SAMPE_IN_A1. Select input SAMPLE_IN_A1.	
		1	SAMPE_IN_B1. Select input SAMPLE_IN_B1.	
2	IN2SEL		Select SCTIPU input source for output channel 2.	0
		0	SAMPE_IN_A2. Select input SAMPLE_IN_A2.	
		1	SAMPE_IN_B2. Select input SAMPLE_IN_B2.	
3	IN3SEL		Select. SCTIPU input source for output channel 3.	0
		0	SAMPE_IN_A3. Select input SAMPLE_IN_A3.	
		1	SAMPE_IN_B3. Select input SAMPLE_IN_B3.	
5:4	SAMPLE_EN0SEL		Select the sample enable input as the latch/sample-enable control for the Sample_Output(0) latch. Depending on the value of the corresponding LATCHn_EN bit, this latch is transparent when the LATCHn_EN bit is 1 or latched when the LATCHn_EN bit is 0.	0
		0x0	Selects Sample_Enable_A as the latch/sample-enable control for the Sample_Output(0) latch.	
		0x1	Selects Sample_Enable_B as the latch/sample-enable control for the Sample_Output(0) latch.	
		0x2	Selects Sample_Enable_C as the latch/sample-enable control for the Sample_Output(0) latch.	
		0x3	Selects Sample_Enable_D as the latch/sample-enable control for the Sample_Output(0) latch.	
7:6	SAMPLE_EN1SEL		Select the sample enable input as the latch/sample-enable control for the Sample_Output(1) latch. Depending on the value of the corresponding LATCHn_EN bit, this latch is transparent when the LATCHn_EN bit is 1 or latched when the LATCHn_EN bit is 0.	0
		0x0	Selects Sample_Enable_A as the latch/sample-enable control for the Sample_Output(1) latch.	
		0x1	Selects Sample_Enable_B as the latch/sample-enable control for the Sample_Output(1) latch.	
		0x2	Selects Sample_Enable_C as the latch/sample-enable control for the Sample_Output(1) latch.	
		0x3	Selects Sample_Enable_D as the latch/sample-enable control for the Sample_Output(1) latch.	

Table 196: SCTIPU Sample control register (SAMPLE_CTRL, address 0x400B 8000) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
9:8	SAMPLE_EN2SEL		Select the sample enable input as the latch/sample-enable control for the Sample_Output(2) latch. Depending on the value of the corresponding LATCHn_EN bit, this latch is transparent when the LATCHn_EN bit is 1 or latched when the LATCHn_EN bit is 0.	0
		0x0	Selects Sample_Enable_A as the latch/sample-enable control for the Sample_Output(2) latch.	
		0x1	Selects Sample_Enable_B as the latch/sample-enable control for the Sample_Output(2) latch.	
		0x2	Selects Sample_Enable_C as the latch/sample-enable control for the Sample_Output(2) latch.	
		0x3	Selects Sample_Enable_D as the latch/sample-enable control for the Sample_Output(2) latch.	
11:10	SAMPLE_EN3SEL		Select the sample enable input as the latch/sample-enable control for the Sample_Output(3) latch. Depending on the value of the corresponding LATCHn_EN bit, this latch is transparent when the LATCHn_EN bit is 1 or latched when the LATCHn_EN bit is 0.	0
		0x0	Selects Sample_Enable_A as the latch/sample-enable control for the Sample_Output(3) latch.	
		0x1	Selects Sample_Enable_B as the latch/sample-enable control for the Sample_Output(3) latch.	
		0x2	Selects Sample_Enable_C as the latch/sample-enable control for the Sample_Output(3) latch.	
		0x3	Selects Sample_Enable_D as the latch/sample-enable control for the Sample_Output(3) latch.	
12	LATCHEN0		Enable latch for output channel 0.	0
		0	Transparent mode. Sample_Output(0) latch is forced into transparent mode. The selected Sample_Input is passed directly through to Sample_Output(0). The sample-enable control line selected for this latch has no effect.	
		1	Latched mode. The Sample_Output(0) latch is operational and will sample or latch based on the state of the selected sample-enable control signal.	
13	LATCHEN1		Enable latch for output channel 1.	0
		0	Transparent mode. Sample_Output(1) latch is forced into transparent mode. The selected Sample_Input is passed directly through to Sample_Output(1). The sample-enable control line selected for this latch has no effect.	
		1	Latched mode. The Sample_Output(1) latch is operational and will sample or latch based on the state of the selected sample-enable control signal.	
14	LATCHEN2		Enable latch for output channel 2.	0
		0	Transparent mode. Sample_Output(2) latch is forced into transparent mode. The selected Sample_Input is passed directly through to Sample_Output(2). The sample-enable control line selected for this latch has no effect.	
		1	Latched mode. The Sample_Output(2) latch is operational and will sample or latch based on the state of the selected sample-enable control signal.	

Table 196: SCTIPU Sample control register (SAMPLE_CTRL, address 0x400B 8000) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
15	LATCHEN3		Enable latch for output channel 3.	0
		0	Transparent mode. Sample_Output(3) latch is forced into transparent mode. The selected Sample_Input is passed directly through to Sample_Output(3). The sample-enable control line selected for this latch has no effect.	
		1	Latched mode. The Sample_Output(3) latch is operational and will sample or latch based on the state of the selected sample-enable control signal.	
31:16			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

14.6.2 SCT Abort enable registers 0 to 3

For each of the four abort sub-modules, this register selects which input sources will be enabled to contribute to the ORed output.

Table 197: SCTIPU Abort enable register (ABORT_ENABLE[0:3], address 0x400B 8020 (ABORT_ENABLE0) to 0x400B 8080 (ABORT_ENABLE3)) bit description

Bit	Symbol	Value	Description	Reset value
0	ENA0		Enable abort source SCT_ABORT0. Select pin from switch matrix.	0
		0	Disabled.	
		1	Enabled.	
1	ENA1		Enable abort source SCT_ABORT1. Select pin from switch matrix.	0
		0	Disabled.	
		1	Enabled.	
2	ENA2		Enable abort source SCT0_OUT9.	0
		0	Disabled.	
		1	Enabled.	
3	ENA3		Enable abort source ADC0_THCMP_IRQ.	0
		0	Disabled.	
		1	Enabled.	
4	ENA4		Enable abort source ADC1_THCMP_IRQ.	0
		0	Disabled.	
		1	Enabled.	
5	ENA5		Enable abort source ACMP0_O output.	0
		0	Disabled.	
		1	Enabled.	
6	ENA6		Enable abort source ACMP1_O output.	0
		0	Disabled.	
		1	Enabled.	
7	ENA7		Enable abort source ACMP2_O output.	0
		0	Disabled.	
		1	Enabled.	

Table 197: SCTIPU Abort enable register (ABORT_ENABLE[0:3], address 0x400B 8020 (ABORT_ENABLE0) to 0x400B 8080 (ABORT_ENABLE3)) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
8	ENA8		Enable abort source ACMP3_O output.	0
		0	Disabled.	
		1	Enabled.	
31:9			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

14.6.3 SCT Abort source registers 0 to 3

For each sub-module, this status register indicates which one (or more) abort inputs occurred to cause the abort output to be activated.

Remark: The accuracy of the Abort Source Status register is not guaranteed under all conditions. If an abort-enabled input is applied for a relatively short period of time, it is possible that it may be captured in this register even if it wasn't sustained long enough to actually be recognized as a valid input by the SCT.

Table 198: SCTIPU Abort source register (ABORT_SOURCE[0:3], address 0x400B 8024 (ABORT_SOURCE0) to 0x400B 8084 (ABORT_SOURCE3)) bit description

Bit	Symbol	Value	Description	Reset value
0	ACT0		Source SCT_ABORT0 activated. This bit is set by hardware when the source is active. Write 0 to clear. This function can be assigned to any pin via the PINASSIGN10 register in the switch matrix.	0
		0	Not activated.	
		1	Activated.	
1	ACT1		Source SCT_ABORT1 activated. This bit is set by hardware when the source is active. Write 0 to clear. This function can be assigned to any pin via the PINASSIGN10 register in the switch matrix.	0
		0	Not activated.	
		1	Activated.	
2	ACT2		Source SCT0_OUT9 activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
3	ACT3		Source ADC0_THCMP_IRQ activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
4	ACT4		Source ADC1_THCMP_IRQ activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	

Table 198: SCTIPU Abort source register (ABORT_SOURCE[0:3], address 0x400B 8024 (ABORT_SOURCE0) to 0x400B 8084 (ABORT_SOURCE3)) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
5	ACT5		Source ACMP0_O output activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
6	ACT6		Source ACMP1_O output activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
7	ACT7		Source ACMP2_O output activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
8	ACT8		Source ACMP3_O output activated. This bit is set by hardware when the source is active. Write 0 to clear.	0
		0	Not activated.	
		1	Activated.	
31:9			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

15.1 How to read this chapter

The large SCTs are available on all parts.

The SCT0 and SCT1 outputs #7 cannot be connected to external pins on the LQFP48 and LQFP64 packages. (They are available for internal connections.)

15.2 Features

The following feature list summarizes the configuration for the two large SCTs. Each large SCT has a companion small SCT (see Section 16.2) with fewer inputs and outputs and a reduced feature set.

- Each SCT supports:
  - 16 match/capture registers
  - 16 events
  - 16 states
  - Match register 0 to 5 support a fractional component for the dither engine
  - 8 inputs and 10 outputs
- Counter/timer features:
  - Configurable as two 16-bit counters or one 32-bit counter.
  - Counters clocked by bus clock or selected input.
  - Up counters or up-down counters.
  - Configurable number of match and capture registers. Up to 16 match and capture registers total.
  - Upon match create the following events: interrupt, stop, limit timer or change direction; toggle outputs.
  - Counter value can be loaded into capture register triggered by match or input/output toggle.
- PWM features:
  - Counters can be used in conjunction with match registers to toggle outputs and create time-proportioned PWM signals.
  - Up to 8 single-edge or dual-edge controlled PWM outputs with independent duty cycles and common PWM cycle length.
- Event creation features:
  - The following conditions define an event: a counter match condition, an input (or output) condition, a combination of a match and/or and input/output condition in a specified state.
  - Selected events can limit, halt, start, or stop a counter.
  - Events control state changes, outputs, interrupts, and DMA requests.

  - Match register 0 can be used as an automatic limit.
  - In bi-directional mode, events can be enabled based on the count direction.
  - Match events can be held until another qualifying event occurs.
- State control features:
  - A state is defined by events that can take place in the state while the counter is running.
  - A state changes into another state as result of an event.
  - Each event can be assigned to one or more states.
  - State variable allows sequencing across multiple counter cycles.
- Dither engine.
- Integrated with an input pre-processing unit (SCTIPU) to combine or delay input events.

Inputs and outputs on the SCT0 and SCT1 are configured as follows:

- 8 inputs
  - 7 inputs. Each input except input 7 can select one of 23 sources from an input multiplexer.
  - One input connected directly to the SCT PLL for a high-speed dedicated clock input.
- 10 outputs (some outputs are connected to multiple locations)
  - Three outputs connected to external pins through the switch matrix as movable functions.
  - Five outputs connected to external pins through the switch matrix as fixed-pin functions.
  - Two outputs connected to the SCTIPU to use as sample and abort inputs.
  - One output connected to the other large SCT
  - Four outputs connected to one small SCT
  - Two outputs connected to each ADC trigger input

15.3 Basic configuration

Configure the SCT as follows:

- Use the SYSAHBCLKCTRL register (Table 51) to enable the clock to the SCT0/1 registers interface and peripheral clock. The system clock is the input clock to the SCT clock processing and is the source of the SCT clock.
- Clear the SCT peripheral reset using the PRESETCTRL register (Table 36).
- The SCT0/1 combined interrupts are connected to slots #16/17 in the NVIC.
- SCT inputs are selected from the SCT0/1 input mux registers (see Table 127 and Table 128).
- Use the switch matrix to connect the SCT outputs to pins (see Table 200).

Remark: For applications that require exact timing of the SCT outputs (for example PWM), assign the outputs only to fixed-pin functions to ensure that the output skew is nearly the same for all outputs.

- The SCT DMA request lines 0 and 1 are connected to the DMA trigger inputs via the DMA_ITRIG_INMUX registers. See Table 132.
- For logically combining SCT input events for an abort function and blocking SCT inputs for a fixed amount of time, see Section 15.3.5 “Use the SCT with the SCTIPU”.

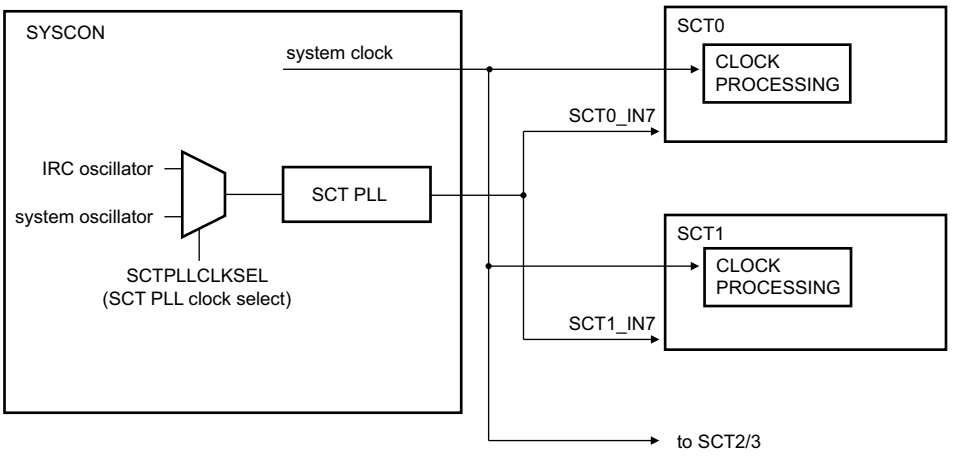


Fig 25. Large SCT clocking

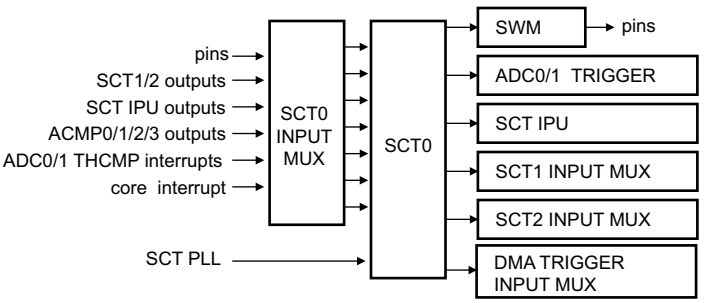


Fig 26. SCT0 connections

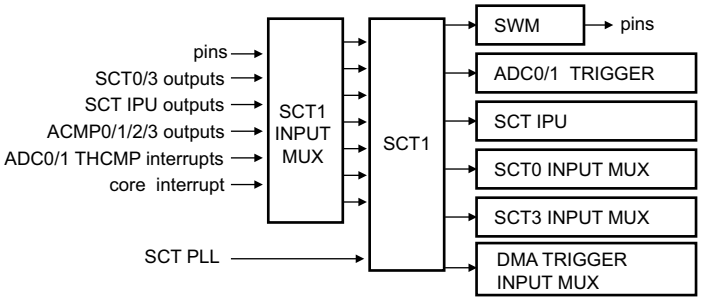


Fig 27. SCT1 connections

15.3.1 SCT inputs and outputs

Each of the SCT inputs has a pre-selector that can select one of 23 possible input signals (SCT input mux). The input signals are numbered 0 to 22 (see Figure 28), and the signal number is programmed in the SCT0_INMUXn or SCT1_INMUXn register for any particular input:

SCT0 inputs are selected through input muxes SCT0_INMUX[0:6]. See Table 127.

SCT1 inputs are selected through input muxes SCT1_INMUX[0:6]. See Table 128.

One SCT input (SCT_IN7) is dedicated to the high-speed clock signal generated by the SCT PLL in the CGU (see Figure 3 and Section 3.6.44).

Each large SCT has 10 outputs. Eight outputs are connected to external pins through the switch matrix. Some outputs are routed to the other SCT blocks, to the ADC trigger inputs, and to the SCTIPU. An SCT output can be routed to multiple places.

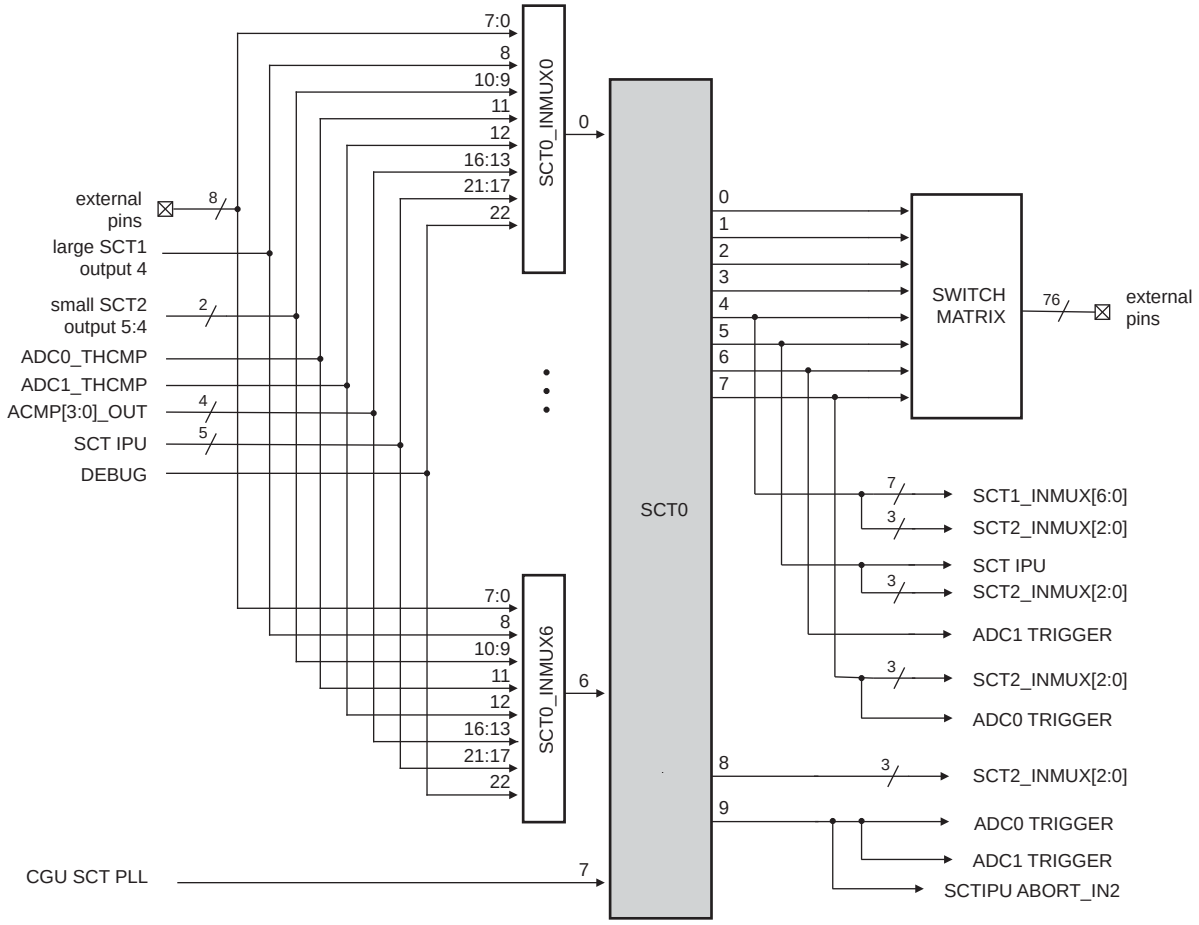


Fig 28. SCT0 inputs and outputs

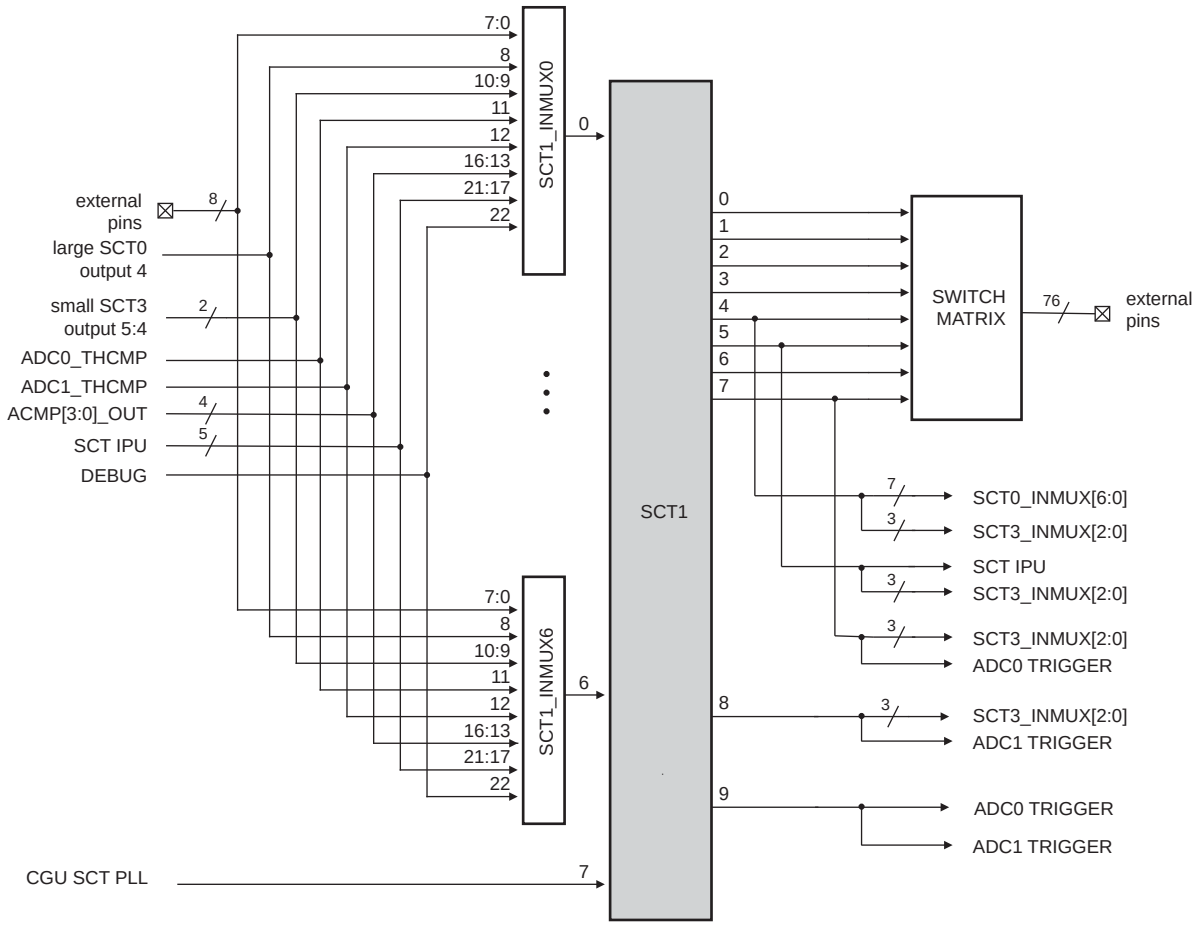


Fig 29. SCT1 inputs and outputs

15.3.2 Connections between large and small SCTs

One large and one small SCT can be chained together through their inputs and outputs. Each large SCT has one associated small SCT. The two SCTs are connected through the three outputs of the large SCT and the input pin muxes of the small SCT. Two outputs of the small SCT are connected back to the input pin muxes of the associated large SCT.

In addition, each large SCT has one output connected to the other large SCT. The small SCTs are not connected to each other.

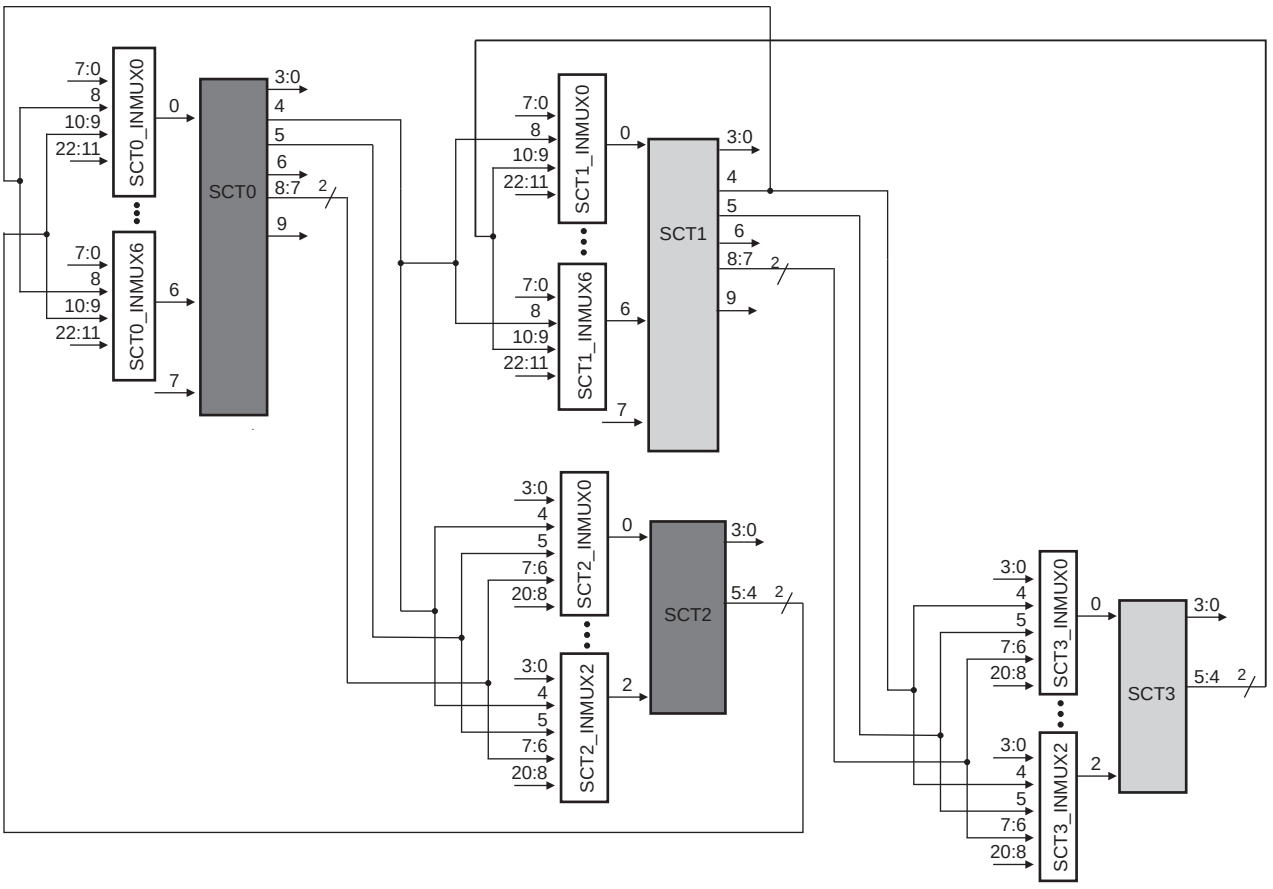


Fig 30. SCT chaining

15.3.3 Connections between the SCTs and the ADC trigger inputs

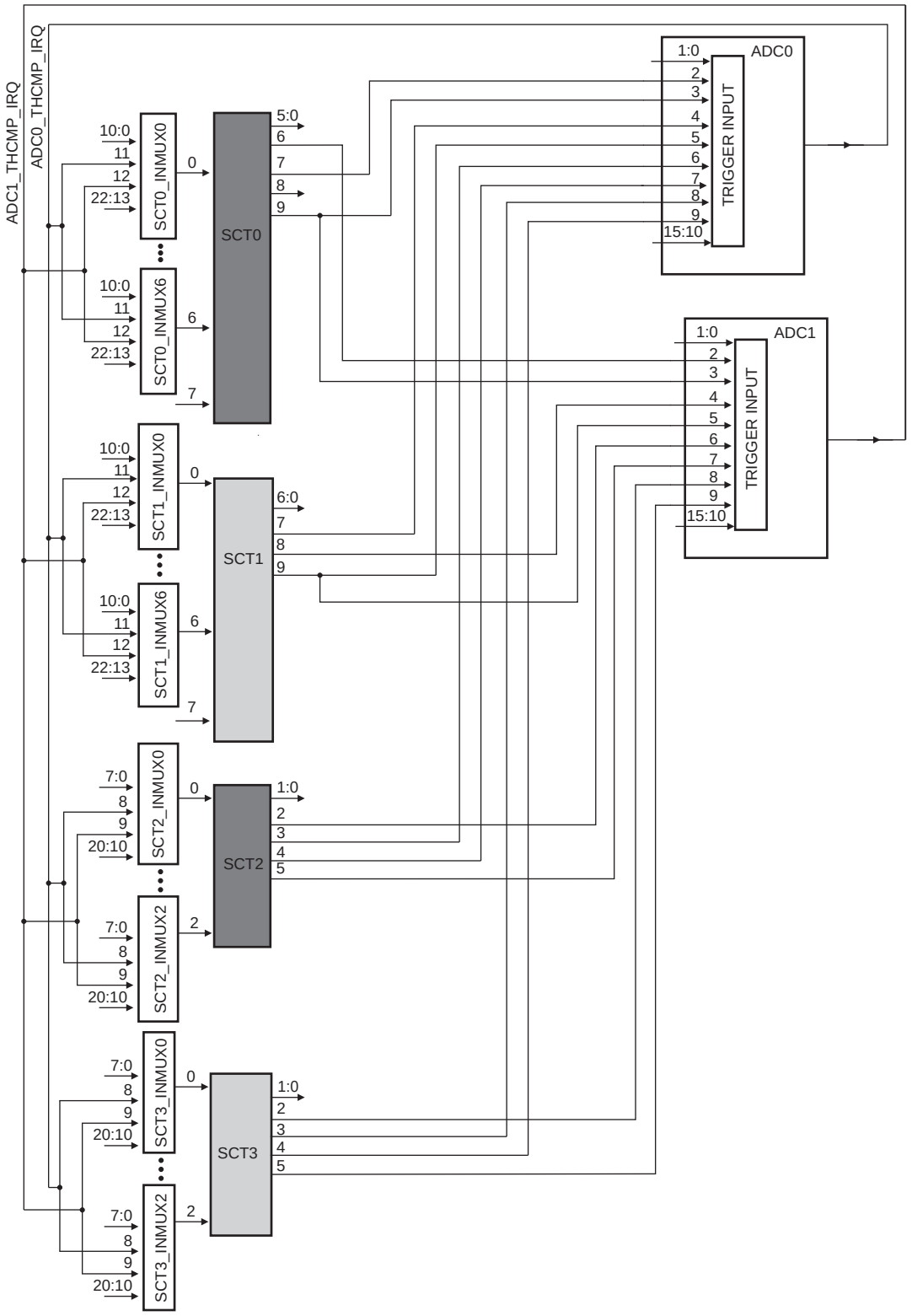


Fig 31. SCT to ADC connections

15.3.4 Use the SCT as a simple timer

To configure the SCT as a simple timer with match or capture functionality, follow these steps:

1. Set up the SCT as one 32-bit timer or one or two 16-bit timers. See Table 202.
2. Preload the 32-bit timer or the 16-bit timers with a count value. See Table 209.
3. If you want to create a match event when the timer reaches a match value:
   a. Configure the register map for match registers. See Table 212.
   b. Configure one or more match registers with a match value. See Table 222.
   c. For each match value, create a match event. See Table 229.
   d. If you want to create an interrupt on a match event, enable the event for interrupt. See Table 219.
   e. If you want to create a match output on a pin, connect the SCTn_OUTn function to a pin (see Section 15.4) and select an output for the match event in the EVn_CTRL register. See Table 229. The EVn_CTRL registers also control what type of output signal is created.
4. If you want to capture a timer value on a capture signal:
   a. Configure the register map for capture registers. See Table 212.
   b. Create one or more capture events. See Table 229.
   c. Connect the SCT_IN functions to pins (see Section 15.4) and configure the signal to create an event. See Table 229.
5. Start the timer by writing to the CRTL register. See Table 203.
6. Read the capture registers to read the timer value at the time of the capture events.

15.3.5 Use the SCT with the SCTIPU

The SCTIPU provides a pre-processing unit for the SCT inputs for two purposes:

- Combine signal transitions to one transition that always generates the same SCT response. An example is aborting an SCT operation. See Section 14.5.1
- Provide latched inputs so that signals can be blocked for some fixed time before being passed on to the SCT inputs. See Section 14.5.2.

15.3.5.1 Abort function

A set of transitions on the SCT_ABORT pins, the ADC threshold compare outputs, the comparator outputs, or the SCT0 output 9 can be logically combined in the SCTIPU so that any transition in the set triggers the same SCT response. The pre-processing replaces many individual signal transitions from multiple SCT inputs by one transition monitored on one input.

Remark: Signals on the SCT_ABORT pins must be active HIGH or rising edge and the SCT0_OUT9 output must also be configured for active HIGH/rising edge if used. The ADC interrupts and the comparator outputs are active HIGH.

To set up an abort function for SCT0 on input 5 SCT0_IN5, follow these steps:

1. In the SCTIPU ABORT_ENABLE0 register, enable one or more inputs. Any LOW-HIGH transition on the enabled input produces a rising edge on the output of the abort module 0.
2. If you want to use the SCT_ABORT0/SCT_ABORT1 pin function, connect pin functions to a pin using the switch matrix PINASSIGN10 register.
3. In the SCT0_INMUX5 register, select the SCTIPU_ABORT input (#17).
4. Set up the SCT0 by defining SCT0_IN5 as rising edge or HIGH input and configure an event that is triggered on SCT0_IN5. There are many possible actions in response to this event:
   - Stop counting.
   - Change the count direction.
   - Force any of the SCT0 outputs HIGH or LOW.

Remark: The same procedure can be used for applications other than an abort whenever an SCT response to a fixed set of input transitions is required.

15.4 Pin description

See Section 8.3.1 “Connect an internal signal to a package pin” to assign the SCT functions to external pins.

Remark: For applications that require exact timing of the SCT outputs (for example PWM), assign the outputs only to fixed-pin functions to ensure that the output skew is nearly the same for all outputs.

All SCT input signals are selected through the input mux except for input 7 which is connected to the SCT PLL output. The signals from the external pins selected in the input mux registers are connected directly to the SCT inputs and not routed through the switch matrix.

SCT outputs are routed to multiple places and can be connected to pins and inputs of the other SCTs, SCTIPU, and ADC triggers at the same time.

Table 199. SCT0/1 pin description (inputs)

Function	Type	Connect to	Use register	Reference
SCT0_IN[0:6]	external to pins or internal	one of the following: 8 pins, SCT1_OUT4, SCT2_OUT[5:4], ADC0_THCMP_IRQ, ADC1_THCMP_IRQ, ACMP_OUT[3:0], SCT_IPU outputs, DEBUG_HALTED	SCT0_INMUX[0:6]	Table 127
SCT0_IN7	internal	SCT PLL output	n/a	-
SCT1_IN[0:6]	external to pins or internal	one of the following: 8 pins, SCT0_OUT4, SCT3_OUT[5:4], ADC0_THCMP_IRQ, ADC1_THCMP_IRQ, ACMP_OUT[3:0], SCT_IPU outputs, DEBUG_HALTED	SCT1_INMUX[0:6]	Table 128
SCT1_IN7	internal	SCT PLL output	n/a	-

Table 200. SCT0/1 pin description (outputs)

Function	Type	Connect to	Use register	Reference
SCT0_OUT[0:2]	external to pin	any pin	PINASSIGN7	Table 114
SCT0_OUT3	external to pin	PIO0_0	PINENABLE1	Table 124
SCT0_OUT4	external to pin	PIO0_1	PINENABLE1	Table 124
	internal	SCT1 input mux	SCT1_INMUX[6:0]	Table 128
	internal	SCT2 input mux	SCT2_INMUX[2:0]	Table 129
SCT0_OUT5	external to pin	PIO0_18	PINENABLE1	Table 124
	internal	SCTIPU	-	-
	internal	SCT2 input mux	SCT2_INMUX[2:0]	Table 129
SCT0_OUT6	external to pin	PIO0_24	PINENABLE1	Table 124
	internal	ADC1 trigger	SEQ_A, SEQ_B	Table 437, Table 438
	internal	SCT2 input mux	SCT2_INMUX[2:0]	Table 129
	internal	ADC0 trigger	SEQ_A, SEQ_B	Table 437, Table 438
SCT0_OUT7	external to pin	PIO1_14	PINENABLE1	Table 124
	internal	ADC0 trigger	SEQ_A, SEQ_B	Table 437, Table 438
	internal	SCT2 input mux	SCT2_INMUX[2:0]	Table 129
SCT0_OUT8	external to pin	n/a	n/a	-
	internal	SCT2 input mux	SCT2_INMUX[2:0]	Table 129

Table 200. SCT0/1 pin description (outputs)

Function	Type	Connect to	Use register	Reference
SCT0_OUT9	external to pin	n/a	n/a	-
	internal	ADC0 trigger	SEQ_A, SEQ_B	Table 437, Table 438
	internal	ADC1 trigger	SEQ_A, SEQ_B	Table 437, Table 438
	internal	SCTIPU abort input 2	ABORT_ENABLE	Table 197
SCT1_OUT[0:2]	external to pin	any pin	PINASSIGN8	Table 114
SCT1_OUT3	external to pin	PIO0_2	PINENABLE1	Table 124
SCT1_OUT4	external to pin	PIO0_3	PINENABLE1	Table 124
	internal	SCT0 input mux	SCT0_INMUX[6:0]	Table 127
	internal	SCT3 input mux	SCT3_INMUX[2:0]	Table 130
SCT1_OUT5	external to pin	PIO0_14	PINENABLE1	Table 124
	internal	SCTIPU	-	-
	internal	SCT3 input mux	SCT3_INMUX[2:0]	Table 130
SCT1_OUT6	external to pin	PIO0_20	PINENABLE1	Table 124
SCT1_OUT7	external to pin	PIO1_17	PINENABLE1	Table 124
	internal	SCT3_INMUX[2:0]	SCT3_INMUX[2:0]	Table 130
	internal	ADC0 trigger	SEQ_A, SEQ_B	Table 437, Table 438
SCT1_OUT8	external to pin	n/a	-	-
	internal	SCT3_INMUX[2:0]	SCT3_INMUX[2:0]	Table 130
	internal	ADC1 trigger	SEQ_A, SEQ_B	Table 437, Table 438
SCT1_OUT9	external to pin	n/a	-	-
	internal	ADC0 trigger	SEQ_A, SEQ_B	Table 437, Table 438
	internal	ADC1 trigger	SEQ_A, SEQ_B	Table 437, Table 438

15.5 General description

The State Configurable Timer (SCT) allows a wide variety of timing, counting, output modulation, and input capture operations.

The most basic user-programmable option is whether a SCT operates as two 16-bit counters or a unified 32-bit counter. In the two-counter case, in addition to the counter value the following operational elements are independent for each half:

- State variable
- Limit, halt, stop, and start conditions
- Values of Match/Capture registers, plus reload or capture control values

In the two-counter case, the following operational elements are global to the SCT:

- Clock selection

- Inputs
- Events
- Outputs
- Interrupts

Events, outputs, and interrupts can use match conditions from either counter.

Remark: In this chapter, the term bus error indicates an SCT response that makes the processor take an exception.

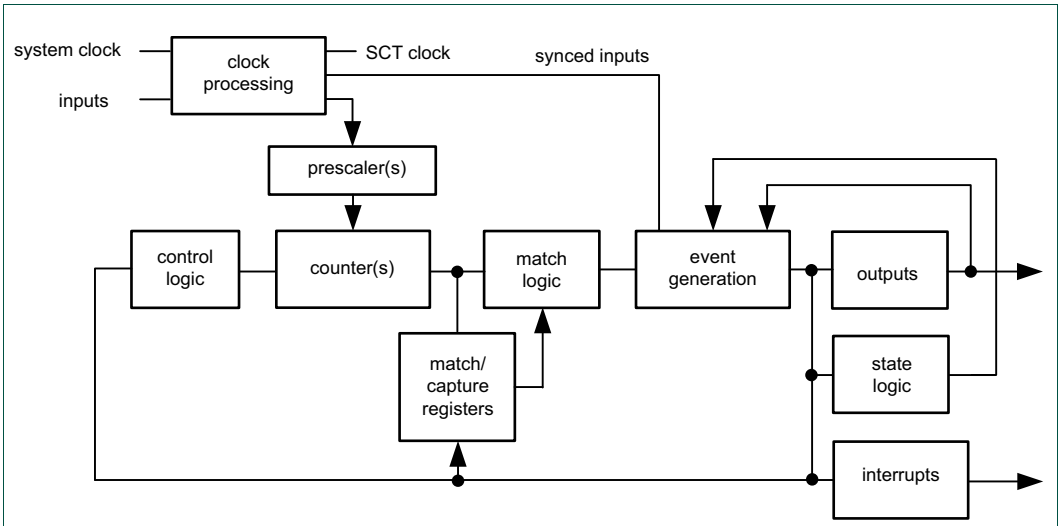


Fig 32. SCT block diagram

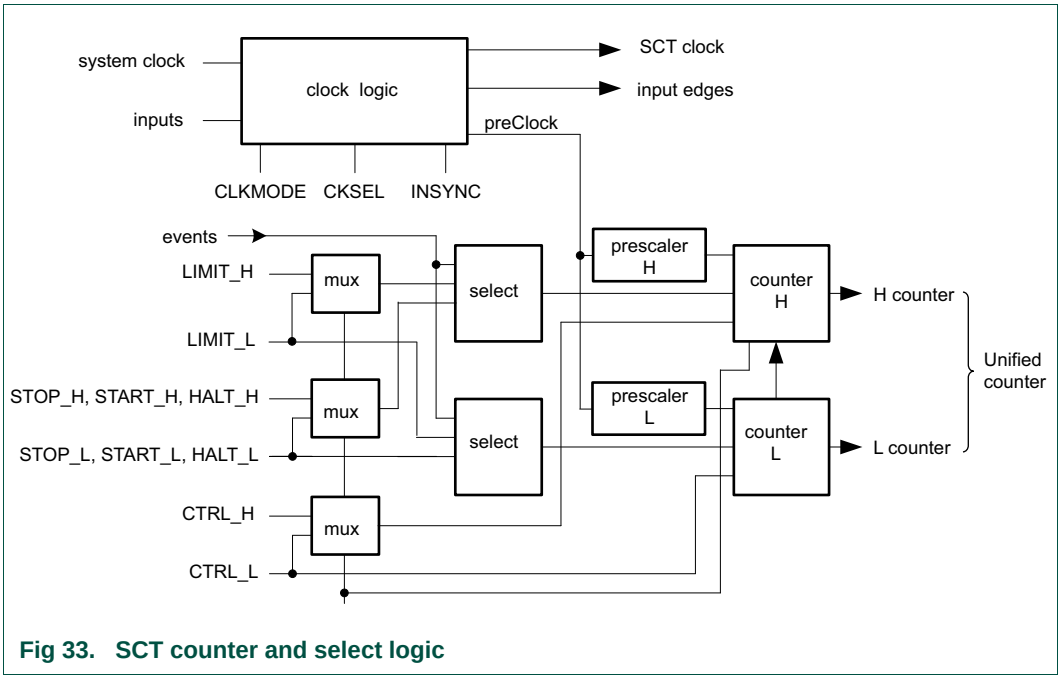


Fig 33. SCT counter and select logic

15.6 Register description

The register addresses of the State Configurable Timer are shown in Table 201. For most of the SCT registers, the register function depends on the setting of certain other register bits:

1. The UNIFY bit in the CONFIG register determines whether the SCT is used as one 32-bit register (for operation as one 32-bit counter/timer) or as two 16-bit counter/timers named L and H. The setting of the UNIFY bit is reflected in the register map:
   - UNIFY = 1: Only one register is used (for operation as one 32-bit counter/timer).
   - UNIFY = 0: Access the L and H registers by a 32-bit read or write operation or can be read or written to individually (for operation as two 16-bit counter/timers).

   Typically, the UNIFY bit is configured by writing to the CONFIG register before any other registers are accessed.
2. The REGMODEn bits in the REGMODE register determine whether each set of Match/Capture registers uses the match or capture functionality:
   - REGMODEn = 1: Registers operate as match and reload registers.
   - REGMODEn = 0: Registers operate as capture and capture control registers.

Table 201. Register overview: State Configurable Timer (base address 0x1C01 8000 (SCT0) and 0x1C01 C000 (SCT1))

Name	Access	Address offset	Description	Reset value	Reference
CONFIG	R/W	0x000	SCT configuration register	0x0000 7E00	Table 202
CTRL	R/W	0x004	SCT control register	0x0004 0004	Table 203
CTRL_L	R/W	0x004	SCT control register low counter 16-bit	0x0004 0004	Table 203
CTRL_H	R/W	0x006	SCT control register high counter 16-bit	0x0004 0004	Table 203
LIMIT	R/W	0x008	SCT limit register	0x0000 0000	Table 204
LIMIT_L	R/W	0x008	SCT limit register low counter 16-bit	0x0000 0000	Table 204
LIMIT_H	R/W	0x00A	SCT limit register high counter 16-bit	0x0000 0000	Table 204
HALT	R/W	0x00C	SCT halt condition register	0x0000 0000	Table 205
HALT_L	R/W	0x00C	SCT halt condition register low counter 16-bit	0x0000 0000	Table 205
HALT_H	R/W	0x00E	SCT halt condition register high counter 16-bit	0x0000 0000	Table 205
STOP	R/W	0x010	SCT stop condition register	0x0000 0000	Table 206
STOP_L	R/W	0x010	SCT stop condition register low counter 16-bit	0x0000 0000	Table 206
STOP_H	R/W	0x012	SCT stop condition register high counter 16-bit	0x0000 0000	Table 206
START	R/W	0x014	SCT start condition register	0x0000 0000	Table 207
START_L	R/W	0x014	SCT start condition register low counter 16-bit	0x0000 0000	Table 207
START_H	R/W	0x016	SCT start condition register high counter 16-bit	0x0000 0000	Table 207
DITHER	R/W	0x018	SCT dither condition register		Table 208
DITHER_L	R/W	0x018	SCT dither condition register low counter 16-bit		Table 208
DITHER_H	R/W	0x01A	SCT dither condition register high counter 16-bit		Table 208
-	-	0x01C - 0x03C	Reserved		-
COUNT	R/W	0x040	SCT counter register	0x0000 0000	Table 209

Table 201. Register overview: State Configurable Timer (base address 0x1C01 8000 (SCT0) and 0x1C01 C000 (SCT1)) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
COUNT_L	R/W	0x040	SCT counter register low counter 16-bit	0x0000 0000	Table 209
COUNT_H	R/W	0x042	SCT counter register high counter 16-bit	0x0000 0000	Table 209
STATE	R/W	0x044	SCT state register	0x0000 0000	Table 210
STATE_L	R/W	0x044	SCT state register low counter 16-bit	0x0000 0000	Table 210
STATE_H	R/W	0x046	SCT state register high counter 16-bit	0x0000 0000	Table 210
INPUT	RO	0x048	SCT input register	0x0000 0000	Table 211
REGMODE	R/W	0x04C	SCT match/capture registers mode register	0x0000 0000	Table 212
REGMODE_L	R/W	0x04C	SCT match/capture registers mode register low counter 16-bit	0x0000 0000	Table 212
REGMODE_H	R/W	0x04E	SCT match/capture registers mode register high counter 16-bit	0x0000 0000	Table 212
OUTPUT	R/W	0x050	SCT output register	0x0000 0000	Table 213
OUTPUTDIRCTRL	R/W	0x054	SCT output counter direction control register	0x0000 0000	Table 214
RES	R/W	0x058	SCT conflict resolution register	0x0000 0000	Table 215
DMAREQ0	R/W	0x05C	SCT DMA request 0 register	0x0000 0000	Table 216
DMAREQ1	R/W	0x060	SCT DMA request 1 register	0x0000 0000	Table 217
-	-	0x064 - 0x0EC	Reserved	-	-
EVEN	R/W	0x0F0	SCT event enable register	0x0000 0000	Table 218
EVFLAG	R/W	0x0F4	SCT event flag register	0x0000 0000	Table 219
CONEN	R/W	0x0F8	SCT conflict enable register	0x0000 0000	Table 220
CONFLAG	R/W	0x0FC	SCT conflict flag register	0x0000 0000	Table 221
MATCH0 to MATCH15	R/W	0x100 to 0x13C	SCT match value register of match channels 0 to 15; REGMOD0 to REGMODE15 = 0	0x0000 0000	Table 221
MATCH0_L to MATCH15_L	R/W	0x100 to 0x13C	SCT match value register of match channels 0 to 15; low counter 16-bit; REGMOD0_L to REGMODE15_L = 0	0x0000 0000	Table 221
MATCH0_H to MATCH15_H	R/W	0x102 to 0x13E	SCT match value register of match channels 0 to 15; high counter 16-bit; REGMOD0_H to REGMODE15_H = 0	0x0000 0000	Table 221
CAP0 to CAP15	R	0x100 to 0x13C	SCT capture register of capture channel 0 to 15; REGMOD0 to REGMODE15 = 1	0x0000 0000	Table 224
CAP0_L to CAP15_L	R	0x100 to 0x13C	SCT capture register of capture channel 0 to 15; low counter 16-bit; REGMOD0_L to REGMODE15_L = 1	0x0000 0000	Table 224
CAP0_H to CAP15_H	R	0x102 to 0x13E	SCT capture register of capture channel 0 to 15; high counter 16-bit; REGMOD0_H to REGMODE15_H = 1	0x0000 0000	Table 224
FRACMAT0 to 5	R/W	0x140 to 0x154	Fractional match registers 0 to 5 for SCT match value registers 0 to 5.	0x0000 0000	Table 223
FRACMAT0_L to FRACMAT5_L	R/W	0x140 to 0x154	Fractional match registers 0 to 5 for SCT match value registers 0 to 5; low counter 16-bit.	0x0000 0000	Table 223
FRACMAT0_H to FRACMAT5_H	R/W	0x142 to 0x156	Fractional match registers 0 to 5 for SCT match value registers 0 to 5; high counter 16-bit.	0x0000 0000	Table 223

Table 201. Register overview: State Configurable Timer (base address 0x1C01 8000 (SCT0) and 0x1C01 C000 (SCT1)) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
MATCHREL0 to MATCHREL15	R/W	0x200 to 0x23C	SCT match reload value register 0 to 15; REGMOD0 = 0 to REGMODE15 = 0	0x0000 0000	Table 225
MATCHREL0_L to MATCHREL15_L	R/W	0x200 to 0x23C	SCT match reload value register 0 to 15; low counter 16-bit; REGMOD0_L = 0 to REGMODE15_L = 0	0x0000 0000	Table 225
MATCHREL0_H to MATCHREL15_H	R/W	0x202 to 0x23E	SCT match reload value register 0 to 15; high counter 16-bit; REGMOD0_H = 0 to REGMODE15_H = 0	0x0000 0000	Table 225
CAPCTRL0 to CAPCTRL15	R/W	0x200 to 0x23C	SCT capture control register 0 to 15; REGMOD0 = 1 to REGMODE15 = 1	0x0000 0000	Table 227
CAPCTRL0_L to CAPCTRL15_L	R/W	0x200 to 0x23C	SCT capture control register 0 to 15; low counter 16-bit; REGMOD0_L = 1 to REGMODE15_L = 1	0x0000 0000	Table 227
CAPCTRL0_H to CAPCTRL15_H	R/W	0x202 to 0x23E	SCT capture control register 0 to 15; high counter 16-bit; REGMOD0 = 1 to REGMODE15 = 1	0x0000 0000	Table 227
FRACMATREL0 to FRACMATREL5	R/W	0x240 to 0x254	Fractional match reload registers 0 to 5 for SCT match value registers 0 to 5.	0x0000 0000	Table 226
FRACMATREL0_L to FRACMATREL5_L	R/W	0x240 to 0x254	Fractional match reload registers 0 to 5 for SCT match value registers 0 to 5; low counter 16-bit.	0x0000 0000	Table 226
FRACMATREL0_H to FRACMATREL5_H	R/W	0x242 to 0x256	Fractional match reload registers 0 to 5 for SCT match value registers 0 to 5; high counter 16-bit.	0x0000 0000	Table 226
EV0_STATE	R/W	0x300	SCT event state register 0	0x0000 0000	Table 228
EV0_CTRL	R/W	0x304	SCT event control register 0	0x0000 0000	Table 229
EV1_STATE	R/W	0x308	SCT event state register 1	0x0000 0000	Table 228
EV1_CTRL	R/W	0x30C	SCT event control register 1	0x0000 0000	Table 229
EV2_STATE	R/W	0x310	SCT event state register 2	0x0000 0000	Table 228
EV2_CTRL	R/W	0x314	SCT event control register 2	0x0000 0000	Table 229
EV3_STATE	R/W	0x318	SCT event state register 3	0x0000 0000	Table 228
EV3_CTRL	R/W	0x31C	SCT event control register 3	0x0000 0000	Table 229
EV4_STATE	R/W	0x320	SCT event state register 4	0x0000 0000	Table 228
EV4_CTRL	R/W	0x324	SCT event control register4	0x0000 0000	Table 229
EV5_STATE	R/W	0x328	SCT event state register 5	0x0000 0000	Table 228
EV5_CTRL	R/W	0x32C	SCT event control register 5	0x0000 0000	Table 229
EV6_STATE	R/W	0x330	SCT event state register 6	0x0000 0000	Table 228
EV6_CTRL	R/W	0x334	SCT event control register 6	0x0000 0000	Table 229
EV7_STATE	R/W	0x338	SCT event state register 7	0x0000 0000	Table 228
EV7_CTRL	R/W	0x33C	SCT event control register 7	0x0000 0000	Table 229
EV8_STATE	R/W	0x340	SCT event state register 8	0x0000 0000	Table 228
EV8_CTRL	R/W	0x344	SCT event control register 8	0x0000 0000	Table 229
EV9_STATE	R/W	0x348	SCT event state register 9	0x0000 0000	Table 228
EV9_CTRL	R/W	0x34C	SCT event control register 9	0x0000 0000	Table 229
EV10_STATE	R/W	0x350	SCT event state register 10	0x0000 0000	Table 228
EV10_CTRL	R/W	0x354	SCT event control register 10	0x0000 0000	Table 229

Table 201. Register overview: State Configurable Timer (base address 0x1C01 8000 (SCT0) and 0x1C01 C000 (SCT1)) *…continued*

Name	Access	Address offset	Description	Reset value	Reference
EV11_STATE	R/W	0x358	SCT event state register 11	0x0000 0000	Table 228
EV11_CTRL	R/W	0x35C	SCT event control register 11	0x0000 0000	Table 229
EV12_STATE	R/W	0x360	SCT event state register 12	0x0000 0000	Table 228
EV12_CTRL	R/W	0x364	SCT event control register 12	0x0000 0000	Table 229
EV13_STATE	R/W	0x368	SCT event state register 13	0x0000 0000	Table 228
EV13_CTRL	R/W	0x36C	SCT event control register 13	0x0000 0000	Table 229
EV14_STATE	R/W	0x370	SCT event state register 14	0x0000 0000	Table 228
EV14_CTRL	R/W	0x374	SCT event control register 14	0x0000 0000	Table 229
EV15_STATE	R/W	0x378	SCT event state register 15	0x0000 0000	Table 228
EV15_CTRL	R/W	0x37C	SCT event control register 15	0x0000 0000	Table 229
OUT0_SET	R/W	0x500	SCT output 0 set register	0x0000 0000	Table 230
OUT0_CLR	R/W	0x504	SCT output 0 clear register	0x0000 0000	Table 231
OUT1_SET	R/W	0x508	SCT output 1 set register	0x0000 0000	Table 230
OUT1_CLR	R/W	0x50C	SCT output 1 clear register	0x0000 0000	Table 231
OUT2_SET	R/W	0x510	SCT output 2 set register	0x0000 0000	Table 230
OUT2_CLR	R/W	0x514	SCT output 2 clear register	0x0000 0000	Table 231
OUT3_SET	R/W	0x518	SCT output 3 set register	0x0000 0000	Table 230
OUT3_CLR	R/W	0x51C	SCT output 3 clear register	0x0000 0000	Table 231
OUT4_SET	R/W	0x520	SCT output 4 set register	0x0000 0000	Table 230
OUT4_CLR	R/W	0x524	SCT output 4 clear register	0x0000 0000	Table 231
OUT5_SET	R/W	0x528	SCT output 5 set register	0x0000 0000	Table 230
OUT5_CLR	R/W	0x52C	SCT output 5 clear register	0x0000 0000	Table 231
OUT6_SET	R/W	0x530	SCT output 6 set register	0x0000 0000	Table 230
OUT6_CLR	R/W	0x534	SCT output 6 clear register	0x0000 0000	Table 231
OUT7_SET	R/W	0x538	SCT output 7 set register	0x0000 0000	Table 230
OUT7_CLR	R/W	0x53C	SCT output 7 clear register	0x0000 0000	Table 231
OUT8_SET	R/W	0x540	SCT output 8 set register	0x0000 0000	Table 230
OUT8_CLR	R/W	0x544	SCT output 8 clear register	0x0000 0000	Table 231
OUT9_SET	R/W	0x548	SCT output 9 set register	0x0000 0000	Table 230
OUT9_CLR	R/W	0x54C	SCT output 9 clear register	0x0000 0000	Table 231

15.6.1 SCT configuration register

This register configures the overall operation of the SCT. Write to this register before any other registers.

Table 202. SCT configuration register (CONFIG, address 0x1C01 8000) bit description

Bit	Symbol	Value	Description	Reset value
0	UNIFY		SCT operation	0
		0	The SCT operates as two 16-bit counters named L and H.	
		1	The SCT operates as a unified 32-bit counter.	
2:1	CLKMODE		SCT clock mode	00
		0x0	System clock. The system clock clocks the SCT and prescalers.	
		0x1	Prescaled system clock. The SCT clock is the system clock, but the prescalers are enabled to count only when sampling of the input selected by the CKSEL field finds the selected edge. The minimum pulse width on the clock input is 1 bus clock period. This mode is the high-performance sampled-clock mode.	
		0x2	SCT input. The input selected by CKSEL clocks the SCT and prescalers. The input is synchronized to the system clock and possibly inverted. The minimum pulse width on the clock input is 1 bus clock period. This mode is the low-power sampled-clock mode.	
		0x3	Prescaled SCT input. The SCT and prescalers are clocked by the input edge selected by the CKSEL field. In this mode, most of the SCT is clocked by the (selected polarity of the) input. The outputs are switched synchronously to the input clock. The input clock rate must be at least half the system clock rate and can be the same or faster than the system clock.	
6:3	CKSEL		SCT clock select	0000
		0x0	Rising edges on input 0.	
		0x1	Falling edges on input 0.	
		0x2	Rising edges on input 1.	
		0x3	Falling edges on input 1.	
		0x4	Rising edges on input 2.	
		0x5	Falling edges on input 2.	
		0x6	Rising edges on input 3.	
		0x7	Falling edges on input 3.	
		0x8	Rising edges on input 4.	
		0x9	Falling edges on input 4.	
		0xA	Rising edges on input 5.	
		0xB	Falling edges on input 5.	
		0xC	Rising edges on input 6.	
		0xD	Falling edges on input 6.	
		0xE	Rising edges on input 7.	
		0xF	Falling edges on input 7.	
7	NORELAOD_L	-	A 1 in this bit prevents the lower match and fractional match registers from being reloaded from their respective reload registers. Software can write to set or clear this bit at any time. This bit applies to both the higher and lower registers when the UNIFY bit is set.	0
8	NORELOAD_H	-	A 1 in this bit prevents the higher match and fractional match registers from being reloaded from their respective reload registers. Software can write to set or clear this bit at any time. This bit is not used when the UNIFY bit is set.	0

Table 202. SCT configuration register (CONFIG, address 0x1C01 8000) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
16:9	INSYNC	-	Synchronization for input n (bit 9 = input 0, bit 10 = input 1,..., bit 16 = input 7). A 1 in one of these bits subjects the corresponding input to synchronization to the SCT clock, before it is used to create an event. If an input is synchronous to the SCT clock, keep its bit 0 for faster response. When the CKMODE field is 1x, the bit in this field, corresponding to the input selected by the CKSEL field, is not used.	1
17	AUTOLIMIT_L	-	A one in this bit causes a match on match register 0 to be treated as a de-facto LIMIT condition without the need to define an associated event. As with any LIMIT event, this automatic limit causes the counter to be cleared to zero in uni-directional mode or to change the direction of count in bi-directional mode. Software can write to set or clear this bit at any time. This bit applies to both the higher and lower registers when the UNIFY bit is set.	0
18	AUTOLIMIT_H	-	A one in this bit will cause a match on match register 0 to be treated as a de-facto LIMIT condition without the need to define an associated event. As with any LIMIT event, this automatic limit causes the counter to be cleared to zero in uni-directional mode or to change the direction of count in bi-directional mode. Software can write to set or clear this bit at any time. This bit is not used when the UNIFY bit is set.	0
31:19	-		Reserved	-

15.6.2 SCT control register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers CTRL_L and CTRL_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

All bits in this register can be written to when the counter is stopped or halted. When the counter is running, the only bits that can be written are STOP or HALT. (Other bits can be written in a subsequent write after HALT is set to 1.)

Remark: If CLKMODE = 0x3 is selected, wait at least 12 system clock cycles between a write access to the H, L or unified version of this register and the next write access. This restriction does not apply when writing to the HALT bit or bits and then writing to the CTRL register again to restart the counters - for example because software must update the MATCH register, which is only allowed when the counters are halted.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 203. SCT control register (CTRL, address 0x1C01 8004) bit description

Bit	Symbol	Value	Description	Reset value
0	DOWN_L	-	This bit is 1 when the L or unified counter is counting down. Hardware sets this bit when the counter limit is reached and BIDIR is 1. Hardware clears this bit when the counter reaches 0 or when the counter is counting down and a limit condition occurs.	0
1	STOP_L	-	When this bit is 1 and HALT is 0, the L or unified counter does not run but I/O events related to the counter can occur. If such an event matches the mask in the Start register, this bit is cleared and counting resumes.	0
2	HALT_L	-	When this bit is 1, the L or unified counter does not run and no events related to the L-counter can occur. A reset sets this bit. When the HALT_L bit is one, the STOP_L bit is cleared. If you want to remove the halt condition and keep the SCT in the stop condition (not running), then you can change the halt and stop condition with one single write to this register. **Remark:** Once set, only software can clear this bit to restore counter operation.	1
3	CLRCTR_L	-	Writing a 1 to this bit clears the L or unified counter. This bit always reads as 0.	0
4	BIDIR_L		L or unified counter direction select	0
		0	The counter counts up to its limit condition, then is cleared to zero.	
		1	The counter counts up to its limit, then counts down to a limit condition or to 0.	
12:5	PRE_L	-	Specifies the factor by which the SCT clock is prescaled to produce the L or unified counter clock. The counter clock is clocked at the rate of the SCT clock divided by PRE_L+1. **Remark:** Clear the counter (by writing a 1 to the CLRCTR bit) whenever changing the PRE value.	0
15:13	-		Reserved	
16	DOWN_H	-	This bit is 1 when the H counter is counting down. Hardware sets this bit when the counter limit is reached and BIDIR is 1. Hardware clears this bit when the counter reaches 0 or when the counter is counting down and a limit condition occurs.	0
17	STOP_H	-	When this bit is 1 and HALT is 0, the H counter does not run but I/O events related to the counter can occur. If such an event matches the mask in the Start register, this bit is cleared and counting resumes.	0
18	HALT_H	-	When this bit is 1, the H counter does not run and no events related to the H counter can occur. A reset sets this bit. When the HALT_H bit is one, the STOP_H bit is cleared. If you want to remove the halt condition and keep the SCT in the stop condition (not running), then you can change the halt and stop condition with one single write to this register. **Remark:** Once set, this bit can only be cleared by software to restore counter operation.	1
19	CLRCTR_H	-	Writing a 1 to this bit clears the H counter. This bit always reads as 0.	0
20	BIDIR_H		Direction select	0
		0	The H counter counts up to its limit condition, then is cleared to zero.	
		1	The H counter counts up to its limit, then counts down to a limit condition or to 0.	
28:21	PRE_H	-	Specifies the factor by which the SCT clock is prescaled to produce the H counter clock. The counter clock is clocked at the rate of the SCT clock divided by PRELH+1. **Remark:** Clear the counter (by writing a 1 to the CLRCTR bit) whenever changing the PRE value.	0
31:29	-		Reserved	

15.6.3 SCT limit register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers LIMIT_L and LIMIT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

The bits in this register set which events act as counter limits. After a counter has reached its limit, the counter is cleared to zero in unidirectional mode or changes its direction of count in bidirectional mode. When the counter reaches all ones, this state is always treated as a limit event, and the counter is cleared in unidirectional mode or, in bidirectional mode, begins counting down on the next clock edge - even if no limit event as defined by the SCT limit register has occurred.

In addition to using this register to specify events that serve as limits, it is also possible to automatically cause a limit condition whenever a match register 0 match occurs. This eliminates the need to define an event for the sole purpose of creating a limit. The AUTOLIMIT_L and AUTOLIMIT_H bits in the configuration register enable/disable this feature (see Table 202).

Table 204. SCT limit register (LIMIT, address 0x1C01 8008) bit description

Bit	Symbol	Description	Reset value
15:0	LIMMSK_L	If bit n is one, event n is used as a counter limit event for the L or unified counter (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	LIMMSK_H	If bit n is one, event n is used as a counter limit event for the H counter (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

15.6.4 SCT halt condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers HALT_L and HALT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Remark: Any event halting the counter disables its operation until software clears the HALT bit (or bits) in the CTRL register (Table 203).

Table 205. SCT halt condition register (HALT, address 0x1C01 800C) bit description

Bit	Symbol	Description	Reset value
15:0	HALTMSK_L	If bit n is one, event n sets the HALT_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	HALTMSK_H	If bit n is one, event n sets the HALT_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

15.6.5 SCT stop condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers STOP_L and STOP_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Table 206. SCT stop condition register (STOP, address 0x1C01 8010) bit description

Bit	Symbol	Description	Reset value
15:0	STOPMSK_L	If bit n is one, event n sets the STOP_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	STOPMSK_H	If bit n is one, event n sets the STOP_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

15.6.6 SCT start condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers START_L and START_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

The bits in this register select which events, if any, clear the STOP bit in the Control register. (Since no events can occur when HALT is 1, only software can clear the HALT bit by writing the Control register.)

Table 207. SCT start condition register (START, address 0x1C01 8014) bit description

Bit	Symbol	Description	Reset value
15:0	STARTMSK_L	If bit n is one, event n clears the STOP_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	STARTMSK_H	If bit n is one, event n clears the STOP_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

15.6.7 SCT dither condition register

If UNIFY = 1 in the CONFIG register, only the L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers DITHER_L and DITHER_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

When the Dither Condition register contains all zeroes (the default value), the dither engine advances to the next count in the dither pattern **every time** the SCT counter reaches zero (i.e. at the start of every new SCT counter cycle).

It is possible, using this register, to alter that behavior by qualifying the advancement through the dither pattern with designated events. As with the other condition/mask registers (HALT, STOP, LIMIT, etc.) each bit in this register corresponds to an event.

Setting one or more of the bits in this register to ones will cause the dither engine to advance to the next element in the dither pattern (i.e. increment the 16-state cycle counter) only following SCT counter cycles during which one or more of the designated dither events have occurred.

There is one, global Dither Condition register per 16-bit SCT. This register controls advancement through the dither patterns for all of the match registers associated with that half of the SCT.

For details on the dither engine and the dither pattern, see Section 15.7.9.1.

Table 208. SCT dither condition register (DITHER, address 0x1C01 8018) bit description

Bit	Symbol	Description	Reset value
15:0	DITHMSK_L	If bit n is one, the event n causes the dither engine to advance to the next element in the dither pattern at the start of the next counter cycle of the 16-bit low counter or the unified counter (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15). If all bits are set to 0, the dither pattern automatically advances at the start of every new counter cycle.	0
31:16	DITHMSK_H	If bit n is one, the event n causes the dither engine to advance to the next element in the dither pattern at the start of the next counter cycle of the 16-bit high counter (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15). If all bits are set to 0, the dither pattern automatically advances at the start of every new counter cycle.	0

15.6.8 SCT counter register

If UNIFY = 1 in the CONFIG register, the counter is a unified 32-bit register and both the _L and _H bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers COUNT_L and COUNT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation. In this case, the L and H registers count independently under the control of the other registers.

Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register). Software can read the counter registers at any time.

Table 209. SCT counter register (COUNT, address 0x1C01 8040) bit description

Bit	Symbol	Description	Reset value
15:0	CTR_L	When UNIFY = 0, read or write the 16-bit L counter value. When UNIFY = 1, read or write the lower 16 bits of the 32-bit unified counter.	0
31:16	CTR_H	When UNIFY = 0, read or write the 16-bit H counter value. When UNIFY = 1, read or write the upper 16 bits of the 32-bit unified counter.	0

15.6.9 SCT state register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers STATE_L and STATE_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Software can read the state associated with a counter at any time. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

The state variable is the main feature that distinguishes the SCT from other counter/timer/PWM blocks. Events can be made to occur only in certain states. Events, in turn, can perform the following actions:

- set and clear outputs
- limit, stop, and start the counter
- cause interrupts and DMA requests
- modify the state variable

The value of a state variable is completely under the control of the application. If an application does not use states, the value of the state variable remains zero, which is the default value.

A state variable can be used to track and control multiple cycles of the associated counter in any desired operational sequence. The state variable is logically associated with a state machine diagram which represents the SCT configuration. See Section 15.6.26 and 15.6.27 for more about the relationship between states and events.

The STATELD/STADEV fields in the event control registers of all defined events set all possible values for the state variable. The change of the state variable during multiple counter cycles reflects how the associated state machine moves from one state to the next.

Table 210. SCT state register (STATE, address 0x1C01 8044) bit description

Bit	Symbol	Description	Reset value
4:0	STATE_L	State variable.	0
15:5	-	Reserved.	-
20:16	STATE_H	State variable.	0
31:21	-	Reserved.	

15.6.10 SCT input register

Software can read the state of the SCT inputs in this read-only register in slightly different forms.

1. The AIN bit represents the input sampled by the SCT clock. This corresponds to a nearly direct read-out of the input but can cause spurious fluctuations in case of an asynchronous input signal.
2. The SIN bit represents the input sampled by the SCT clock after the INSYNC select (this signal is also used for event generation):

- If the INSYNC bit is set for the input, the input is synchronized to the SCT clock using three SCT clock cycles resulting in a stable signal that is delayed by three SCT clock cycles.
- If the INSYNC bit is not set, the SIN bit value is the same as the AIN bit value.

Table 211. SCT input register (INPUT, address 0x1C01 8048) bit description

Bit	Symbol	Description	Reset value
0	AIN0	Input 0 state.Direct read.	-
1	AIN1	Input 1 state. Direct read.	-
2	AIN2	Input 2 state. Direct read.	-
3	AIN3	Input 3 state. Direct read.	-
4	AIN4	Input 4 state. Direct read.	-
5	AIN5	Input 5 state. Direct read.	-
6	AIN6	Input 6 state. Direct read.	-
7	AIN7	Input 7 state.Direct read.	-
15:8	-	Reserved.	-
16	SIN0	Input 0 state.	-
17	SIN1	Input 1 state.	-
18	SIN2	Input 2 state.	-
19	SIN3	Input 3 state.	-
20	SIN4	Input 4 state.	-
21	SIN5	Input 5 state.	-
22	SIN6	Input 6 state.	-
23	SIN7	Input 7 state.	-
31:24	-	Reserved	-

15.6.11 SCT match/capture registers mode register

If UNIFY = 1 in the CONFIG register, only the _L bits of this register are used. The L bits control whether each set of match/capture registers operates as unified 32-bit capture/match registers.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers REGMODE_L and REGMODE_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation. The _L bits/registers control the L match/capture registers, and the _H bits/registers control the H match/capture registers.

The SCT contains 16 Match/Capture register pairs. The Register Mode register selects whether each register pair acts as a Match register (see Section 15.6.20) or as a Capture register (see Section 15.6.22). Each Match/Capture register has an accompanying register which serves as a Reload register when the register is used as a Match register (Section 15.6.23) or as a Capture-Control register when the register is used as a capture register (Section 15.6.25). REGMODE_H is used only when the UNIFY bit is 0.

Table 212. SCT match/capture registers mode register (REGMODE, address 0x1C01 804C) bit description

Bit	Symbol	Description	Reset value
15:0	REGMOD_L	Each bit controls one pair of match/capture registers (register 0 = bit 0, register 1 = bit 1,..., register 15 = bit 15). 0 = registers operate as match registers. 1 = registers operate as capture registers.	0
31:16	REGMOD_H	Each bit controls one pair of match/capture registers (register 0 = bit 16, register 1 = bit 17,..., register 15 = bit 31). 0 = registers operate as match registers. 1 = registers operate as capture registers.	0

15.6.12 SCT output register

The SCT supports 10 outputs, each of which has a corresponding bit in this register.

Software can write to any of the output registers when both counters are halted to control the outputs directly. Writing to the OUT register is only allowed when all counters (L-counter, H-counter, and unified counter) are halted (HALT bits are set to 1 in the CTRL register).

Software can read this register at any time to sense the state of the outputs.

Table 213. SCT output register (OUTPUT, address 0x1C01 8050) bit description

Bit	Symbol	Description	Reset value
9:0	OUT	Writing a 1 to bit n makes the corresponding output HIGH. 0 makes the corresponding output LOW (output 0 = bit 0, output 1 = bit 1,..., output 9 = bit 9).	0
31:10	-	Reserved	

15.6.13 SCT bidirectional output control register

This register specifies (for each output) the impact of the counting direction on the meaning of set and clear operations on the output (see Section 15.6.28 and Section 15.6.29).

Table 214. SCT bidirectional output control register (OUTPUTDIRCTRL, address 0x1C01 8054) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SETCLR0		Set/clear operation on output 0. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
3:2	SETCLR1		Set/clear operation on output 1. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	

Table 214. SCT bidirectional output control register (OUTPUTDIRCTRL, address 0x1C01 8054) bit description

Bit	Symbol	Value	Description	Reset value
5:4	SETCLR2		Set/clear operation on output 2. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
7:6	SETCLR3		Set/clear operation on output 3. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
9:8	SETCLR4		Set/clear operation on output 4. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
11:10	SETCLR5		Set/clear operation on output 5. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
13:12	SETCLR6		Set/clear operation on output 6. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
15:14	SETCLR7		Set/clear operation on output 7. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
17:16	SETCLR8		Set/clear operation on output 8. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
19:18	SETCLR9		Set/clear operation on output 9. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
31:20	-		Reserved.	

15.6.14 SCT conflict resolution register

The registers OUTn_SET (Section 15.6.28) and OUTn_CLR (Section 15.6.29) allow both setting and clearing to be indicated for an output in the same clock cycle, even for the same event. This SCT conflict resolution register resolves this conflict.

To enable an event to toggle an output, set the OnRES value to 0x3 in this register, and set the event bits in both the Set and Clear registers.

Table 215. SCT conflict resolution register (RES, address 0x1C01 8058) bit description

Bit	Symbol	Value	Description	Reset value
1:0	O0RES		Effect of simultaneous set and clear on output 0.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR0 field).	
		0x2	Clear output (or set based on the SETCLR0 field).	
		0x3	Toggle output.	
3:2	O1RES		Effect of simultaneous set and clear on output 1.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR1 field).	
		0x2	Clear output (or set based on the SETCLR1 field).	
		0x3	Toggle output.	
5:4	O2RES		Effect of simultaneous set and clear on output 2.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR2 field).	
		0x2	Clear output n (or set based on the SETCLR2 field).	
		0x3	Toggle output.	
7:6	O3RES		Effect of simultaneous set and clear on output 3.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR3 field).	
		0x2	Clear output (or set based on the SETCLR3 field).	
		0x3	Toggle output.	
9:8	O4RES		Effect of simultaneous set and clear on output 4.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR4 field).	
		0x2	Clear output (or set based on the SETCLR4 field).	
		0x3	Toggle output.	
11:10	O5RES		Effect of simultaneous set and clear on output 5.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR5 field).	
		0x2	Clear output (or set based on the SETCLR5 field).	
		0x3	Toggle output.	
13:12	O6RES		Effect of simultaneous set and clear on output 6.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR6 field).	
		0x2	Clear output (or set based on the SETCLR6 field).	
		0x3	Toggle output.	

Table 215. SCT conflict resolution register (RES, address 0x1C01 8058) bit description

Bit	Symbol	Value	Description	Reset value
15:14	O7RES		Effect of simultaneous set and clear on output 7.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR7 field).	
		0x2	Clear output (or set based on the SETCLR7 field).	
		0x3	Toggle output.	
17:16	O8RES		Effect of simultaneous set and clear on output 8.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR8 field).	
		0x2	Clear output (or set based on the SETCLR8 field).	
		0x3	Toggle output.	
19:18	O9RES		Effect of simultaneous set and clear on output 9.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR9 field).	
		0x2	Clear output (or set based on the SETCLR9 field).	
		0x3	Toggle output.	
31:20	-		Reserved.	

15.6.15 SCT DMA request 0 and 1 registers

The SCT includes two DMA request outputs. These registers enable the DMA requests to be triggered when a particular event occurs or when counter Match registers are loaded from its Reload registers.

Table 216. SCT DMA 0 request register (DMAREQ0, address 0x1C01 805C) bit description

Bit	Symbol	Description	Reset value
15:0	DEV_0	If bit n is one, event n sets DMA request 0 (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
29:16	-	Reserved	-
30	DRL0	A 1 in this bit makes the SCT set DMA request 0 when it loads the Match_L/Unified registers from the Reload_L/Unified registers.	
31	DRQ0	This read-only bit indicates the state of DMA Request 0	

Table 217. SCT DMA 1 request register (DMAREQ1, address 0x1C01 8060) bit description

Bit	Symbol	Description	Reset value
15:0	DEV_1	If bit n is one, event n sets DMA request 1 (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0

Table 217. SCT DMA 1 request register (DMAREQ1, address 0x1C01 8060) bit description

Bit	Symbol	Description	Reset value
29:16	-	Reserved	-
30	DRL1	A 1 in this bit makes the SCT set DMA request 1 when it loads the Match L/Unified registers from the Reload L/Unified registers.	
31	DRQ1	This read-only bit indicates the state of DMA Request 1.	

15.6.16 SCT flag enable register

This register enables flags to request an interrupt if the FLAGn bit in the SCT event flag register (Section 15.6.17) is also set.

Table 218. SCT flag enable register (EVEN, address 0x1C01 80F0) bit description

Bit	Symbol	Description	Reset value
15:0	IEN	The SCT requests interrupt when bit n of this register and the event flag register are both one (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
31:16	-	Reserved	

15.6.17 SCT event flag register

This register records events. Writing ones to this register clears the corresponding flags and negates the SCT interrupt request if all enabled Flag bits are zero.

Table 219. SCT event flag register (EVFLAG, address 0x1C01 80F4) bit description

Bit	Symbol	Description	Reset value
15:0	FLAG	Bit n is one if event n has occurred since reset or a 1 was last written to this bit (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
31:16	-	Reserved	-

15.6.18 SCT conflict enable register

This register enables the “no change conflict” events specified in the SCT conflict resolution register to request an interrupt.

Table 220. SCT conflict enable register (CONEN, address 0x1C01 80F8) bit description

Bit	Symbol	Description	Reset value
9:0	NCEN	The SCT requests interrupt when bit n of this register and the SCT conflict flag register are both one (output 0 = bit 0, output 1 = bit 1,..., output 9 = bit 9).	0
31:10	-	Reserved	

15.6.19 SCT conflict flag register

This register records interrupt-enabled no-change conflict events and provides details of a bus error. Writing ones to the NCFLAG bits clears the corresponding read bits and negates the SCT interrupt request if all enabled Flag bits are zero.

Table 221. SCT conflict flag register (CONFLAG, address 0x1C01 80FC) bit description

Bit	Symbol	Description	Reset value
9:0	NCFLAG	Bit n is one if a no-change conflict event occurred on output n since reset or a 1 was last written to this bit (output 0 = bit 0, output 1 = bit 1,..., output 9 = bit 9).	0
29:10	-	Reserved.	-
30	BUSERRL	The most recent bus error from this SCT involved writing CTR L/Unified, STATE L/Unified, MATCH L/Unified, or the Output register when the L/U counter was not halted. A word write to certain L and H registers can be half successful and half unsuccessful.	0
31	BUSERRH	The most recent bus error from this SCT involved writing CTR H, STATE H, MATCH H, or the Output register when the H counter was not halted.	0

15.6.20 SCT match registers 0 to 15 (REGMODEn bit = 0)

Match registers are compared to the counters to help create events. When the UNIFY bit is 0, the L and H registers are independently compared to the L and H counters. When UNIFY is 1, the L and H registers hold a 32-bit value that is compared to the unified counter. A Match can only occur in a clock in which the counter is running (STOP and HALT are both 0).

Match registers can be read at any time. Writing to the MATCH_L, MATCH_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register). Match events occur in the SCT clock in which the counter is (or would be) incremented to the next value. When a Match event limits its counter as described in Section 15.6.3, the value in the Match register is the last value of the counter before it is cleared to zero (or decremented if BIDIR is 1).

There is no “write-through” from Reload registers to Match registers. Before starting a counter, software can write one value to the Match register used in the first cycle of the counter and a different value to the corresponding Match Reload register used in the second cycle.

Table 222. SCT match registers 0 to 15 (MATCH[0:15], address 0x1C01 8100 (MATCH0) to 0x1C01 813C (MATCH15)) bit description (REGMODEn bit = 0)

Bit	Symbol	Description	Reset value
15:0	MATCHn_L	When UNIFY = 0, read or write the 16-bit value to be compared to the L counter. When UNIFY = 1, read or write the lower 16 bits of the 32-bit value to be compared to the unified counter.	0
31:16	MATCHn_H	When UNIFY = 0, read or write the 16-bit value to be compared to the H counter. When UNIFY = 1, read or write the upper 16 bits of the 32-bit value to be compared to the unified counter.	0

15.6.21 SCT fractional match registers 0 to 5

Fractional Match registers are provided for up to the first six of the match registers. The values programmed in these registers provide higher average resolution over time by applying a dither pattern as described in Section 15.7.9.1. This dither pattern results in delaying recognition of a match for one counter clock for n (0 to 15) out of every 16 counter cycles. The value of n is programmed in these Fractional Match registers.

Fractional Match registers can be read at any time. Writing to the FRACMAT_L, FRACMAT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Each Fractional Match register has a Fractional Match Reload register associated with it. The contents of the reload registers are transferred into the Fractional Match registers at the start of every new SCT counter cycle unless the NORELOAD bit for the appropriate half-counter is set.

The reload registers may be written to at any time, regardless of whether or not the counter is running.

There is no write-through from the Fractional Match Reload registers to the Fractional Match registers. Before starting a counter, software can write one value to the Fractional Match register that will be used in the first cycle or period of operation, and a different value to the corresponding Fractional Match Reload register that will be used in the second cycle or period.

Table 223. SCT fractional match registers 0 to 5 (FRACMAT[0:5], address 0x1C01 8140 (FRACMAT0) to 0x1C01 8154 (FRACMAT5)) bit description

Bit	Symbol	Description	Reset value
3:0	FRACMAT_L	When UNIFY = 0, read or write the 4-bit value specifying the dither pattern to be applied to the corresponding MATCHn_L register (n = 0 to 5). When UNIFY = 1, the value applies to the unified, 32-bit fractional match register.	0
15:4	-	Reserved.	-
19:16	FRACMAT_H	When UNIFY = 0, read or write 4-bit value specifying the dither pattern to be applied to the corresponding MATCHn_H register (n = 0 to 5).	0
31:20	-	Reserved.	-

[1] See Section 15.7.9.1 for selecting the dither pattern.

15.6.22 SCT capture registers 0 to 15 (REGMODEn bit = 1)

These registers allow software to read the counter values at which the event selected by the corresponding Capture Control registers occurred.

Table 224. SCT capture registers 0 to 15 (CAP[0:15], address 0x1C01 8100 (CAP0) to 0x1C01 813C (CAP15)) bit description (REGMODEn bit = 1)

Bit	Symbol	Description	Reset value
15:0	CAPn_L	When UNIFY = 0, read the 16-bit counter value at which this register was last captured. When UNIFY = 1, read the lower 16 bits of the 32-bit value at which this register was last captured.	0
31:16	CAPn_H	When UNIFY = 0, read the 16-bit counter value at which this register was last captured. When UNIFY = 1, read the upper 16 bits of the 32-bit value at which this register was last captured.	0

15.6.23 SCT match reload registers 0 to 15 (REGMODEn bit = 0)

A Match register (L, H, or unified 32-bit) is loaded from the corresponding Reload register when BIDIR is 0 and the counter reaches its limit condition, or when BIDIR is 1 and the counter reaches 0.

Table 225. SCT match reload registers 0 to 15 (MATCHREL[0:15], address 0x1C01 8200 (MATCHREL0) to 0x1C01 823C (MATCHREL15) bit description (REGMODEn bit = 0)

Bit	Symbol	Description	Reset value
15:0	RELOADn_L	When UNIFY = 0, read or write the 16-bit value to be loaded into the MATCHn_L register. When UNIFY = 1, read or write the lower 16 bits of the 32-bit value to be loaded into the MATCHn register.	0
31:16	RELOADn_H	When UNIFY = 0, read or write the 16-bit to be loaded into the MATCHn_H register. When UNIFY = 1, read or write the upper 16 bits of the 32-bit value to be loaded into the MATCHn register.	0

15.6.24 SCT fractional match reload registers 0 to 5

A Fractional Match register (L, H, or unified 32-bit) is loaded from the corresponding Fractional Match Reload register when BIDIR is 0 and the counter reaches its limit condition, or BIDIR is 1 and the counter reaches 0, unless the appropriate NORELOAD bit is set.

Table 226. SCT fractional match reload registers 0 to 5 (FRACMATREL[0:5], address 0x1C01 8240 (FRACMATREL0) to 0x1C01 8254 (FRACMATREL5) bit description

Bit	Symbol	Description	Reset value
3:0	RELFRAC_L	When UNIFY = 0, read or write the 4-bit value to be loaded into the FRACMATn_L register. When UNIFY = 1, read or write the lower 4 bits to be loaded into the FRACMATn register.	0
15:4	-	Reserved.	-
19:16	RELFRAC_H	When UNIFY = 0, read or write the 4-bit value to be loaded into the FRACMATn_H register. When UNIFY = 1, read or write the upper 4 bits with the 4-bit value to be loaded into the FRACMATn register.	0
31:20	-	Reserved.	-

15.6.25 SCT capture control registers 0 to 15 (REGMODEn bit = 1)

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers CAPCTRLn_L and CAPCTRLn_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Each Capture Control register (L, H, or unified 32-bit) controls which events load the corresponding Capture register from the counter.

Table 227. SCT capture control registers 0 to 15 (CAPCTRL[0:15], address 0x1C01 8200 (CAPCTRL0) to 0x1C01 823C (CAPCTRL15)) bit description (REGMODEn bit = 1)

Bit	Symbol	Description	Reset value
15:0	CAPCONn_L	If bit m is one, event m causes the CAPn_L (UNIFY = 0) or the CAPn (UNIFY = 1) register to be loaded (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
31:16	CAPCONn_H	If bit m is one, event m causes the CAPn_H (UNIFY = 0) register to be loaded (event 0 = bit 16, event 1 = bit 17,..., event 15 = bit 31).	0

15.6.26 SCT event state mask registers 0 to 15

Each event has one associated SCT event state mask register that allows this event to happen in one or more states of the counter selected by the HEVENT bit in the corresponding EVCTRLn register.

An event n is disabled when its EVn_STATE register contains all zeros, since it is masked regardless of the current state.

In simple applications that do not use states, write 0x01 to this register to enable an event. Since the state always remains at its reset value of 0, writing 0x01 effectively permanently state-enables this event.

Table 228. SCT event state mask registers 0 to 15 (EV[0:15]_STATE, addresses 0x1C01 8300 (EV0_STATE) to 0x1C01 8378 (EV15_STATE) (SCT0) and 0x1C01 C300 (EV0_STATE) to 0x1C01 C378 (EV15_STATE) (SCT1)) bit description

Bit	Symbol	Description	Reset value
15:0	STATEMSKn	If bit m is one, event n (n= 0 to 15) happens in state m of the counter selected by the HEVENT bit (m = state number; state 0 = bit 0, state 1= bit 1,..., state 15 = bit 15).	0
31:16	-	Reserved.	-

15.6.27 SCT event control registers 0 to 15

This register defines the conditions for event n to occur, other than the state variable which is defined by the state mask register. Most events are associated with a particular counter (high, low, or unified), in which case the event can depend on a match to that register. The other possible ingredient of an event is a selected input or output signal.

When the UNIFY bit is 0, each event is associated with a particular counter by the HEVENT bit in its event control register. An event cannot occur when its related counter is halted nor when the current state is not enabled to cause the event as specified in its event mask register. An event is permanently disabled when its event state mask register contains all zeros.

An enabled event can be programmed to occur based on a selected input or output edge or level and/or based on its counter value matching a selected match register. In BIDR mode, events can also be enabled based on the count direction.

Each event can modify its counter STATE value. If more than one event associated with the same counter occurs in a given clock cycle, only the state change specified for the highest-numbered event among them takes place. Other actions dictated by any simultaneously occurring events all take place.

Table 229. SCT event control register 0 to 15 (EV[0:15]_CTRL, address 0x1C01 8304 (EV0_CTRL) to 0x1C01 837C (EV15_CTRL) (SCT0) and EV[0:15]_CTRL, address 0x1C01 C304 (EV0_CTRL) to 0x1C01 C37C (EV15_CTRL) (SCT1)) bit description

Bit	Symbol	Value	Description	Reset value
3:0	MATCHSEL	-	Selects the Match register associated with this event (if any). A match can occur only when the counter selected by the HEVENT bit is running.	0
4	HEVENT		Select L/H counter. Do not set this bit if UNIFY = 1.	0
		0	Selects the L state and the L match register selected by MATCHSEL.	
		1	Selects the H state and the H match register selected by MATCHSEL.	
5	OUTSEL		Input/output select	0
		0	Selects the input selected by IOSEL.	
		1	Selects the output selected by IOSEL.	
9:6	IOSEL	-	Selects the input or output signal number associated with this event (if any). Do not select an input in this register, if CKMODE is 1x. In this case the clock input is an implicit ingredient of every event.	0
11:10	IOCOND		Selects the I/O condition for event n. (The detection of edges on outputs lags the conditions that switch the outputs by one SCT clock). In order to guarantee proper edge/state detection, an input must have a minimum pulse width of at least one SCT clock period .	0
		0x0	LOW	
		0x1	Rise	
		0x2	Fall	
		0x3	HIGH	
13:12	COMBMODE		Selects how the specified match and I/O condition are used and combined.	0
		0x0	OR. The event occurs when either the specified match or I/O condition occurs.	
		0x1	MATCH. Uses the specified match only.	
		0x2	IO. Uses the specified I/O condition only.	
		0x3	AND. The event occurs when the specified match and I/O condition occur simultaneously.	
14	STATELD		This bit controls how the STATEV value modifies the state selected by HEVENT when this event is the highest-numbered event occurring for that state.	0
		0	STATEV value is added into STATE (the carry-out is ignored).	
		1	STATEV value is loaded into STATE.	
19:15	STATEV		This value is loaded into or added to the state selected by HEVENT, depending on STATELD, when this event is the highest-numbered event occurring for that state. If STATELD and STATEV are both zero, there is no change to the STATE value.	0

Table 229. SCT event control register 0 to 15 (EV[0:15]_CTRL, address 0x1C01 8304 (EV0_CTRL) to 0x1C01 837C (EV15_CTRL) (SCT0) and EV[0:15]_CTRL, address 0x1C01 C304 (EV0_CTRL) to 0x1C01 C37C (EV15_CTRL) (SCT1)) bit description

Bit	Symbol	Value	Description	Reset value
20	MATCHMEM		If this bit is one and the COMBMODE field specifies a match component to the triggering of this event, then a match is considered to be active whenever the counter value is GREATER THAN OR EQUAL TO the value specified in the match register when counting up, LESS THEN OR EQUAL TO the match value when counting down. If this bit is zero, a match is only be active during the cycle when the counter is equal to the match value.	
22:21	DIRECTION		Direction qualifier for event generation. This field only applies when the counters are operating in BIDIR mode. If BIDIR = 0, the SCT ignores this field. Value 0x3 is reserved.	
		0x0	Direction independent. This event is triggered regardless of the count direction.	
		0x1	Counting up. This event is triggered only during up-counting when BIDIR = 1.	
		0x2	Counting down. This event is triggered only during down-counting when BIDIR = 1.	
31:23	-		Reserved	

15.6.28 SCT output set registers 0 to 9

Each output n has one set register that controls how events affect each output. Whether outputs are set or cleared depends on the setting of the SETCLRn field in the OUTPUTDIRCTRL register.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 230. SCT output set register 0 to 9 (OUT[0:9]_SET, address 0x1C01 8500 (OUT0_SET) to 0x1C01 8548 (OUT9_SET) (SCT0) and 0x1C01 C500 (OUT0_SET) to 0x1C01 C548 (OUT9_SET) (SCT1)) bit description

Bit	Symbol	Description	Reset value
15:0	SET	A 1 in bit m selects event m to set output n (or clear it if SETCLRn = 0x1 or 0x2) event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15.	0
31:16	-	Reserved	

15.6.29 SCT output clear registers 0 to 9

Each output n has one clear register that controls how events affect each output. Whether outputs are set or cleared depends on the setting of the SETCLRn field in the OUTPUTDIRCTRL register.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 231. SCT output clear register 0 to 9 (OUT[0:9]_CLR, address 0x1C01 8504 (OUT0_CLR) to 0x1C01 854C (OUT9_CLR) (SCT0) and OUT[0:9]_CLR and 0x1C01 8504 (OUT0_CLR) to 0x1C01 854C (OUT9_CLR) (SCT1)) bit description

Bit	Symbol	Description	Reset value
15:0	CLR	A 1 in bit m selects event m to clear output n (or set it if SETCLRn = 0x1 or 0x2) event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15.	0
31:16	-	Reserved	

15.7 Functional description

15.7.1 Match logic

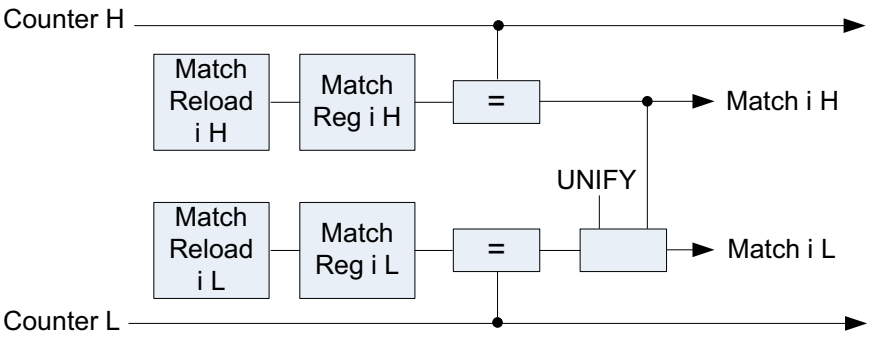


Fig 34. Match logic

15.7.2 Capture logic

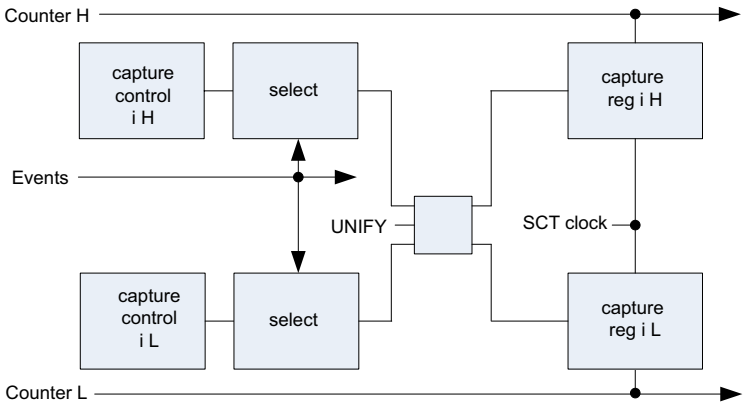


Fig 35. Capture logic

15.7.3 Event selection

State variables allow control of the SCT across more than one cycle of the counter. Counter matches, input/output edges, and state values are combined into a set of general-purpose events that can switch outputs, request interrupts, and change state values.

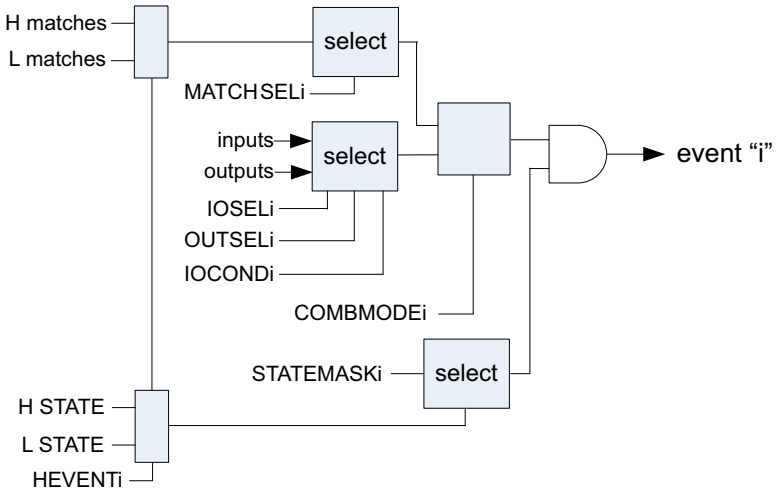


Fig 36. Event selection

15.7.4 Output generation

Figure 37 shows one output slice of the SCT.

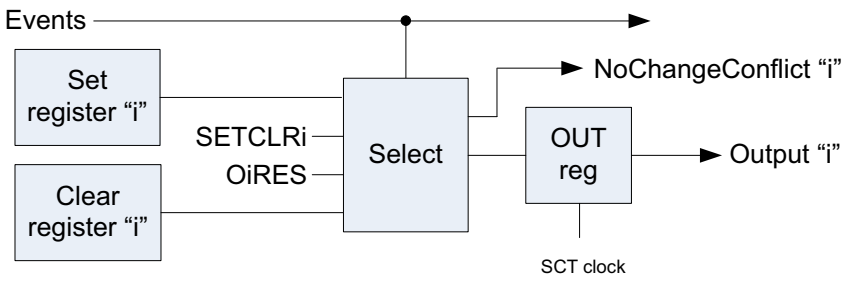


Fig 37. Output slice i

15.7.5 State logic

The SCT can be configured as a timer/counter with multiple programmable states. The states are user-defined through the events that can be captured in each particular state. In a multi-state SCT. the SCT can change from one state to another state when a user-defined event triggers a state change. The state change is set up through each event’s EV_CTRL register in one of the following ways:

- The event can increment the current state number by a new value.
- The event can write a new state value.

If an event increments the state number beyond the number of available states, the SCT enters a locked state in which all further events are ignored while the counter is still running. Software must interfere to change out of this state.

Software can capture the counter value (and potentially create an interrupt and write to all outputs) when the event moving the SCT into a locked state occurs. Later, while the SCT is in the locked state, software can read the counter again to record the time passed since the locking event and can also read the state variable to obtain the current state number

If the SCT registers an event that forces an abort, putting the SCT in a locked state can be a safe way to record the time that has passed since the abort event while no new events are allowed to occur. Since multiple states (any state number between the maximum implemented state and 31) are locked states, multiple abort or error events can be defined each incrementing the state number by a different value. Also see Section 15.3.5.1 “Abort function”.

15.7.6 Interrupt generation

The SCT generates one interrupt to the NVIC.

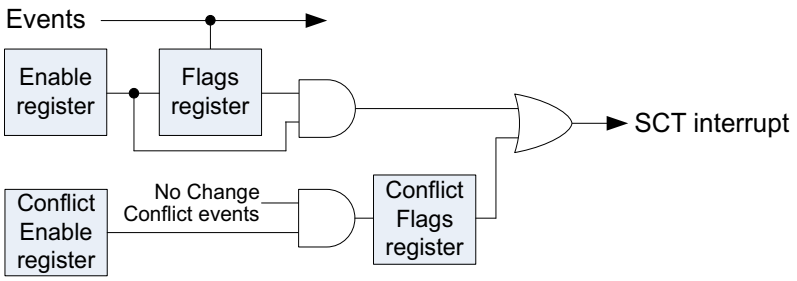


Fig 38. SCT interrupt generation

15.7.7 Clearing the prescaler

When enabled by a non-zero PRE field in the Control register, the prescaler acts as a clock divider for the counter, like a fractional part of the counter value. The prescaler is cleared whenever the counter is cleared or loaded for any of the following reasons:

- Hardware reset
- Software writing to the counter register
- Software writing a 1 to the CLRCTR bit in the control register
- an event selected by a 1 in the counter limit register when BIDIR = 0

When BIDIR is 0, a limit event caused by an I/O signal can clear a non-zero prescaler. However, a limit event caused by a Match only clears a non-zero prescaler in one special case as described Section 15.7.8.

A limit event when BIDIR is 1 does not clear the prescaler. Rather it clears the DOWN bit in the Control register, and decrements the counter on the same clock if the counter is enabled in that clock.

15.7.8 Match vs. I/O events

Counter operation is complicated by the prescaler and by clock mode 01 in which the SCT clock is the bus clock. However, the prescaler and counter are enabled to count only when a selected edge is detected on a clock input.

- The prescaler is enabled when the clock mode is not 01, or when the input edge selected by the CLKSEL field is detected.
- The counter is enabled when the prescaler is enabled, and (PRELIM=0 or the prescaler is equal to the value in PRELIM).

An I/O component of an event can occur in any SCT clock when its counter HALT bit is 0. In general, a Match component of an event can only occur in a UT clock when its counter HALT and STOP bits are both 0 and the counter is enabled.

Table 232 shows when the various kinds of events can occur.

Table 232. Event conditions

COMBMODE	IOMODE	Event can occur on clock:
IO	Any	Event can occur whenever HALT = 0 (type A).
MATCH	Any	Event can occur when HALT = 0 and STOP = 0 and the counter is enabled (type C).
OR	Any	From the IO component: Event can occur whenever HALT = 0 (A). From the match component: Event can occur when HALT = 0 and STOP = 0 and the counter is enabled (C).
AND	LOW or HIGH	Event can occur when HALT = 0 and STOP = 0 and the counter is enabled (C).
AND	RISE or FALL	Event can occur whenever HALT = 0 (A).

15.7.9 Fractional matches

The first 6 match registers may be configured to have a fractional portion to their match values. Higher average resolution is achievable on the match registers with associated fractional match register by using a dithering mechanism. The dither engine delays the assertion of a match by one counter clock every n (0 to 15) out of 16 counter cycles. The value of n is specified in the 4-bit FRACMAT register associated with each of the first six match registers.

Dithering can be disabled on any of the match registers by loading all zeroes (the default value) into its FRACMAT register.

15.7.9.1 Dithering

At the start of each new SCT counter cycle (i.e. when the counter counts-down to zero in bi-directional mode or is cleared to zero by a limit event), the dither engine determines which matches are to be delayed by one clock during the coming counter cycle. Delaying the match effectively adds 1 to the designated match value when up-counting or subtracts 1 when down-counting, during that particular counter cycle.

For each dither-enabled match register, the value programmed in its associated FRACMAT register specifies how many out of every 16 counter cycles its match is to be delayed. An algorithm applied to the FRACMAT value distributes this number as evenly as possible across the 16 counter cycles. This results in a unique dither pattern for each match register (SeeTable 233).

Additional control over the dithering process is provided to the user via a Dither Condition (event-mask) register. Typically, the dither engine advances though the match dither patterns at the start of every new SCT counter cycle. The Dither Condition register allows the user to specify that advancement to the next element in the dither patterns will only occur if one or more designated events occurred during the previous cycle of the counter.

The dither algorithm is designed to spread out the cycles in which the matches are delayed as evenly as possible across the 16 counter cycles. The following table shows the dither pattern that is applied for each value of FRACMAT. A 'D' indicates the counter cycles where a match on the relevant match register is delayed.

Table 233. Dither pattern

	Counter cycle															
FRACMAT	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0x0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0x1	-	-	-	-	-	-	-	-	D	-	-	-	-	-	-	-
0x2	-	-	-	-	D	-	-	-	-	-	-	-	D	-	-	-
0x3	-	-	-	-	D	-	-	-	D	-	-	-	D	-	-	-
0x4	-	-	D	-	-	-	D	-	-	-	D	-	-	-	D	-
0x5	-	-	D	-	-	-	D	-	D	-	D	-	-	-	D	-
0x6	-	-	D	-	D	-	D	-	-	-	D	-	D	-	D	-
0x7	-	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-
0x8	-	D	-	D	-	D	-	D	-	D	-	D	-	D	-	D
0x9	-	D	-	D	-	D	-	D	D	D	-	D	-	D	-	D
0xA	-	D	-	D	D	D	-	D	-	D	-	D	D	D	-	D
0xB	-	D	-	D	D	D	-	D	D	D	-	D	D	D	-	D
0xC	-	D	D	D	-	D	D	D	-	D	D	D	-	D	D	D
0xD	-	D	D	D	-	D	D	D	D	D	D	D	-	D	D	D
0xE	-	D	D	D	D	D	D	D	-	D	D	D	D	D	D	D
0xF	-	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

15.7.10 SCT operation

In its simplest, single-state configuration, the SCT operates as an event controlled one- or bidirectional counter. Events can be configured to be counter match events, an input or output level, transitions on an input or output pin, or a combination of match and input/output behavior. In response to an event, the SCT output or outputs can transition, or the SCT can perform other actions such as creating an interrupt or starting, stopping, or resetting the counter. Multiple simultaneous actions are allowed for each event. Furthermore, any number of events can trigger one specific action of the SCT.

An action or multiple actions of the SCT uniquely define an event. A state is defined by which events are enabled to trigger an SCT action or actions in any stage of the counter. Events not selected for this state are ignored.

In a multi-state configuration, states change in response to events. A state change is an additional action that the SCT can perform when the event occurs. When an event is configured to change the state, the new state defines a new set of events resulting in different actions of the SCT. Through multiple cycles of the counter, events can change the state multiple times and thus create a large variety of event controlled transitions on the SCT outputs and/or interrupts.

Once configured, the SCT can run continuously without software intervention and can generate multiple output patterns entirely under the control of events.

- To configure the SCT, see Section 15.7.11.
- To start, run, and stop the SCT, see Section 15.7.12.
- To configure the SCT as simple event controlled counter/timer, see Section 15.7.13.

15.7.11 Configure the SCT

To set up the SCT for multiple events and states, perform the following configuration steps:

15.7.11.1 Configure the counter

1. Configure the L and H counters in the CONFIG register by selecting two independent 16-bit counters (L counter and H counter) or one combined 32-bit counter in the UNIFY field.
2. Select the SCT clock source in the CONFIG register (fields CLKMODE and CLKSEL) from any of the inputs or an internal clock.

15.7.11.2 Configure the match and capture registers

1. Select how many match and capture registers the application uses (total of up to 5):
   - In the REGMODE register, select for each of the 5 match/capture register pairs whether the register is used as a match register or capture register.
2. Define match conditions for each match register selected:
   - Each match register MATCH sets one match value, if a 32-bit counter is used, or two match values, if the L and H 16-bit counters are used.
   - Each match reload register MATCHRELOAD sets a reload value that is loaded into the match register when the counter reaches a limit condition or the value 0.

15.7.11.3 Configure events and event responses

1. Define when each event can occur in the following way in the EVn_CTRL registers (up to 6, one register per event):
   - Select whether the event occurs on an input or output changing, on an input or output level, a match condition of the counter, or a combination of match and input/output conditions in field COMBMODE.
   - For a match condition:

     Select the match register that contains the match condition for the event to occur. Enter the number of the selected match register in field MATCHSEL.

     If using L and H counters, define whether the event occurs on matching the L or the H counter in field HEVENT.
   - For an SCT input or output level or transition:

     Select the input number or the output number that is associated with this event in fields IOSEL and OUTSEL.

     Define how the selected input or output triggers the event (edge or level sensitive) in field IOCOND.
2. Define what the effect of each event is on the SCT outputs in the OUTn_SET or OUTn_CLR registers (up to 4 outputs, one register per output):
   - For each SCT output, select which events set or clear this output. More than one event can change the output, and each event can change multiple outputs.
3. Define how each event affects the counter:
   - Set the corresponding event bit in the LIMIT register for the event to set an upper limit for the counter.

     When a limit event occurs in unidirectional mode, the counter is cleared to zero and begins counting up on the next clock edge.

     When a limit event occurs in bidirectional mode, the counter begins to count down from the current value on the next clock edge.
   - Set the corresponding event bit in the HALT register for the event to halt the counter. If the counter is halted, it stops counting and no new events can occur. The counter operation can only be restored by clearing the HALT_L and/or the HALT_H bits in the CTRL register.
   - Set the corresponding event bit in the STOP register for the event to stop the counter. If the counter is stopped, it stops counting. However, an event that is configured as a transition on an input/output can restart the counter.
   - Set the corresponding event bit in the START register for the event to restart the counting. Only events that are defined by an input changing can be used to restart the counter.
4. Define which events contribute to the SCT interrupt:
   - Set the corresponding event bit in the EVEN and the EVFLAG registers to enable the event to contribute to the SCT interrupt.

15.7.11.4 Configure multiple states

1. In the EVn_STATE register for each event (up to 6 events, one register per event), select the state or states (up to 2) in which this event is allowed to occur. Each state can be selected for more than one event.
2. Determine how the event affects the system state:

   In the EVn_CTRL registers (up to 6 events, one register per event), set the new state value in the STATEV field for this event. If the event is the highest numbered in the current state, this value is either added to the existing state value or replaces the existing state value, depending on the field STATELD.

   Remark: If there are higher numbered events in the current state, this event cannot change the state.

   If the STATEV and STATELD values are set to zero, the state does not change.

15.7.11.5 Miscellaneous options

- There are a certain (selectable) number of capture registers. Each capture register can be programmed to capture the counter contents when one or more events occur.
- If the counter is in bidirectional mode, the effect of set and clear of an output can be made to depend on whether the counter is counting up or down by writing to the OUTPUTDIRCTRL register.

15.7.12 Run the SCT

1. Configure the SCT (see Section 15.7.11 “Configure the SCT”).
2. Write to the STATE register to define the initial state. By default the initial state is state 0.
3. To start the SCT, write to the CTRL register:
   - Clear the counters.
   - Clear or set the STOP_L and/or STOP_H bits.

     Remark: The counter starts counting once the STOP bit is cleared as well. If the STOP bit is set, the SCT waits instead for an event to occur that is configured to start the counter.
   - For each counter, select unidirectional or bidirectional counting mode (field BIDIR_L and/or BIDIR_H).
   - Select the prescale factor for the counter clock (CTRL register).
   - Clear the HALT_L and/or HALT_H bit. By default, the counters are halted and no events can occur.
4. To stop the counters by software at any time, stop or halt the counter (write to STOP_L and/or STOP_H bits or HALT_L and/or HALT_H bits in the CTRL register).
   - When the counters are stopped, both an event configured to clear the STOP bit or software writing a zero to the STOP bit can start the counter again.
   - When the counter are halted, only a software write to clear the HALT bit can start the counter again. No events can occur.
   - When the counters are halted, software can set any SCT output HIGH or LOW directly by writing to the OUT register.

The current state can be read at any time by reading the STATE register.

To change the current state by software (that is independently of any event occurring), set the HALT bit and write to the STATE register to change the state value. Writing to the STATE register is only allowed when the counter is halted (the HALT_L and/or HALT_H bits are set) and no events can occur.

15.7.13 Configure the SCT without using states

The SCT can be used as standard counter/timer with external capture inputs and match outputs without using the state logic. To operate the SCT without states, configure the SCT as follows:

- Write zero to the STATE register (zero is the default).
- Write zero to the STATELD and STATEV fields in the EVCTRL registers for each event.
- Write 0x1 to the EVn_STATE register of each event. Writing 0x1 enables the event.

  In effect, the event is allowed to occur in a single state which never changes while the counter is running.

15.7.14 SCT Example

Figure 39 shows a simple application of the SCT using two sets of match events (EV0/1 and EV3/4) to set/clear SCT output 0. The timer is automatically reset whenever it reaches the MAT0 match value.

In the initial state 0, match event EV0 sets output 0 to HIGH and match event EV1 clears output 0. The SCT input 0 is monitored: If input0 is found LOW by the next time the timer is reset (EV2), the state is changed to state 1, and EV3/4 are enabled, which create the same output but triggered by different match values. If input 0 is found HIGH by the next time the timer is reset, the associated event (EV5) causes the state to change back to state 0where the events EV0 and EV1 are enabled.

The example uses the following SCT configuration:

- 1 input
- 1 output
- 5 match registers
- 6 events and match 0 used with autolimit function
- 2 states

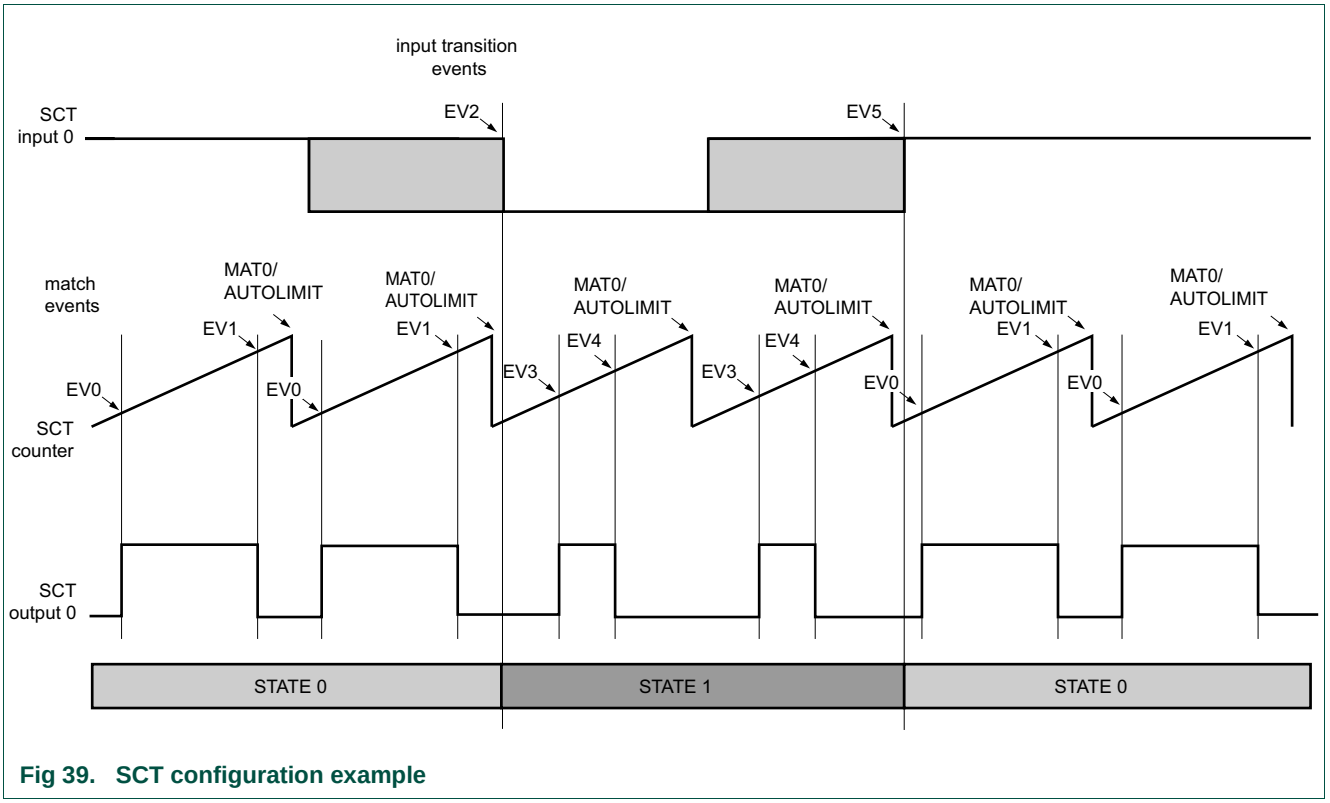


Fig 39. SCT configuration example

This application of the SCT uses the following configuration (all register values not listed in Table 234 are set to their default values):

Table 234. SCT configuration example

Configuration	Registers	Setting
Counter	CONFIG	Uses one counter (UNIFY = 1).
	CONFIG	Enable the autolimit for MAT0. (AUTOLIMIT = 1).
	CTRL	Uses unidirectional counter (BIDIR_L = 0).
Clock base	CONFIG	Uses default values for clock configuration.
Match/Capture registers	REGMODE	Configure one match register for each match event by setting REGMODE_L bits 0,1, 2, 3, 4 to 0. This is the default.
Define match values	MATCH0/1/2/3/4	Set a match value MATCH0/1/2/3/4_L in each register. The match 0 register serves as an automatic limit event that resets the counter without using an event. To enable the automatic limit, set the AUTOLIMIT bit in the CONFIG register.
Define match reload values	MATCHREL0/1/2/3/4	Set a match reload value RELOAD0/1/2/3/4_L in each register (same as the match value in this example).
Define when event 0 occurs	EV0_CTRL	• Set COMBMODE = 0x1. Event 0 uses match condition only. • Set MATCHSEL = 1. Select match value of match register 1. The match value of MAT1 is associated with event 0.
Define when event 1 occurs	EV1_CTRL	• Set COMBMODE = 0x1. Event 1 uses match condition only. • Set MATCHSEL = 2 Select match value of match register 2. The match value of MAT2 is associated with event 1.

Table 234. SCT configuration example

Configuration	Registers	Setting
Define when event 2 occurs	EV2_CTRL	• Set COMBMODE = 0x3. Event 2 uses match condition and I/O condition. • Set IOSEL = 0. Select input 0. • Set IOCOND = 0x0. Input 0 is LOW. • Set MATCHSEL = 0. Chooses match register 0 to qualify the event.
Define how event 2 changes the state	EV2_CTRL	Set STATEV bits to 1 and the STATED bit to 1. Event 2 changes the state to state 1.
Define when event 3 occurs	EV3_CTRL	• Set COMBMODE = 0x1. Event 3 uses match condition only. • Set MATCHSEL = 0x3. Select match value of match register 3. The match value of MAT3 is associated with event 3..
Define when event 4 occurs	EV4_CTRL	• Set COMBMODE = 0x1. Event 4 uses match condition only. • Set MATCHSEL = 0x4. Select match value of match register 4.The match value of MAT4 is associated with event 4.
Define when event 5 occurs	EV5_CTRL	• Set COMBMODE = 0x3. Event 5 uses match condition and I/O condition. • Set IOSEL = 0. Select input 0. • Set IOCOND = 0x3. Input 0 is HIGH. • Set MATCHSEL = 0. Chooses match register 0 to qualify the event.
Define how event 5 changes the state	EV5_CTRL	Set STATEV bits to 0 and the STATED bit to 1. Event 5 changes the state to state 0.
Define by which events output 0 is set	OUT0_SET	Set SET0 bits 0 (for event 0) and 3 (for event 3) to one to set the output when these events 0 and 3 occur.
Define by which events output 0 is cleared	OUT0_CLR	Set CLR0 bits 1 (for events 1) and 4 (for event 4) to one to clear the output when events 1 and 4 occur.
Configure states in which event 0 is enabled	EV0_STATE	Set STATEMSK0 bit 0 to 1. Set all other bits to 0. Event 0 is enabled in state 0.
Configure states in which event 1 is enabled	EV1_STATE	Set STATEMSK1 bit 0 to 1. Set all other bits to 0. Event 1 is enabled in state 0.
Configure states in which event 2 is enabled	EV2_STATE	Set STATEMSK2 bit 0 to 1. Set all other bits to 0. Event 2 is enabled in state 0.
Configure states in which event 3 is enabled	EV3_STATE	Set STATEMSK3 bit 1 to 1. Set all other bits to 0. Event 3 is enabled in state 1.
Configure states in which event 4 is enabled	EV4_STATE	Set STATEMSK4 bit 1 to 1. Set all other bits to 0. Event 4 is enabled in state 1.
Configure states in which event 5 is enabled	EV5_STATE	Set STATEMSK5 bit 1 to 1. Set all other bits to 0. Event 5 is enabled in state 1.

16.1 How to read this chapter

The small SCTs are available on all parts. The SCT2 and SCT3 outputs #5 cannot be connected to external pins on the LQFP48 and LQFP64 packages. The SCT3 output #4 cannot be connected to a pin on the LQFP48 package. (All outputs are available for internal connections.)

16.2 Features

The following feature list summarizes the configuration for the two small SCTs. Each small SCT has a companion large SCT (see Section 15.2) with more inputs and outputs and a dither engine.

- Each SCT supports:
  - 8 match/capture registers
  - 10 events
  - 10 states
  - 3 inputs and 6 outputs
  - DMA support
- Counter/timer features:
  - Configurable as two 16-bit counters or one 32-bit counter.
  - Counters clocked by bus clock or selected input.
  - Up counters or up-down counters.
  - Configurable number of match and capture registers. Up to 16 match and capture registers total.
  - Upon match create the following events: interrupt, stop, limit timer or change direction; toggle outputs; change state.
  - Counter value can be loaded into capture register triggered by match or input/output toggle.
- PWM features:
  - Counters can be used in conjunction with match registers to toggle outputs and create time-proportioned PWM signals.
  - Up to six single-edge or dual-edge controlled PWM outputs with independent duty cycles if configured as 32-bit timers.
- Event creation features:
  - The following conditions define an event: a counter match condition, an input (or output) condition, a combination of a match and/or and input/output condition in a specified state.
  - Selected events can limit, halt, start, or stop a counter.
  - Events control state changes, outputs, interrupts, and DMA requests.

  - Match register 0 can be used as an automatic limit.
  - In bi-directional mode, events can be enabled based on the count direction.
  - Match events can be held until another qualifying event occurs.
- State control features:
  - A state is defined by events that can take place in the state while the counter is running.
  - A state changes into another state as result of an event.
  - Each event can be assigned to one or more states.
  - State variable allows sequencing across multiple counter cycles.
- Integrated with an input pre-processing unit (SCTIPU) to combine or delay input events.

Inputs and outputs on the SCT2 and SCT3 are configured as follows:

- 3 inputs. Each input selects one of 21 sources from an input multiplexer.
- 6 outputs (some outputs are connected to multiple locations)
  - Three outputs connected to external pins through the switch matrix as movable functions.
  - Three outputs connected to external pins through the switch matrix as fixed-pin functions.
  - Two outputs connected to the SCT IPU to use as sample and abort inputs.
  - Four outputs connected to the accompanying large SCT
  - Two outputs connected to each ADC trigger input

16.3 Basic configuration

Configure the SCT2/3 as follows:

- Use the SYSAHBCLKCTRL1 register (Table 51) to enable the clock to the SCT register interface and peripheral clock.
- Clear the SCT2/3 peripheral resets using the PRESETCTRL1 register (Table 36).
- The SCT2/3 combined interrupts are connected to slot #18/19 in the NVIC.
- Use the switch matrix and SCT2/3_INMUX registers to connect the SCT inputs and outputs to external pins or internally (see Section 16.4).

  Remark: For applications that require exact timing of the SCT outputs (for example PWM), assign the outputs only to fixed-pin functions to ensure that the output skew is nearly the same for all outputs.
- The SCT DMA request lines 2 and 3 are connected to the DMA trigger inputs via the DMA_ITRIG_INMUX registers. See Table 132 “DMA input trigger Input mux registers 0 to 17 (DMA_ITRIG_INMUX[0:17], address 0x4001 40E0 (DMA_ITRIG_INMUX0) to 0x4001 4124 (DMA_ITRIG_INMUX17)) bit description”.

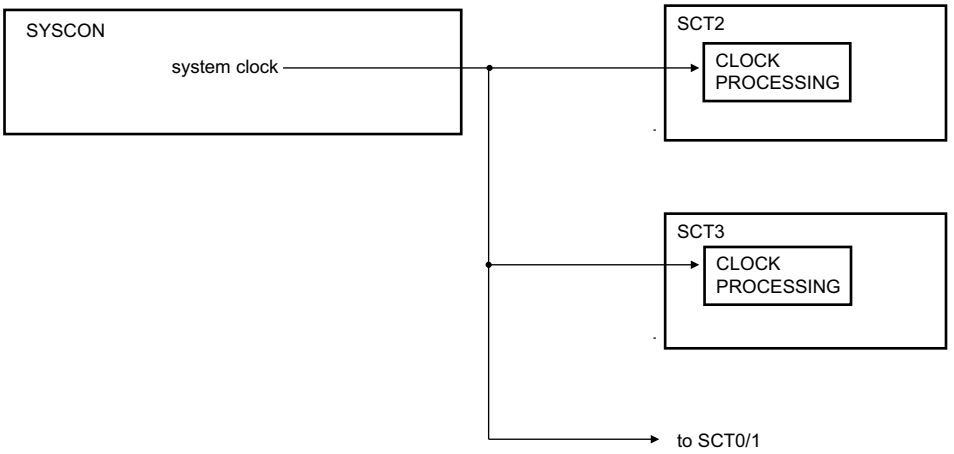


Fig 40. Small SCT clocking

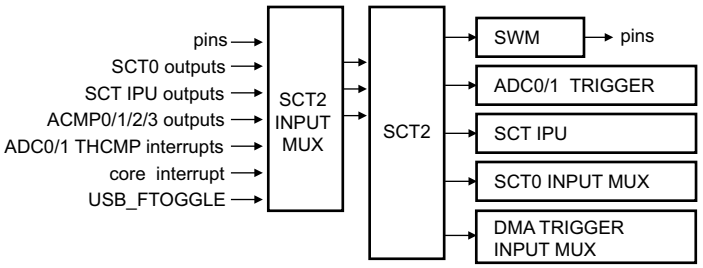


Fig 41. SCT2 connections

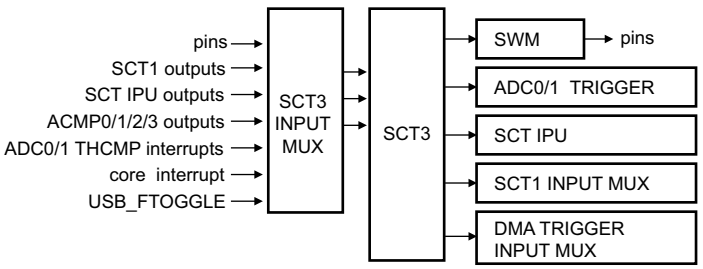


Fig 42. SCT3 connections

16.3.1 SCT inputs and outputs

Each of the SCT inputs has a pre-selector that can select one of 21 possible input signals (SCT input mux). The input signals are numbered 0 to 20 (see Figure 43), and the signal number is programmed in the SCT2_INMUXn or SCT3_INMUXn register for any particular input:

SCT2 inputs are selected through input muxes SCT2_INMUX[0:2]. See Table 129.

SCT3 inputs are selected through input muxes SCT3_INMUX[0:2]. See Table 130.

Each large SCT has 6 outputs. All outputs are connected to external pins through the switch matrix. Some outputs are routed to the other SCT blocks, to the ADC trigger inputs, and to the SCT IPU. An SCT output can be routed to multiple places.

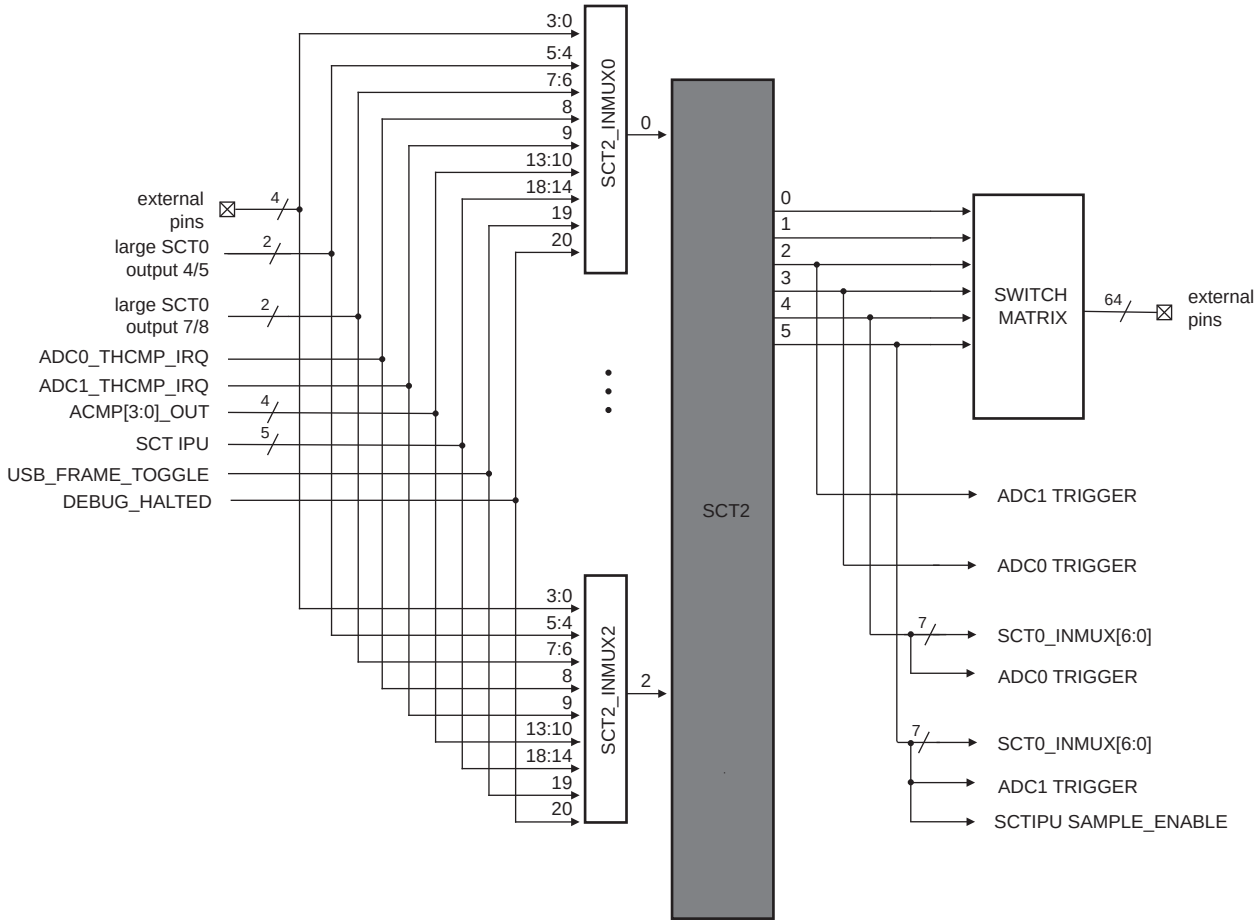


Fig 43. SCT2 inputs and outputs

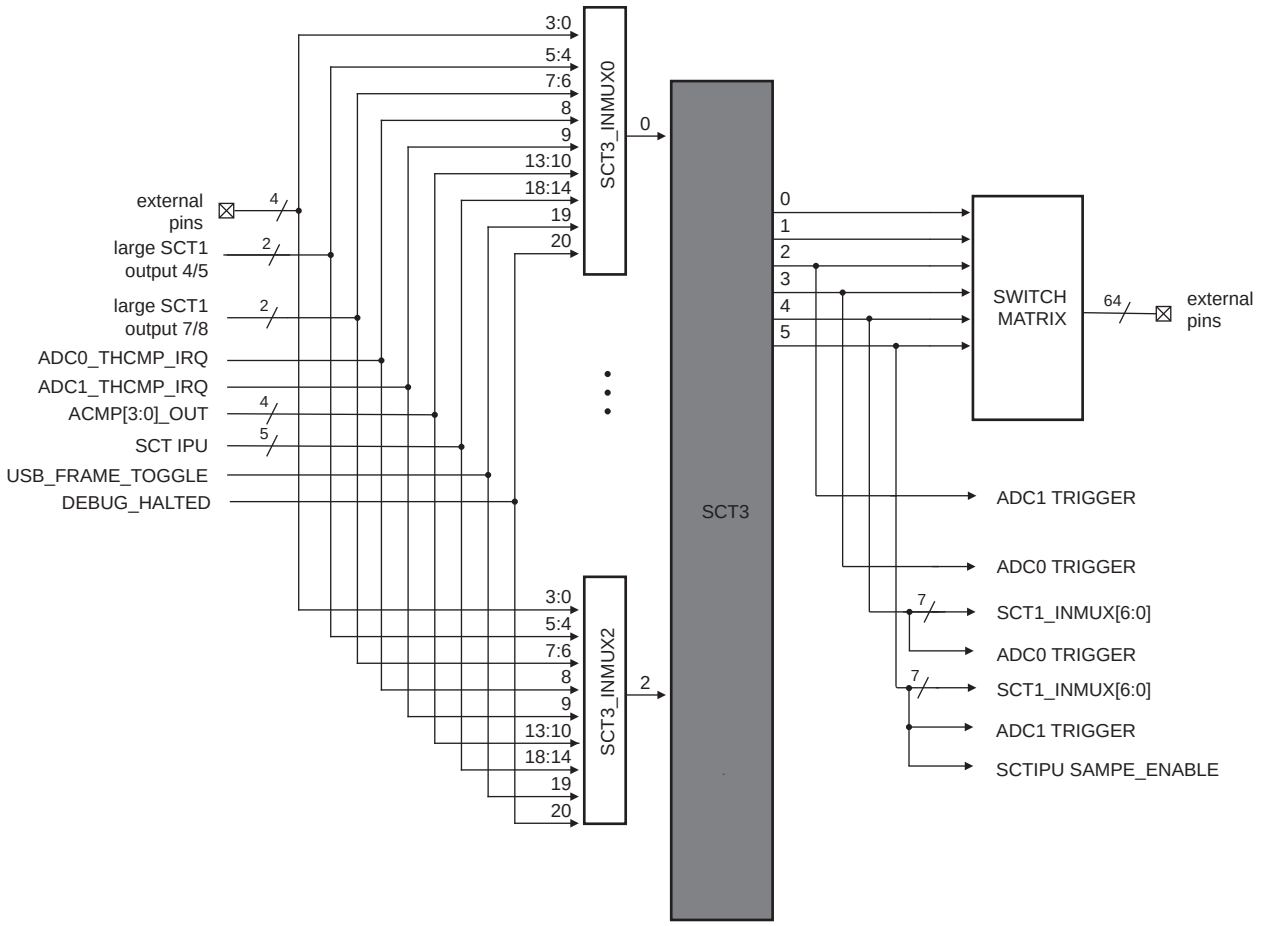


Fig 44. SCT3 inputs and outputs

16.3.2 Use the SCT with the SCTIPU

The SCTIPU provides a pre-processing unit for the SCT inputs for two purposes:

- Combine signal transitions to one transition that always generates the same SCT response. An example is aborting an SCT operation. See Section 14.5.1
- Provide latched inputs so that signals can be blocked for some fixed time before being passed on to the SCT inputs. See Section 14.5.2.

See Section 15.3.5 for details.

16.4 Pin description

The SCT inputs and outputs are movable functions and are assigned to external pins through the switch matrix.

Remark: For applications that require exact timing of the SCT outputs (for example PWM), assign the outputs only to fixed-pin functions to ensure that the output skew is nearly the same for all outputs.

See Section 8.3.1 “Connect an internal signal to a package pin” to assign the SCT functions to package pins.

16.5 General description

The small SCTs are functionally the same as the large SCTs. However, the dither engine is not implemented. For a description of the SCT, see Section 15.5.

16.6 Register description

The register addresses of the State Configurable Timer are shown in Table 235. For most of the SCT registers, the register function depends on the setting of certain other register bits:

1. The UNIFY bit in the CONFIG register determines whether the SCT is used as one 32-bit register (for operation as one 32-bit counter/timer) or as two 16-bit counter/timers named L and H. The setting of the UNIFY bit is reflected in the register map:
   - UNIFY = 1: Only one register is used (for operation as one 32-bit counter/timer).
   - UNIFY = 0: Access the L and H registers by a 32-bit read or write operation or can be read or written to individually (for operation as two 16-bit counter/timers).

   Typically, the UNIFY bit is configured by writing to the CONFIG register before any other registers are accessed.
2. The REGMODEn bits in the REGMODE register determine whether each set of Match/Capture registers uses the match or capture functionality:
   - REGMODEn = 1: Registers operate as match and reload registers.
   - REGMODEn = 0: Registers operate as capture and capture control registers.

Table 235. Register overview: State Configurable Timer (base address 0x1C02 0000 (SCT2) and 0x1C02 4000)

Name	Access	Address offset	Description	Reset value	Reference
CONFIG	R/W	0x000	SCT configuration register	0x0000 7E00	Table 236
CTRL	R/W	0x004	SCT control register	0x0004 0004	Table 237
CTRL_L	R/W	0x004	SCT control register low counter 16-bit	0x0004 0004	Table 237
CTRL_H	R/W	0x006	SCT control register high counter 16-bit	0x0004 0004	Table 237
LIMIT	R/W	0x008	SCT limit register	0x0000 0000	Table 238
LIMIT_L	R/W	0x008	SCT limit register low counter 16-bit	0x0000 0000	Table 238
LIMIT_H	R/W	0x00A	SCT limit register high counter 16-bit	0x0000 0000	Table 238
HALT	R/W	0x00C	SCT halt condition register	0x0000 0000	Table 239
HALT_L	R/W	0x00C	SCT halt condition register low counter 16-bit	0x0000 0000	Table 239
HALT_H	R/W	0x00E	SCT halt condition register high counter 16-bit	0x0000 0000	Table 239
STOP	R/W	0x010	SCT stop condition register	0x0000 0000	Table 240
STOP_L	R/W	0x010	SCT stop condition register low counter 16-bit	0x0000 0000	Table 240
STOP_H	R/W	0x012	SCT stop condition register high counter 16-bit	0x0000 0000	Table 240
START	R/W	0x014	SCT start condition register	0x0000 0000	Table 241
START_L	R/W	0x014	SCT start condition register low counter 16-bit	0x0000 0000	Table 241
START_H	R/W	0x016	SCT start condition register high counter 16-bit	0x0000 0000	Table 241
-	-	0x018 - 0x03C	Reserved		-
COUNT	R/W	0x040	SCT counter register	0x0000 0000	Table 242
COUNT_L	R/W	0x040	SCT counter register low counter 16-bit	0x0000 0000	Table 242
COUNT_H	R/W	0x042	SCT counter register high counter 16-bit	0x0000 0000	Table 242
STATE	R/W	0x044	SCT state register	0x0000 0000	Table 243
STATE_L	R/W	0x044	SCT state register low counter 16-bit	0x0000 0000	Table 243
STATE_H	R/W	0x046	SCT state register high counter 16-bit	0x0000 0000	Table 243
INPUT	RO	0x048	SCT input register	0x0000 0000	Table 244
REGMODE	R/W	0x04C	SCT match/capture registers mode register	0x0000 0000	Table 245
REGMODE_L	R/W	0x04C	SCT match/capture registers mode register low counter 16-bit	0x0000 0000	Table 245
REGMODE_H	R/W	0x04E	SCT match/capture registers mode register high counter 16-bit	0x0000 0000	Table 245
OUTPUT	R/W	0x050	SCT output register	0x0000 0000	Table 246
OUTPUTDIRCTRL	R/W	0x054	SCT output counter direction control register	0x0000 0000	Table 247
RES	R/W	0x058	SCT conflict resolution register	0x0000 0000	Table 248
DMAREQ0	R/W	0x05C	SCT DMA request 0 register	0x0000 0000	Table 249
DMAREQ1	R/W	0x060	SCT DMA request 1 register	0x0000 0000	Table 250
-	-	0x064 - 0x0EC	Reserved	-	-
EVEN	R/W	0x0F0	SCT event enable register	0x0000 0000	Table 251
EVFLAG	R/W	0x0F4	SCT event flag register	0x0000 0000	Table 252
CONEN	R/W	0x0F8	SCT conflict enable register	0x0000 0000	Table 253
CONFLAG	R/W	0x0FC	SCT conflict flag register	0x0000 0000	Table 254

Table 235. Register overview: State Configurable Timer (base address 0x1C02 0000 (SCT2) and 0x1C02 4000)

Name	Access	Address offset	Description	Reset value	Reference
MATCH0 to MATCH7	R/W	0x100 to 0x11C	SCT match value register of match channels 0 to 7; REGMOD0 to REGMODE7 = 0	0x0000 0000	Table 254
MATCH0_L to MATCH7_L	R/W	0x100 to 0x11C	SCT match value register of match channels 0 to 7; low counter 16-bit; REGMOD0_L to REGMODE7_L = 0	0x0000 0000	Table 254
MATCH0_H to MATCH7_H	R/W	0x102 to 0x11E	SCT match value register of match channels 0 to 7; high counter 16-bit; REGMOD0_H to REGMODE7_H = 0	0x0000 0000	Table 254
CAP0 to CAP7		0x100 to 0x11C	SCT capture register of capture channel 0 to 7; REGMOD0 to REGMODE7 = 1	0x0000 0000	Table 256
CAP0_L to CAP7_L		0x100 to 0x11C	SCT capture register of capture channel 0 to 7; low counter 16-bit; REGMOD0_L to REGMODE7_L = 1	0x0000 0000	Table 256
CAP0_H to CAP7_H		0x102 to 0x11E	SCT capture register of capture channel 0 to 7; high counter 16-bit; REGMOD0_H to REGMODE7_H = 1	0x0000 0000	Table 256
MATCHREL0 to MATCHREL7	R/W	0x200 to 0x21C	SCT match reload value register 0 to 7; REGMOD0 = 0 to REGMODE7 = 0	0x0000 0000	Table 257
MATCHREL0_L to MATCHREL7_L	R/W	0x200 to 0x21C	SCT match reload value register 0 to 7; low counter 16-bit; REGMOD0_L = 0 to REGMODE7_L = 0	0x0000 0000	Table 257
MATCHREL0_H to MATCHREL7_H	R/W	0x202 to 0x21E	SCT match reload value register 0 to 7; high counter 16-bit; REGMOD0_H = 0 to REGMODE7_H = 0	0x0000 0000	Table 257
CAPCTRL0 to CAPCTRL7		0x200 to 0x21C	SCT capture control register 0 to 7; REGMOD0 = 1 to REGMODE7 = 1	0x0000 0000	Table 258
CAPCTRL0_L to CAPCTRL7_L		0x200 to 0x21C	SCT capture control register 0 to 7; low counter 16-bit; REGMOD0_L = 1 to REGMODE7_L = 1	0x0000 0000	Table 258
CAPCTRL0_H to CAPCTRL7_H		0x202 to 0x21E	SCT capture control register 0 to 7; high counter 16-bit; REGMOD0 = 1 to REGMODE7 = 1	0x0000 0000	Table 258
EV0_STATE	R/W	0x300	SCT event state register 0	0x0000 0000	Table 259
EV0_CTRL	R/W	0x304	SCT event control register 0	0x0000 0000	Table 260
EV1_STATE	R/W	0x308	SCT event state register 1	0x0000 0000	Table 259
EV1_CTRL	R/W	0x30C	SCT event control register 1	0x0000 0000	Table 260
EV2_STATE	R/W	0x310	SCT event state register 2	0x0000 0000	Table 259
EV2_CTRL	R/W	0x314	SCT event control register 2	0x0000 0000	Table 260
EV3_STATE	R/W	0x318	SCT event state register 3	0x0000 0000	Table 259
EV3_CTRL	R/W	0x31C	SCT event control register 3	0x0000 0000	Table 260
EV4_STATE	R/W	0x320	SCT event state register 4	0x0000 0000	Table 259
EV4_CTRL	R/W	0x324	SCT event control register4	0x0000 0000	Table 260
EV5_STATE	R/W	0x328	SCT event state register 5	0x0000 0000	Table 259
EV5_CTRL	R/W	0x32C	SCT event control register 5	0x0000 0000	Table 260
EV6_STATE	R/W	0x330	SCT event state register 6	0x0000 0000	Table 259
EV6_CTRL	R/W	0x334	SCT event control register 6	0x0000 0000	Table 260
EV7_STATE	R/W	0x338	SCT event state register 7	0x0000 0000	Table 259

Table 235. Register overview: State Configurable Timer (base address 0x1C02 0000 (SCT2) and 0x1C02 4000)

Name	Access	Address offset	Description	Reset value	Reference
EV7_CTRL	R/W	0x33C	SCT event control register 7	0x0000 0000	Table 260
EV8_STATE	R/W	0x340	SCT event state register 8	0x0000 0000	Table 259
EV8_CTRL	R/W	0x344	SCT event control register 8	0x0000 0000	Table 260
EV9_STATE	R/W	0x348	SCT event state register 9	0x0000 0000	Table 259
EV9_CTRL	R/W	0x34C	SCT event control register 9	0x0000 0000	Table 260
OUT0_SET	R/W	0x500	SCT output 0 set register	0x0000 0000	Table 261
OUT0_CLR	R/W	0x504	SCT output 0 clear register	0x0000 0000	Table 262
OUT1_SET	R/W	0x508	SCT output 1 set register	0x0000 0000	Table 261
OUT1_CLR	R/W	0x50C	SCT output 1 clear register	0x0000 0000	Table 262
OUT2_SET	R/W	0x510	SCT output 2 set register	0x0000 0000	Table 261
OUT2_CLR	R/W	0x514	SCT output 2 clear register	0x0000 0000	Table 262
OUT3_SET	R/W	0x518	SCT output 3 set register	0x0000 0000	Table 261
OUT3_CLR	R/W	0x51C	SCT output 3 clear register	0x0000 0000	Table 262
OUT4_SET	R/W	0x520	SCT output 4 set register	0x0000 0000	Table 261
OUT4_CLR	R/W	0x524	SCT output 4 clear register	0x0000 0000	Table 262
OUT5_SET	R/W	0x528	SCT output 5 set register	0x0000 0000	Table 261
OUT5_CLR	R/W	0x52C	SCT output 5 clear register	0x0000 0000	Table 262

16.6.1 SCT configuration register

This register configures the overall operation of the SCT. Write to this register before any other registers.

Table 236. SCT configuration register (CONFIG, address 0x1C02 0000 (SCT2) and 0x1C02 4000 (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
0	UNIFY		SCT operation	0
		0	The SCT operates as two 16-bit counters named L and H.	
		1	The SCT operates as a unified 32-bit counter.	
2:1	CLKMODE		SCT clock mode	00
		0x0	The bus clock clocks the SCT and prescalers.	
		0x1	The SCT clock is the bus clock, but the prescalers are enabled to count only when sampling of the input selected by the CKSEL field finds the selected edge. The minimum pulse width on the clock input is 1 bus clock period. This mode is the high-performance sampled-clock mode.	
		0x2	The input selected by CKSEL clocks the SCT and prescalers. The input is synchronized to the system clock and possibly inverted. The minimum pulse width on the clock input is 1 bus clock period. This mode is the low-power sampled-clock mode.	
		0x3	Prescaled SCT input. The SCT and prescalers are clocked by the input edge selected by the CKSEL field. In this mode, most of the SCT is clocked by the (selected polarity of the) input. The outputs are switched synchronously to the input clock. The input clock rate must be at least half the system clock rate and can the same or faster than the system clock.	

Table 236. SCT configuration register (CONFIG, address 0x1C02 0000 (SCT2) and 0x1C02 4000 (SCT3)) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
6:3	CKSEL		SCT clock select	0000
		0x0	Rising edges on input 0.	
		0x1	Falling edges on input 0.	
		0x2	Rising edges on input 1.	
		0x3	Falling edges on input 1.	
		0x4	Rising edges on input 2.	
		0x5	Falling edges on input 2.	
		0x6	Rising edges on input 3.	
		0x7	Falling edges on input 3.	
7	NORELAOD_L	-	A 1 in this bit prevents the lower match registers from being reloaded from their respective reload registers. Software can write to set or clear this bit at any time. This bit applies to both the higher and lower registers when the UNIFY bit is set.	0
8	NORELOAD_H	-	A 1 in this bit prevents the higher match registers from being reloaded from their respective reload registers. Software can write to set or clear this bit at any time. This bit is not used when the UNIFY bit is set.	0
16:9	INSYNC	-	Synchronization for input N (bit 9 = input 0, bit 10 = input 1,..., bit 16 = input 7). A 1 in one of these bits subjects the corresponding input to synchronization to the SCT clock, before it is used to create an event. If an input is synchronous to the SCT clock, keep its bit 0 for faster response. When the CKMODE field is 1x, the bit in this field, corresponding to the input selected by the CKSEL field, is not used.	1
17	AUTOLIMIT_L	-	A one in this bit causes a match on match register 0 to be treated as a de-facto LIMIT condition without the need to define an associated event. As with any LIMIT event, this automatic limit causes the counter to be cleared to zero in uni-directional mode or to change the direction of count in bi-directional mode. Software can write to set or clear this bit at any time. This bit applies to both the higher and lower registers when the UNIFY bit is set.	
18	AUTOLIMIT_H	-	A one in this bit will cause a match on match register 0 to be treated as a de-facto LIMIT condition without the need to define an associated event. As with any LIMIT event, this automatic limit causes the counter to be cleared to zero in uni-directional mode or to change the direction of count in bi-directional mode. Software can write to set or clear this bit at any time. This bit is not used when the UNIFY bit is set.	
31:19	-		Reserved	-

16.6.2 SCT control register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers CTRL_L and CTRL_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

All bits in this register can be written to when the counter is stopped or halted. When the counter is running, the only bits that can be written are STOP or HALT.(Other bits can be written in a subsequent write after HALT is set to 1.)

Remark: If CLKMODE = 0x3 is selected, wait at least 12 system clock cycles between a write access to the H, L or unified version of this register and the next write access. This restriction does not apply when writing to the HALT bit or bits and then writing to the CTRL register again to restart the counters - for example because software must update the MATCH register, which is only allowed when the counters are halted.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 237. SCT control register (CTRL, address 0x1C02 0004 (SCT2) and 0x1C02 4004 (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
0	DOWN_L	-	This bit is 1 when the L or unified counter is counting down. Hardware sets this bit when the counter limit is reached and BIDIR is 1. Hardware clears this bit when the counter is counting down and a limit condition occurs or when the counter reaches 0.	0
1	STOP_L	-	When this bit is 1 and HALT is 0, the L or unified counter does not run, but I/O events related to the counter can occur. If such an event matches the mask in the Start register, this bit is cleared and counting resumes.	0
2	HALT_L	-	When this bit is 1, the L or unified counter does not run and no events can occur. A reset sets this bit. When the HALT_L bit is one, the STOP_L bit is cleared. If you want to remove the halt condition and keep the SCT in the stop condition (not running), then you can change the halt and stop condition with one single write to this register. **Remark:** Once set, only software can clear this bit to restore counter operation.	1
3	CLRCTR_L	-	Writing a 1 to this bit clears the L or unified counter. This bit always reads as 0.	0
4	BIDIR_L		L or unified counter direction select	0
		0	The counter counts up to its limit condition, then is cleared to zero.	
		1	The counter counts up to its limit, then counts down to a limit condition or to 0.	
12:5	PRE_L	-	Specifies the factor by which the SCT clock is prescaled to produce the L or unified counter clock. The counter clock is clocked at the rate of the SCT clock divided by PRE_L+1. **Remark:** Clear the counter (by writing a 1 to the CLRCTR bit) whenever changing the PRE value.	0
15:13	-		Reserved	
16	DOWN_H	-	This bit is 1 when the H counter is counting down. Hardware sets this bit when the counter limit is reached and BIDIR is 1. Hardware clears this bit when the counter is counting down and a limit condition occurs or when the counter reaches 0.	0
17	STOP_H	-	When this bit is 1 and HALT is 0, the H counter does not, run but I/O events related to the counter can occur. If such an event matches the mask in the Start register, this bit is cleared and counting resumes.	0
18	HALT_H	-	When this bit is 1, the H counter does not run and no events can occur. A reset sets this bit. When the HALT_H bit is one, the STOP_H bit is cleared. If you want to remove the halt condition and keep the SCT in the stop condition (not running), then you can change the halt and stop condition with one single write to this register. **Remark:** Once set, this bit can only be cleared by software to restore counter operation.	1

Table 237. SCT control register (CTRL, address 0x1C02 0004 (SCT2) and 0x1C02 4004 (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
19	CLRCTR_H	-	Writing a 1 to this bit clears the H counter. This bit always reads as 0.	0
20	BIDIR_H		Direction select	0
		0	The H counter counts up to its limit condition, then is cleared to zero.	
		1	The H counter counts up to its limit, then counts down to a limit condition or to 0.	
28:21	PRE_H	-	Specifies the factor by which the SCT clock is prescaled to produce the H counter clock. The counter clock is clocked at the rate of the SCT clock divided by PRELH+1. **Remark:** Clear the counter (by writing a 1 to the CLRCTR bit) whenever changing the PRE value.	0
31:29	-		Reserved	

16.6.3 SCT limit register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers LIMIT_L and LIMIT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

The bits in this register set which events act as counter limits. When a limit event occurs, the counter is cleared to zero in unidirectional mode or changes the direction of count in bidirectional mode. When the counter reaches all ones, this state is always treated as a limit event, and the counter is cleared in unidirectional mode or, in bidirectional mode, begins counting down on the next clock edge - even if no limit event as defined by the SCT limit register has occurred.

Note that in addition to using this register to specify events that serve as limits, it is also possible to automatically cause a limit condition whenever a match register 0 match occurs. This eliminates the need to define an event for the sole purpose of creating a limit. The AUTOLIMITL and AUTOLIMITH bits in the configuration register enable/disable this feature (see Table 236).

Table 238. SCT limit register (LIMIT, address 0x1C02 0008 (SCT2) and 0x1C02 4008 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	LIMMSK_L	If bit n is one, event n is used as a counter limit for the L or unified counter (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	LIMMSK_H	If bit n is one, event n is used as a counter limit for the H counter (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

16.6.4 SCT halt condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers HALT_L and HALT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Remark: Any event halting the counter disables its operation until software clears the HALT bit (or bits) in the CTRL register (Table 237).

Table 239. SCT halt condition register (HALT, address 0x1C02 000C (SCT2) and 0x1C02 400C (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	HALTMSK_L	If bit n is one, event n sets the HALT_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	HALTMSK_H	If bit n is one, event n sets the HALT_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

16.6.5 SCT stop condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers STOPT_L and STOP_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Table 240. SCT stop condition register (STOP, address 0x1C02 0010 (SCT2) and 0x1C02 4010 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	STOPMSK_L	If bit n is one, event n sets the STOP_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	STOPMSK_H	If bit n is one, event n sets the STOP_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

16.6.6 SCT start condition register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers START_L and START_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

The bits in this register select which events, if any, clear the STOP bit in the Control register. (Since no events can occur when HALT is 1, only software can clear the HALT bit by writing the Control register.)

Table 241. SCT start condition register (START, address 0x1C02 0014 (SCT2) and 0x1C02 4014 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	STARTMSK_L	If bit n is one, event n clears the STOP_L bit in the CTRL register (event 0 = bit 0, event 1 = bit 1, event 15 = bit 15).	0
31:16	STARTMSK_H	If bit n is one, event n clears the STOP_H bit in the CTRL register (event 0 = bit 16, event 1 = bit 17, event 15 = bit 31).	0

16.6.7 SCT counter register

If UNIFY = 1 in the CONFIG register, the counter is a unified 32-bit register and both the _L and _H bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers COUNT_L and COUNT_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation. In this case, the L and H registers count independently under the control of the other registers.

Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Table 242. SCT counter register (COUNT, address 0x1C02 0040 (SCT2) and 0x1C02 4040 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	CTR_L	When UNIFY = 0, read or write the 16-bit L counter value. When UNIFY = 1, read or write the lower 16 bits of the 32-bit unified counter.	0
31:16	CTR_H	When UNIFY = 0, read or write the 16-bit H counter value. When UNIFY = 1, read or write the upper 16 bits of the 32-bit unified counter.	0

16.6.8 SCT state register

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers STATE_L and STATE_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Software can read the state associated with a counter at any time. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

The state variable is the main feature that distinguishes the SCT from other counter/timer/PWM blocks. Events can be made to occur only in certain states. Events, in turn, can perform the following actions:

- set and clear outputs
- limit, stop, and start the counter
- cause interrupts and DMA requests
- modify the state variable

The value of a state variable is completely under the control of the application. If an application does not use states, the value of the state variable remains zero, which is the default value.

A state variable can be used to track and control multiple cycles of the associated counter in any desired operational sequence. The state variable is logically associated with a state machine diagram which represents the SCT configuration. See Section 16.6.23 and 16.6.24 for more about the relationship between states and events.

The STATELD/STADEV fields in the event control registers of all defined events set all possible values for the state variable. The change of the state variable during multiple counter cycles reflects how the associated state machine moves from one state to the next.

Table 243. SCT state register (STATE, address 0x1C02 0044 (SCT2) and 0x1C02 4044 (SCT3)) bit description

Bit	Symbol	Description	Reset value
4:0	STATE_L	State variable.	0
15:5	-	Reserved.	-
20:16	STATE_H	State variable.	0
31:21	-	Reserved.	

16.6.9 SCT input register

Software can read the state of the SCT inputs in this read-only register in slightly different forms.

1. The AIN bit represents the input sampled by the SCT clock. This corresponds to a nearly direct read-out of the input but can cause spurious fluctuations in case of an asynchronous input signal.
2. The SIN bit represents the input sampled by the SCT clock after the INSYNC select (this signal is also used for event generation):
   - If the INSYNC bit is set for the input, the input is synchronized to the SCT clock using three SCT clock cycles resulting in a stable signal that is delayed by three SCT clock cycles.
   - If the INSYNC bit is not set, the SIN bit value is the same as the AIN bit value.

Table 244. SCT input register (INPUT, address 0x1C02 0048 (SCT2) and 0x1C02 4048 (SCT3)) bit description

Bit	Symbol	Description	Reset value
0	AIN0	Input 0 state. Direct read.	-
1	AIN1	Input 1 state. Direct read.	-
2	AIN2	Input 2 state. Direct read.	-
15:3	-	Reserved.	-
16	SIN0	Input 0 state.	-
17	SIN1	Input 1 state.	-
18	SIN2	Input 2 state.	-
31:19	-	Reserved	-

16.6.10 SCT match/capture registers mode register

If UNIFY = 1 in the CONFIG register, only the _L bits of this register are used. The L bits control whether each set of match/capture registers operates as unified 32-bit capture/match registers.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers REGMODE_L and REGMODE_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.The _L bits/registers control the L match/capture registers, and the _H bits/registers control the H match/capture registers.

The SCT contains 16 Match/Capture register pairs. The Register Mode register selects whether each register pair acts as a Match register (see Section 16.6.19) or as a Capture register (see Section 16.6.20). Each Match/Capture register has an accompanying register which serves as a Reload register when the register is used as a Match register (Section 16.6.21) or as a Capture-Control register when the register is used as a capture register (Section 16.6.22). REGMODE_H is used only when the UNIFY bit is 0.

Table 245. SCT match/capture registers mode register (REGMODE, address 0x1C02 004C (SCT2) and 0x1C02 404C (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	REGMOD_L	Each bit controls one pair of match/capture registers (register 0 = bit 0, register 1 = bit 1,..., register 15 = bit 15). 0 = registers operate as match registers. 1 = registers operate as capture registers.	0
31:16	REGMOD_H	Each bit controls one pair of match/capture registers (register 0 = bit 16, register 1 = bit 17,..., register 15 = bit 31). 0 = registers operate as match registers. 1 = registers operate as capture registers.	0

16.6.11 SCT output register

The SCT supports 6 outputs, each of which has a corresponding bit in this register.

Software can write to any of the output registers when both counters are halted to control the outputs directly. Writing to the OUT register is only allowed when all counters (L-counter, H-counter, and unified counter) are halted (HALT bits are set to 1 in the CTRL register).

Software can read this register at any time to sense the state of the outputs.

Table 246. SCT output register (OUTPUT, address 0x1C02 0050 (SCT2) and 0x1C02 4050 (SCT3)) bit description

Bit	Symbol	Description	Reset value
5:0	OUT	Writing a 1 to bit n makes the corresponding output HIGH. 0 makes the corresponding output LOW (output 0 = bit 0, output 1 = bit 1,..., output 5 = bit 5).	0
31:6	-	Reserved	

16.6.12 SCT bidirectional output control register

This register specifies (for each output) the impact of the counting direction on the meaning of set and clear operations on the output (see Section 16.6.25 and Section 16.6.26).

Table 247. SCT bidirectional output control register (OUTPUTDIRCTRL, address 0x1C02 0054 (SCT2) and 0x1C02 4054 (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	SETCLR0		Set/clear operation on output 0. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
3:2	SETCLR1		Set/clear operation on output 1. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
5:4	SETCLR2		Set/clear operation on output 2. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
7:6	SETCLR3		Set/clear operation on output 3. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
9:8	SETCLR4		Set/clear operation on output 4. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
11:10	SETCLR5		Set/clear operation on output 5. Value 0x3 is reserved. Do not program this value.	0
		0x0	Set and clear do not depend on any counter.	
		0x1	Set and clear are reversed when counter L or the unified counter is counting down.	
		0x2	Set and clear are reversed when counter H is counting down. Do not use if UNIFY = 1.	
31:12	-		Reserved	-

16.6.13 SCT conflict resolution register

The registers OUTn_SET (Section 16.6.25) and OUTn_CLR (Section 16.6.26) allow both setting and clearing to be indicated for an output in the same clock cycle, even for the same event. This SCT conflict resolution register resolves this conflict.

To enable an event to toggle an output, set the OnRES value to 0x3 in this register, and set the event bits in both the Set and Clear registers.

Table 248. SCT conflict resolution register (RES, address 0x1C02 0058 (SCT2) and 0x1C02 4058 (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	O0RES		Effect of simultaneous set and clear on output 0.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR0 field).	
		0x2	Clear output (or set based on the SETCLR0 field).	
		0x3	Toggle output.	
3:2	O1RES		Effect of simultaneous set and clear on output 1.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR1 field).	
		0x2	Clear output (or set based on the SETCLR1 field).	
		0x3	Toggle output.	
5:4	O2RES		Effect of simultaneous set and clear on output 2.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR2 field).	
		0x2	Clear output n (or set based on the SETCLR2 field).	
		0x3	Toggle output.	
7:6	O3RES		Effect of simultaneous set and clear on output 3.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR3 field).	
		0x2	Clear output (or set based on the SETCLR3 field).	
		0x3	Toggle output.	
9:8	O4RES		Effect of simultaneous set and clear on output 4.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR4 field).	
		0x2	Clear output (or set based on the SETCLR4 field).	
		0x3	Toggle output.	
11:10	O5RES		Effect of simultaneous set and clear on output 5.	0
		0x0	No change.	
		0x1	Set output (or clear based on the SETCLR5 field).	
		0x2	Clear output (or set based on the SETCLR5 field).	
		0x3	Toggle output.	
31:12	-	-	Reserved	-

16.6.14 SCT DMA request 0 and 1 registers

The SCT includes two DMA request outputs. These registers enable the DMA requests to be triggered when a particular event occurs or when counter Match registers are loaded from its Reload registers.

Event-triggered DMA requests are particularly useful for launching DMA activity to or from other peripherals under the control of the SCT.

Table 249. SCT DMA 0 request register (DMAREQ0, address 0x1C02 005C (SCT2) and 0x1C02 405C (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	DEV_0	If bit n is one, event n sets DMA request 0 (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
29:16	-	Reserved	-
30	DRL0	A 1 in this bit makes the SCT set DMA request 0 when it loads the Match_L/Unified registers from the Reload_L/Unified registers.	
31	DRQ0	This read-only bit indicates the state of DMA Request 0	

Table 250. SCT DMA 1 request register (DMAREQ1, address 0x1C02 0060 (SCT2) and 0x1C02 4060 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	DEV_1	If bit n is one, event n sets DMA request 1 (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
29:16	-	Reserved	-
30	DRL1	A 1 in this bit makes the SCT set DMA request 1 when it loads the Match L/Unified registers from the Reload L/Unified registers.	
31	DRQ1	This read-only bit indicates the state of DMA Request 1.	

16.6.15 SCT flag enable register

This register enables flags to request an interrupt if the FLAGn bit in the SCT event flag register (Section 16.6.16) is also set.

Table 251. SCT flag enable register (EVEN, address 0x1C02 00F0 (SCT2) and 0x1C02 40F0 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	IEN	The SCT requests interrupt when bit n of this register and the event flag register are both one (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
31:16	-	Reserved	

16.6.16 SCT event flag register

This register records events. Writing ones to this register clears the corresponding flags and negates the SCT interrupt request if all enabled Flag bits are zero.

Table 252. SCT event flag register (EVFLAG, address 0x1C02 00F4 (SCT2) and 0x1C02 40F4 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	FLAG	Bit n is one if event n has occurred since reset or a 1 was last written to this bit (event 0 = bit 0, event 1 = bit 1,..., event 15 = bit 15).	0
31:16	-	Reserved	-

16.6.17 SCT conflict enable register

This register enables the “no change conflict” events specified in the SCT conflict resolution register to request an IRQ.

Table 253. SCT conflict enable register (CONEN, address 0x1C02 00F8 (SCT2) and 0x1C02 40F8 (SCT3)) bit description

Bit	Symbol	Description	Reset value
15:0	NCEN	The SCT requests interrupt when bit n of this register and the SCT conflict flag register are both one (output 0 = bit 0, output 1 = bit 1,..., output 15 = bit 15).	0
31:16	-	Reserved	

16.6.18 SCT conflict flag register

This register records interrupt-enabled no-change conflict events and provides details of a bus error. Writing ones to the NCFLAG bits clears the corresponding read bits and negates the SCT interrupt request if all enabled Flag bits are zero.

Table 254. SCT conflict flag register (CONFLAG, address 0x1C02 00FC (SCT2) and 0x1C02 40FC (SCT3)) bit description

Bit	Symbol	Description	Reset value
5:0	NCFLAG	Bit n is one if a no-change conflict event occurred on output n since reset or a 1 was last written to this bit (output 0 = bit 0, output 1 = bit 1,..., output 5 = bit 5).	0
29:6	-	Reserved.	-
30	BUSERRL	The most recent bus error from this SCT involved writing CTR L/Unified, STATE L/Unified, MATCH L/Unified, or the Output register when the L/U counter was not halted. A word write to certain L and H registers can be half successful and half unsuccessful.	0
31	BUSERRH	The most recent bus error from this SCT involved writing CTR H, STATE H, MATCH H, or the Output register when the H counter was not halted.	0

16.6.19 SCT match registers 0 to 7 (REGMODEn bit = 0)

Match registers are compared to the counters to help create events. When the UNIFY bit is 0, the L and H registers are independently compared to the L and H counters. When UNIFY is 1, the L and H registers hold a 32-bit value that is compared to the unified counter. A Match can only occur in a clock in which the counter is running (STOP and HALT are both 0).

Match registers can be read at any time. Writing to the MATCH_L, MATCH_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register). Match events occur in the SCT clock in which the counter is (or would be) incremented to the next value. When a Match event limits its counter as described in Section 16.6.3, the value in the Match register is the last value of the counter before it is cleared to zero (or decremented if BIDIR is 1).

There is no “write-through” from Reload registers to Match registers. Before starting a counter, software can write one value to the Match register used in the first cycle of the counter and a different value to the corresponding Match Reload register used in the second cycle.

Table 255. SCT match registers 0 to 7 (MATCH[0:7], address 0x1C02 0100 (MATCH0) to 0x1C02 011C (MATCH7) (SCT2) and address 0x1C02 4100 (MATCH0) to 0x1C02 411C (MATCH7) (SCT3)) bit description (REGMODEn bit = 0)

Bit	Symbol	Description	Reset value
15:0	MATCHn_L	When UNIFY = 0, read or write the 16-bit value to be compared to the L counter. When UNIFY = 1, read or write the lower 16 bits of the 32-bit value to be compared to the unified counter.	0
31:16	MATCHn_H	When UNIFY = 0, read or write the 16-bit value to be compared to the H counter. When UNIFY = 1, read or write the upper 16 bits of the 32-bit value to be compared to the unified counter.	0

16.6.20 SCT capture registers 0 to 7 (REGMODEn bit = 1)

These registers allow software to read the counter values at which the event selected by the corresponding Capture Control registers occurred.

Table 256. SCT capture registers 0 to 7 (CAP[0:7], address 0x1C02 0100 (CAP0) to 0x1C02 011C (CAP7) (SCT2) and address 0x1C02 4100 (CAP0) to 0x1C02 411C (CAP7) (SCT3)) bit description (REGMODEn bit = 1)

Bit	Symbol	Description	Reset value
15:0	CAPn_L	When UNIFY = 0, read the 16-bit counter value at which this register was last captured. When UNIFY = 1, read the lower 16 bits of the 32-bit value at which this register was last captured.	0
31:16	CAPn_H	When UNIFY = 0, read the 16-bit counter value at which this register was last captured. When UNIFY = 1, read the upper 16 bits of the 32-bit value at which this register was last captured.	0

16.6.21 SCT match reload registers 0 to 7 (REGMODEn bit = 0)

A Match register (L, H, or unified 32-bit) is loaded from the corresponding Reload register when BIDIR is 0 and the counter reaches its limit condition, or when BIDIR is 1 and the counter reaches 0.

Table 257. SCT match reload registers 0 to 7 (MATCHREL[0:7], address 0x1C02 0200 (MATCHREL0) to 0x1C02 021C (MATCHREL7) (SCT2) and 0x1C02 4200 (MATCHREL0) to 0x1C02 421C (MATCHREL7) (SCT3)) bit description (REGMODEn bit = 0)

Bit	Symbol	Description	Reset value
15:0	RELOADn_L	When UNIFY = 0, read or write the 16-bit value to be loaded into the SCTMATCHn_L register. When UNIFY = 1, read or write the lower 16 bits of the 32-bit value to be loaded into the MATCHn register.	0
31:16	RELOADn_H	When UNIFY = 0, read or write the 16-bit to be loaded into the MATCHn_H register. When UNIFY = 1, read or write the upper 16 bits of the 32-bit value to be loaded into the MATCHn register.	0

16.6.22 SCT capture control registers 0 to 7 (REGMODEn bit = 1)

If UNIFY = 1 in the CONFIG register, only the _L bits are used.

If UNIFY = 0 in the CONFIG register, this register can be written to as two registers CAPCTRLn_L and CAPCTRLn_H. Both the L and H registers can be read or written individually or in a single 32-bit read or write operation.

Each Capture Control register (L, H, or unified 32-bit) controls which events load the corresponding Capture register from the counter.

Table 258. SCT capture control registers 0 to 7 (CAPCTRL[0:7], address 0x1C02 0200 (CAPCTRL0) to 0x1C02 021C (CAPCTRL7) (SCT2) and 0x1C02 4200 (CAPCTRL0) to 0x1C02 421C (CAPCTRL7) (SCT3)) bit description (REGMODEn bit = 1)

Bit	Symbol	Description	Reset value
9:0	CAPCONn_L	If bit m is one, event m causes the CAPn_L (UNIFY = 0) or the CAPn (UNIFY = 1) register to be loaded (event 0 = bit 0, event 1 = bit 1,..., event 9 = bit 9).	0
15:10	-	Reserved.	-
24:16	CAPCONn_H	If bit m is one, event m causes the CAPn_H (UNIFY = 0) register to be loaded (event 0 = bit 16, event 1 = bit 17,..., event 9 = bit 24).	0
31:25	-	Reserved.	-

16.6.23 SCT event state registers 0 to 9

Each event has one associated SCT event state mask register that allow this event to happen in one or more states of the counter selected by the HEVENT bit in the corresponding EVCTRLn register.

An event n is disabled when its EVn_STATE register contains all zeros, since it is masked regardless of the current state.

In simple applications that do not use states, write 0x01 to this register to enable an event. Since the state always remains at its reset value of 0, writing 0x01 effectively permanently state-enables this event.

Table 259. SCT event state mask registers 0 to 9 (EV[0:9]_STATE, addresses 0x1C02 0300 (EV0_STATE) to 0x1C02 0348 (EV9_STATE) (SCT2) and 0x1C02 4300 (EV0_STATE) to 0x1C02 4348 (EV9_STATE) (SCT3)) bit description

Bit	Symbol	Description	Reset value
9:0	STATEMSKn	If bit m is one, event n (n= 0 to 9) happens in state m of the counter selected by the HEVENT bit (m = state number; state 0 = bit 0, state 1= bit 1,..., state 9 = bit 9).	0
31:10	-	Reserved.	-

16.6.24 SCT event control registers 0 to 9

This register defines the conditions for event n to occur, other than the state variable which is defined by the state mask register. Most events are associated with a particular counter (high, low, or unified), in which case the event can depend on a match to that register. The other possible ingredient of an event is a selected input or output signal.

When the UNIFY bit is 0, each event is associated with a particular counter by the HEVENT bit in its event control register. An event cannot occur when its related counter is halted nor when the current state is not enabled to cause the event as specified in its event mask register. An event is permanently disabled when its event state mask register contains all 0s.

An enabled event can be programmed to occur based on a selected input or output edge or level and/or based on its counter value matching a selected match register (STOP bit = 0). An event can be enabled by the event counter's HALT bit and STATE register. In bi-directional mode, events can also be enabled based on the direction of count.

Each event can modify its counter STATE value. If more than one event associated with the same counter occurs in a given clock cycle, only the state change specified for the highest-numbered event among them takes place. Other actions dictated by any simultaneously occurring events all take place.

Table 260. SCT event control register 0 to 9 (EV[0:9]_CTRL, address 0x1C02 0304 (EV0_CTRL) to 0x1C02 034C (EV9_CTRL) (SCT2) and 0x1C02 4304 (EV0_CTRL) to 0x1C02 434C (EV9_CTRL) (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
3:0	MATCHSEL	-	Selects the Match register associated with this event (if any). A match can occur only when the counter selected by the HEVENT bit is running.	0
4	HEVENT		Select L/H counter. Do not set this bit if UNIFY = 1.	0
		0	Selects the L state and the L match register selected by MATCHSEL.	
		1	Selects the H state and the H match register selected by MATCHSEL.	
5	OUTSEL		Input/output select	0
		0	Selects the inputs elected by IOSEL.	
		1	Selects the outputs selected by IOSEL.	
9:6	IOSEL	-	Selects the input or output signal associated with this event (if any). Do not select an input in this register, if CKMODE is 1x. In this case the clock input is an implicit ingredient of every event.	0
11:10	IOCOND		Selects the I/O condition for event n. (The detection of edges on outputs lag the conditions that switch the outputs by one SCT clock). In order to guarantee proper edge/state detection, an input must have a minimum pulse width of at least one SCT clock period .	0
		0x0	LOW	
		0x1	Rise	
		0x2	Fall	
		0x3	HIGH	
13:12	COMBMODE		Selects how the specified match and I/O condition are used and combined.	
		0x0	OR. The event occurs when either the specified match or I/O condition occurs.	
		0x1	MATCH. Uses the specified match only.	
		0x2	IO. Uses the specified I/O condition only.	
		0x3	AND. The event occurs when the specified match and I/O condition occur simultaneously.	
14	STATELD		This bit controls how the STATEV value modifies the state selected by HEVENT when this event is the highest-numbered event occurring for that state.	
		0	STATEV value is added into STATE (the carry-out is ignored).	
		1	STATEV value is loaded into STATE.	

Table 260. SCT event control register 0 to 9 (EV[0:9]_CTRL, address 0x1C02 0304 (EV0_CTRL) to 0x1C02 034C (EV9_CTRL) (SCT2) and 0x1C02 4304 (EV0_CTRL) to 0x1C02 434C (EV9_CTRL) (SCT3)) bit description

Bit	Symbol	Value	Description	Reset value
19:15	STATEV		This value is loaded into or added to the state selected by HEVENT, depending on STATELD, when this event is the highest-numbered event occurring for that state. If STATELD and STATEV are both zero, there is no change to the STATE value.	
20	MATCHMEM		If this bit is one and the COMBMODE field specifies a match component to the triggering of this event, then a match is considered to be active whenever the counter value is GREATER THAN OR EQUAL TO the value specified in the match register when counting up, LESS THEN OR EQUAL TO the match value when counting down. If this bit is zero, a match is only be active during the cycle when the counter is equal to the match value.	
22:21	DIRECTION		Direction qualifier for event generation. This field only applies when the counters are operating in BIDIR mode. If BIDIR = 0, the SCT ignores this field. Value 0x3 is reserved.	
		0x0	Direction independent. This event is triggered regardless of the count direction.	
		0x1	Counting up. This event is triggered only during up-counting when BIDIR = 1.	
		0x2	Counting down. This event is triggered only during down-counting when BIDIR = 1.	
31:23	-		Reserved	

16.6.25 SCT output set registers 0 to 5

Each output n has one set register that controls how events affect each output. Whether outputs are set or cleared depends on the setting of the SETCLRn field in the OUTPUTDIRCTRL register.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 261. SCT output set register (OUT[0:5]_SET, address 0x1C02 0500 (OUT0_SET) to 0x1C02 0528 (OUT5_SET) (SCT2) and 0x1C02 4500 (OUT0_SET) to 0x1C02 4528 (OUT5_SET) (SCT3)) bit description

Bit	Symbol	Description	Reset value
9:0	SET	A 1 in bit m selects event m to set output n (or clear it if SETCLRn = 0x1 or 0x2) event 0 = bit 0, event 1 = bit 1,..., event 9 = bit 9.	0
31:10	-	Reserved	

16.6.26 SCT output clear registers 0 to 5

Each output n has one clear register that controls how events affect each output. Whether outputs are set or cleared depends on the setting of the SETCLRn field in the OUTPUTDIRCTRL register.

Remark: If the SCTimer/PWM is operating as two 16-bit counters, events can only modify the state of the outputs when neither counter is halted. This is true regardless of what triggered the event.

Table 262. SCT output clear register (OUT[0:5]_CLR, address 0x1C02 0504 (OUT0_CLR) to 0x1C02 052C (OUT5_CLR) (SCT2) and 0x1C02 4504 (OUT0_CLR) to 0x1C02452C (OUT5_CLR) (SCT2)) bit description

Bit	Symbol	Description	Reset value
9:0	CLR	A 1 in bit m selects event m to clear output n (or set it if SETCLRn = 0x1 or 0x2) event 0 = bit 0, event 1 = bit 1,..., event 9 = bit 9.	0
31:10	-	Reserved	

16.7 Functional description

The small SCTs are functionally the same as the large SCTs. However, the dither engine is not implemented. For a functional description of the SCT, see Section 15.7.

17.1 How to read this chapter

The watchdog timer is identical on all LPC15xx parts.

17.2 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 prior to watchdog time-out.
- Programmable 24-bit timer with internal fixed pre-scaler.
- Selectable time period from 1,024 watchdog clocks ($T_{WDCLK} \times 256 \times 4$) to over 67 million watchdog clocks ($T_{WDCLK} \times 2^{24} \times 4$) in increments of 4 watchdog clocks.
- “Safe” watchdog operation. Once enabled, requires a hardware reset or a Watchdog reset to be disabled.
- Incorrect feed sequence causes immediate watchdog event if enabled.
- The watchdog reload value can optionally be protected such that it can only be changed after the “warning interrupt” time is reached.
- Flag to indicate Watchdog reset.
- The Watchdog clock (WDCLK) source is the fixed 503 kHz clock (+/- 40 %) provided by the low-power watchdog oscillator.
- The Watchdog timer can be configured to run in Deep-sleep or Power-down mode.
- Debug mode.

17.3 Basic configuration

The WWDT is configured through the following registers:

- Power to the register interface (WWDT PCLK clock): In the SYSAHBCLKCTRL0 register, set bit 22 in Table 50.
- Enable the WWDT clock source (the watchdog oscillator) in the PDRUNCFG register (Table 75). This is the clock source for the timer base.
- For waking up from a WWDT interrupt, enable the watchdog interrupt for wake-up in the STARTERP0 register (Table 76).

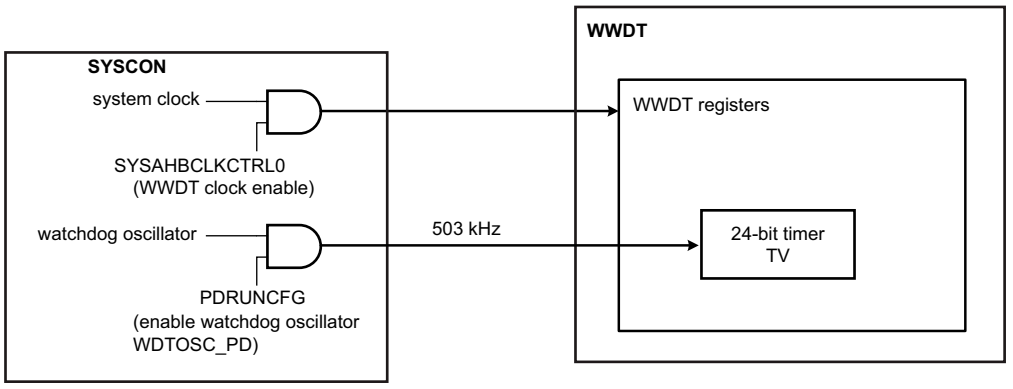


Fig 45. WWDT timing

17.4 Pin description

The WWDT has no external pins.

17.5 General description

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 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, application code could be stuck in an interrupt service that contains a watchdog feed. Setting the window such that this would result in an early feed will generate a watchdog event, allowing for system recovery.

The Watchdog consists of a fixed (divide by 4) pre-scaler and a 24-bit counter which decrements when clocked. The minimum value from which the counter decrements is 0xFF. Setting a value lower than 0xFF causes 0xFF to be loaded in the counter. Hence the minimum Watchdog interval is ($T_{WDCLK} \times 256 \times 4$) and the maximum Watchdog interval is ($T_{WDCLK} \times 2^{24} \times 4$) in multiples of ($T_{WDCLK} \times 4$). The Watchdog should be used in the following manner:

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

When the Watchdog Timer is configured so that a watchdog event will cause a reset and the counter reaches zero, the CPU will be reset, loading the stack pointer and program counter from the vector table as for an external reset. The Watchdog time-out flag (WDTOF) can be examined to determine if the Watchdog has caused the reset condition. The WDTOF flag must be cleared by software.

When the Watchdog Timer is configured to generate a warning interrupt, the interrupt will occur when the counter matches the value defined by the WARNINT register.

17.5.1 Block diagram

The block diagram of the Watchdog is shown below in the Figure 46. The synchronization logic (PCLK - WDCLK) is not shown in the block diagram.

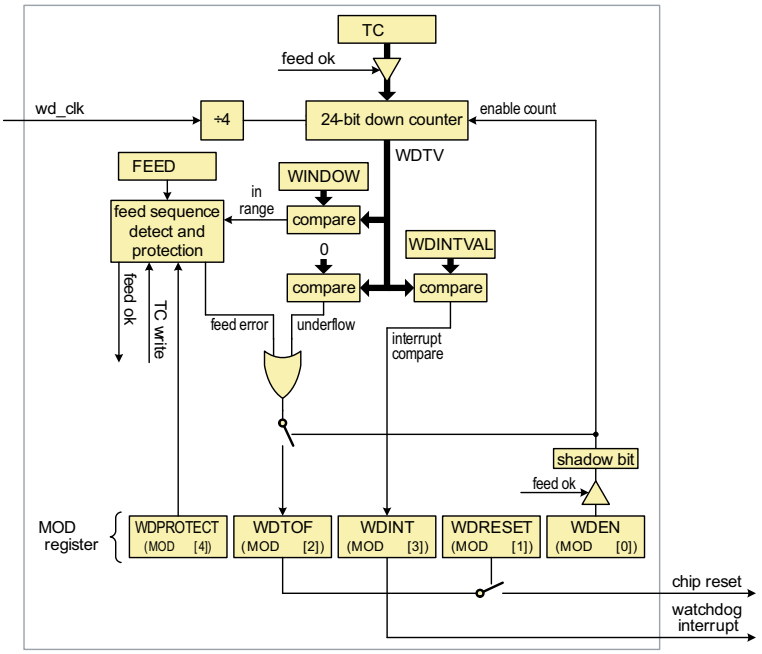


Fig 46. Windowed Watchdog timer block diagram

17.5.2 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 watchdog oscillator.

The synchronization logic between the two clock domains works as follows: When the MOD and TC registers are updated by APB operations, the new value will take effect in 3 WDCLK cycles on the logic in the WDCLK clock domain.

When the watchdog timer is counting on WDCLK, the synchronization logic will first lock the value of the counter on WDCLK and then synchronize it with PCLK, so that the CPU can read the WDTV register.

Remark: Because of the synchronization step, software must add a delay of three WDCLK clock cycles between the feed sequence and the time the WDPROTECT bit is enabled in the MOD register. The length of the delay depends on the selected watchdog clock WDCLK.

17.5.3 Using the WWDT lock features

The WWDT supports the following lock features which can be enabled to ensure that the WWDT is running at all times:

- Disabling the WWDT clock source
- Changing the WWDT reload value

17.5.3.1 Disabling the WWDT clock source

If bit 5 in the WWDT MOD register is set, the WWDT clock source is locked and the watchdog oscillator can not be disabled either by software or by hardware when Sleep, Deep-sleep or Power-down modes are entered.

17.5.3.2 Changing the WWDT reload value

If bit 4 is set in the WWDT MOD register, the watchdog time-out value (TC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW.

The reload overwrite lock mechanism can only be disabled by a reset of any type.

17.6 Register description

The Watchdog Timer contains the registers shown in Table 263.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

Table 263. Register overview: Watchdog timer (base address 0x4002 C000)

Name	Access	Address offset	Description	Reset value	Reference
MOD	R/W	0x000	Watchdog mode register. This register contains the basic mode and status of the Watchdog Timer.	0	Table 264
TC	R/W	0x004	Watchdog timer constant register. This 24-bit register determines the time-out value.	0xFF	Table 266
FEED	WO	0x008	Watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer with the value contained in WDTC.	NA	Table 267
TV	RO	0x00C	Watchdog timer value register. This 24-bit register reads out the current value of the Watchdog timer.	0xFF	Table 268
-	-	0x010	Reserved	-	-
WARNINT	R/W	0x014	Watchdog Warning Interrupt compare value.	0	Table 269
WINDOW	R/W	0x018	Watchdog Window compare value.	0xFF FFFF	Table 270

17.6.1 Watchdog mode register

The WDMOD register controls the operation of the Watchdog. Note that a watchdog feed must be performed before any changes to the WDMOD register take effect.

Table 264. Watchdog mode register (MOD, 0x4002 C000) 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 re-written with a 0. Once this bit is set to one and performing a watchdog feed, the watchdog timer starts running permanently.	0
		0	Stop. The watchdog timer is stopped.	
		1	Run. The watchdog timer is running.	
1	WDRESET		Watchdog reset enable bit. Once this bit has been written with a 1 it cannot be re-written with a 0.	0
		0	Interrupt. A watchdog time-out will not cause a chip reset.	
		1	Reset. A watchdog time-out will cause a chip reset.	
2	WDTOF		Watchdog time-out flag. Set when the watchdog timer times out, by a feed error, or by events associated with WDPROTECT. Cleared by software. Causes a chip reset if WDRESET = 1.	0 (only after external reset)

Table 264. Watchdog mode register (MOD, 0x4002 C000) bit description

Bit	Symbol	Value	Description	Reset value
3	WDINT		Warning interrupt flag. Set when the timer reaches the value in WDWARNINT. Cleared by software.	0
4	WDPROTECT		Watchdog update mode. This bit can be set once by software and is only cleared by a reset.	0
		0	Flexible. The watchdog time-out value (TC) can be changed at any time.	
		1	Threshold. The watchdog time-out value (TC) can be changed only after the counter is below the value of WDWARNINT and WDWINDOW.	
5	LOCK		A 1 in this bit prevents disabling or powering down the watchdog oscillator. This bit can be set once by software and is only cleared by any reset.	0
31:6	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Once the **WDEN**, **WDPROTECT**, or **WDRESET** bits are set they can not be cleared by software. Both flags are cleared by an external reset or a Watchdog timer reset.

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

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

In all power modes except Deep power-down mode, a Watchdog reset or interrupt can occur when the watchdog is running and has an operating clock source. The watchdog oscillator can be configured to keep running in Sleep, Deep-sleep modes, and Power-down modes.

If a watchdog interrupt occurs in Sleep, Deep-sleep mode, or Power-down mode, and the WWDT interrupt is enabled in the NVIC, the device will wake up. Note that in Deep-sleep and Power-down modes, the WWDT interrupt must be enabled in the STARTERP0 register in addition to the NVIC.

See the following registers:

Table 76 “Start logic 0 wake-up enable register 0 (STARTERP0, address 0x4007 4218) bit description”

Table 265. Watchdog operating modes selection

WDEN	WDRESET	Mode of Operation
0	X (0 or 1)	Debug/Operate without the Watchdog running.
1	0	Watchdog interrupt mode: the watchdog warning interrupt will be generated but watchdog reset will not. When this mode is selected, the watchdog counter reaching the value specified by WDWARNINT will set the WDINT flag and the Watchdog interrupt request will be generated.
1	1	Watchdog reset mode: both the watchdog interrupt and watchdog reset are enabled. When this mode is selected, the watchdog counter reaching the value specified by WDWARNINT will set the WDINT flag and the Watchdog interrupt request will be generated, and the watchdog counter reaching zero will reset the microcontroller. A watchdog feed prior to reaching the value of WDWINDOW will also cause a watchdog reset.

17.6.2 Watchdog Timer Constant register

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

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

Table 266. Watchdog Timer Constant register (TC, 0x4002 C004) bit description

Bit	Symbol	Description	Reset Value
23:0	COUNT	Watchdog time-out value.	0x00 00FF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.6.3 Watchdog Feed register

Writing 0xAA followed by 0x55 to this register will reload the Watchdog timer with the WDTC value. This operation will also start the Watchdog if it is enabled via the WDMOD register. Setting the WDEN bit in the WDMOD register is not sufficient to enable the Watchdog. A valid feed sequence must be completed after setting WDEN before the Watchdog is capable of generating a reset. Until then, the Watchdog will ignore feed errors.

After writing 0xAA to WDFEED, access to any Watchdog register other than writing 0x55 to WDFEED causes an immediate reset/interrupt when the Watchdog is enabled, and sets the WDTOF flag. The reset will be generated during the second PCLK following an incorrect access to a Watchdog register during a feed sequence.

It is good practice to disable interrupts around a feed sequence, if the application is such that an 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 267. Watchdog Feed register (FEED, 0x4002 C008) bit description

Bit	Symbol	Description	Reset Value
7:0	FEED	Feed value should be 0xAA followed by 0x55.	NA
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.6.4 Watchdog Timer Value register

The WDTV register is used to read the current value of Watchdog timer counter.

When reading the value of the 24-bit counter, the lock and synchronization procedure takes up to 6 WDCLK cycles plus 6 PCLK cycles, so the value of WDTV is older than the actual value of the timer when it's being read by the CPU.

Table 268. Watchdog Timer Value register (TV, 0x4002 C00C) bit description

Bit	Symbol	Description	Reset Value
23:0	COUNT	Counter timer value.	0x00 00FF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.6.5 Watchdog Timer Warning Interrupt register

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter matches the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

A match of the watchdog timer counter to WARNINT occurs when the bottom 10 bits of the counter have the same value as the 10 bits of WARNINT, and the remaining upper bits of the counter are all 0. This gives a maximum time of 1,023 watchdog timer counts (4,096 watchdog clocks) for the interrupt to occur prior to a watchdog event. If WARNINT is 0, the interrupt will occur at the same time as the watchdog event.

Table 269. Watchdog Timer Warning Interrupt register (WARNINT, 0x4002 C014) bit description

Bit	Symbol	Description	Reset Value
9:0	WARNINT	Watchdog warning interrupt compare value.	0
31:10	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.6.6 Watchdog Timer Window register

The WINDOW 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 WINDOW, a watchdog event will occur.

WINDOW resets to the maximum possible WDTV value, so windowing is not in effect.

Table 270. Watchdog Timer Window register (WINDOW, 0x4002 C018) bit description

Bit	Symbol	Description	Reset Value
23:0	WINDOW	Watchdog window value.	0xFF FFFF
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

17.7 Functional description

The following figures illustrate several aspects of Watchdog Timer operation.

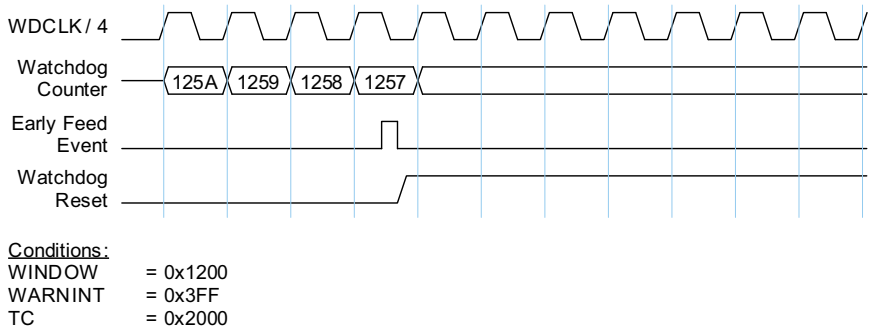


Fig 47. Early watchdog feed with windowed mode enabled

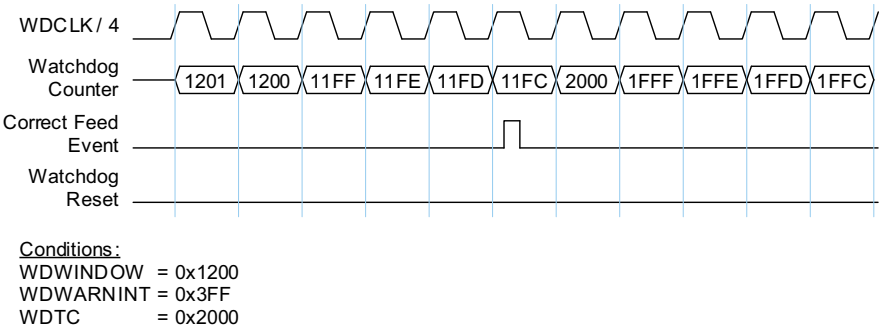


Fig 48. Correct watchdog feed with windowed mode enabled

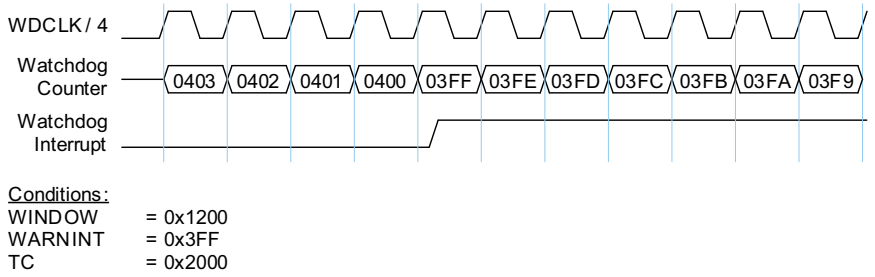


Fig 49. Watchdog warning interrupt

18.1 How to read this chapter

18.2 Features

- The RTC resides in a separate always-on voltage domain with battery back-up. The RTC uses an independent oscillator, also in the always-on voltage domain, which has the following clock outputs:
  - 32 kHz clock, selectable for system clock and CLKOUT pin.
  - 1 Hz clock for RTC timing.
  - 1 kHz clock for high-resolution RTC timing.
- 32-bit, 1 Hz RTC counter and associated match register for alarm generation.
- Separate 16-bit high-resolution/wake-up timer clocked at 1 kHz for 1 ms resolution with a more that one minute maximum time-out period.
- RTC alarm and high-resolution/wake-up timer time-out each generate independent interrupt requests. Either time-out can wake up the part from any of the low power modes, including Deep power-down.

18.3 Basic configuration

Configure the RTC as follows:

- Use the SYSAHBCLKCTRL0 register (Table 50) to enable the clock to the RTC register interface and peripheral clock.
- For RTC software reset use the RTC CTRL register. See Table 272.
- The RTC provides two interrupt lines to the NVIC:
  a. Interrupt raised on a match of the RTC 1 Hz counter connected to NVIC slot #45 (RTC_ALARM).
  b. Interrupt raised on a match of the RTC 1 kHz counter connected to NVIC slot #46 (RTC_WAKE).
- To enable the RTC interrupts for waking up from Deep-sleep and Power-down modes, enable the interrupts in the STARTLOGIC1 register (Table 77) and the NVIC.
- To enable the RTC interrupts for waking up from Deep power-down, enable the appropriate RTC clock and wake-up in the RTC CTRL register (Table 272).
- The RTC has no external pins.
- The RTC oscillator is always running, and therefore the 32 kHz output is always available to be enabled for syscon clock generation (see Table 67). Once enabled, the 32 kHz clock can be selected for the system clock or be observed through the CLKOUT pin. The 1 Hz output is enabled in the RTC CTRL register (RTC_EN bit). Once the 1 Hz output is enabled, you can enable the 1 KHz output for the high-resolution wake-up timer.

- Enable the RTC oscillator that provides the RTC's 1 Hz and 1 kHz clocks in the syscon block. See Table 67.

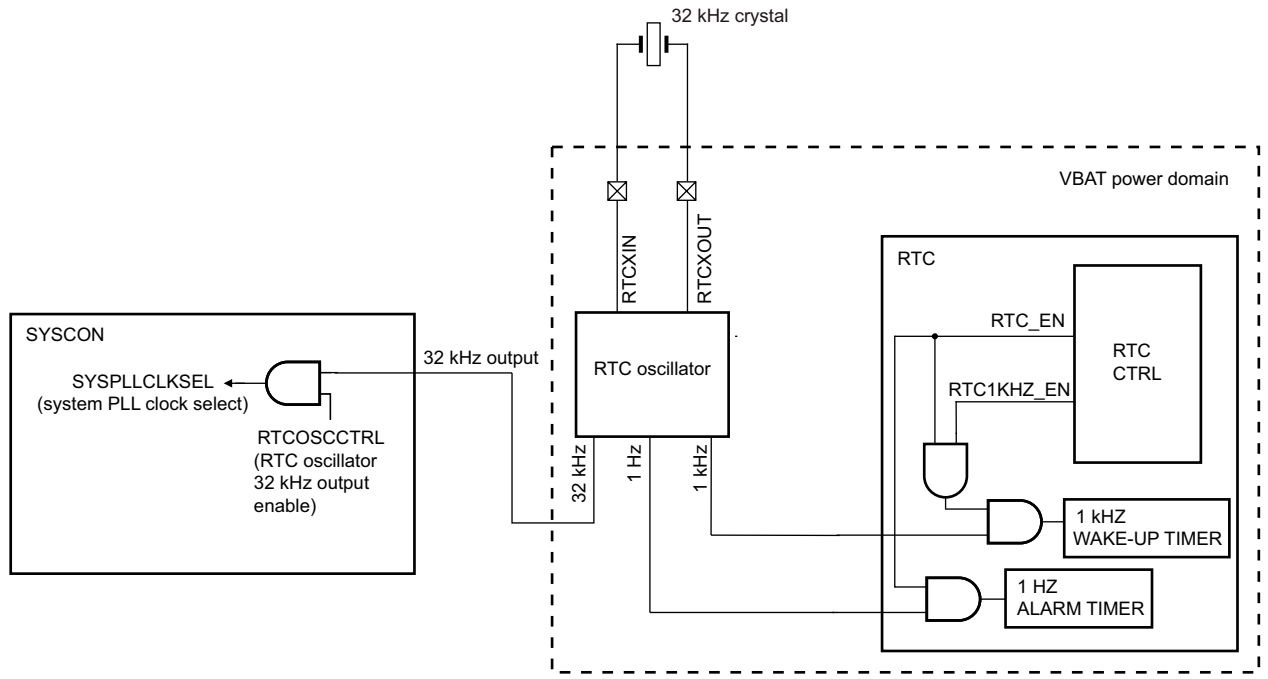


Fig 50. RTC clocking

18.3.1 RTC timers

The RTC contains two timers:

1. The main RTC timer. This 32-bit timer uses a 1 Hz clock and is intended to run continuously as a real-time clock. When the timer value reaches a match value, an interrupt is raised. The alarm interrupt can also wake up the part from any low power mode if enabled.
2. The high-resolution/wake-up timer. This 16-bit timer uses a 1 kHz clock and operates as a one-shot down timer. Once the timer is loaded, it starts counting down to 0 at which point an interrupt is raised. The interrupt can wake up the part from any low power mode if enabled. This timer is intended to be used for timed wake-up from Deep-sleep, power-down, or Deep power-down modes. The high-resolution wake-up timer can be disabled to conserve power if not used.

18.4 General description

18.4.1 Real-time clock

The real-time clock is a 32-bit up-counter which can be cleared or initialized by software. Once enabled, it counts continuously at a 1 Hz clock rate as long as the RTC module remains powered and enabled.

The main purpose of the RTC is to count seconds and generate an alarm interrupt to the processor whenever the counter value equals the value programmed into the associated 32-bit match register.

If the part is in one of the reduced-power modes (deep-sleep, power-down, deep power-down) an RTC alarm interrupt can also wake up the part to exit the power mode and begin normal operation.

18.4.2 High-resolution/wake-up timer

The time interval required for many applications, including waking the part up from a low-power mode, will often demand a greater degree of resolution than the one-second minimum interval afforded by the main RTC counter. For these applications, a higher frequency secondary timer has been provided.

This secondary timer is an independent, stand-alone wake-up or general-purpose timer for timing intervals of up to 64 seconds with approximately one millisecond of resolution.

The High-Resolution/Wake-up Timer is a 16-bit down counter which is clocked at a 1 kHz rate when it is enabled. Writing any non-zero value to this timer will automatically enable the counter and launch a countdown sequence. When the counter is being used as a wake-up timer, this write can occur just prior to entering a reduced power mode.

When a starting count value is loaded, the High-Resolution/Wake-up Timer will turn on, count from the pre-loaded value down to zero, generate an interrupt and/or a wake-up command, and then turn itself off until re-launched by a subsequent software write.

18.4.3 RTC power domain

The RTC module and the 1 Hz/1 kHz clock that drives it, reside in the battery backup always-on voltage domain. As a result, the RTC will continue operating in deep power-down mode when power is removed from the rest of the part. The RTC will also continue to operate in the event that power fails, until the backup battery runs out.

18.5 Register description

Reset Values pertain to initial power-up of the always-on power domain or when an RTC software reset is applied (except where noted). This block is not initialized by a standard POR, pad reset, or by any other system reset.

Table 271. Register overview: RTC (base address 0x4002 8000)

Name	Access	Offset	Description	Reset value	Reference
CTRL	R/W	0x000	RTC control register	0xF	Table 272
MATCH	R/W	0x004	RTC match register	0xFFFF	Table 273
COUNT	R/W	0x008	RTC counter register	0	Table 274
WAKE	R/W	0x00C	RTC high-resolution/wake-up timer control register	0	Table 275

18.5.1 RTC CTRL register

This register controls which clock the RTC uses (1 kHz or 1 Hz) and enables the two RTC interrupts to wake up the part from Deep power-down. To wake up the part from Deep-sleep or Power-down modes, enable the RTC interrupts in the system control block STARTLOGIC1 register.

Table 272. RTC control register (CTRL, address 0x4002 8000) bit description

Bit	Symbol	Value	Description	Reset value
0	SWRESET		Software reset control	1
		0	Not in reset. The RTC is not held in reset. This bit must be cleared prior to configuring or initiating any operation of the RTC.	
		1	In reset. The RTC is held in reset. All register bits within the RTC will be forced to their reset value except the OFD bit. This bit must be cleared before writing to any register in the RTC - including writes to set any of the other bits within this register. Do not attempt to write to any bits of this register at the same time that the reset bit is being cleared. **Remark:** This bit may also serve as a Power Fail Detect flag for the always-on voltage domain.	
1	OFD		Oscillator fail detect status. There is a delay before this bit is updated.	1
		0	Run. The RTC oscillator is running properly. Writing a 0 has no effect.	
		1	Fail. RTC oscillator fail detected. Clear this flag after the following power-up. Writing a 1 clears this bit.	
2	ALARM1HZ		RTC 1 Hz timer alarm flag status.	1
		0	No match. No match has occurred on the 1 Hz RTC timer. Writing a 0 has no effect.	
		1	Match. A match condition has occurred on the 1 Hz RTC timer. This flag generates an RTC alarm interrupt request RTC_ALARM which can also wake up the part from any low power mode. Writing a 1 clears this bit.	
3	WAKE1KHZ		RTC 1 kHz timer wake-up flag status.	1
		0	Run. The RTC 1 kHz timer is running. Writing a 0 has no effect.	
		1	Time-out. The 1 kHz high-resolution/wake-up timer has timed out. This flag generates an RTC wake-up interrupt request RTC-WAKE which can also wake up the part from any low power mode. Writing a 1 clears this bit.	
4	ALARMDPD_EN		RTC 1 Hz timer alarm enable for Deep power-down.	0
		0	Disable. A match on the 1 Hz RTC timer will not bring the part out of Deep power-down mode.	
		1	Enable. A match on the 1 Hz RTC timer bring the part out of Deep power-down mode.	
5	WAKEDPD_EN		RTC 1 kHz timer wake-up enable for Deep power-down.	0
		0	Disable. A match on the 1 kHz RTC timer will not bring the part out of Deep power-down mode.	
		1	Enable. A match on the 1 kHz RTC timer bring the part out of Deep power-down mode.	

Table 272. RTC control register (CTRL, address 0x4002 8000) bit description

Bit	Symbol	Value	Description	Reset value
6	RTC1KHZ_EN		RTC 1 kHz clock enable. This bit can be set to 0 to conserve power if the 1 kHz timer is not used. This bit has no effect when the RTC is disabled (bit 7 of this register is 0).	0
		0	Disable. A match on the 1 kHz RTC timer will not bring the part out of Deep power-down mode.	
		1	Enable. The 1 kHz RTC timer is enabled.	
7	RTC_EN		RTC enable.	0
		0	Disable. The RTC 1 Hz and 1 kHz clocks are shut down and the RTC operation is disabled. This bit should be 0 when writing to load a value in the RTC counter register.	
		1	Enable. The 1 Hz RTC clock is running and RTC operation is enabled. You must set this bit to initiate operation of the RTC. The first clock to the RTC counter occurs 1 s after this bit is set. To also enable the high-resolution, 1 kHz clock, set bit 6 in this register.	
31:8	-		Reserved	0

18.5.2 RTC match register

Table 273. RTC match register (MATCH, address 0x4002 8004) bit description

Bit	Symbol	Description	Reset value
31:0	MATVAL	Contains the match value against which the 1 Hz RTC timer will be compared to generate set the alarm flag RTC_ALARM and generate an alarm interrupt/wake-up if enabled.	0xFFFF

18.5.3 RTC counter register

Table 274. RTC counter register (COUNT, address 0x4002 8008) bit description

Bit	Symbol	Description	Reset value
31:0	VAL	A read reflects the current value of the main, 1 Hz RTC timer. A write loads a new initial value into the timer. The RTC counter will count up continuously at a 1 Hz rate once the RTC Software Reset is removed (by clearing bit 0 of the CTRL register). **Remark:** Only write to this register when the RTC_EN bit in the RTC CTRL Register is 0. The counter increments one second after the RTC_EN bit is set.	0

18.5.4 RTC high-resolution/wake-up register

Table 275. RTC high-resolution/wake-up register (WAKE, address 0x4002 800C) bit description

Bit	Symbol	Description	Reset value
15:0	VAL	A read reflects the current value of the high-resolution/wake-up timer. A write pre-loads a start count value into the wake-up timer and initializes a count-down sequence. Do not write to this register while counting is in progress.	0
31:16	-	Reserved.	

19.1 How to read this chapter

The MRT is available on all LPC15xx parts.

19.2 Features

- 24-bit interrupt timer
- Four channels independently counting down from individually set values
- Repeat and one-shot interrupt modes

19.3 Basic configuration

Configure the MRT using the following registers:

- In the SYSAHBCLKCTRL1 register, set bit 0 (Table 51) to enable the clock to the register interface.
- Clear the MRT reset using the PRESETCTRL1 register (Table 36).
- The global MRT interrupt is connected to interrupt #20 in the NVIC.

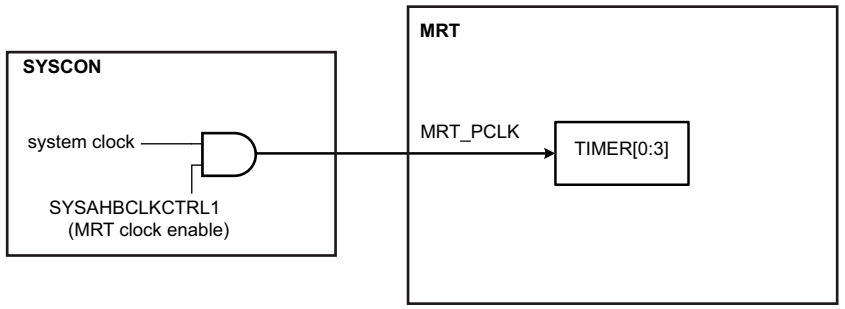


Fig 51. MRT clocking

19.4 Pin description

The MRT has no configurable pins.

19.5 General description

The Multi-Rate Timer (MRT) provides a repetitive interrupt timer with four channels. Each channel can be programmed with an independent time interval.

Each channel operates independently from the other channels in one of the following modes:

- Repeat interrupt mode. See Section 19.5.1.

- One-shot interrupt mode. See Section 19.5.2.

The modes for each timer are set in the timer's control register. See Table 279.

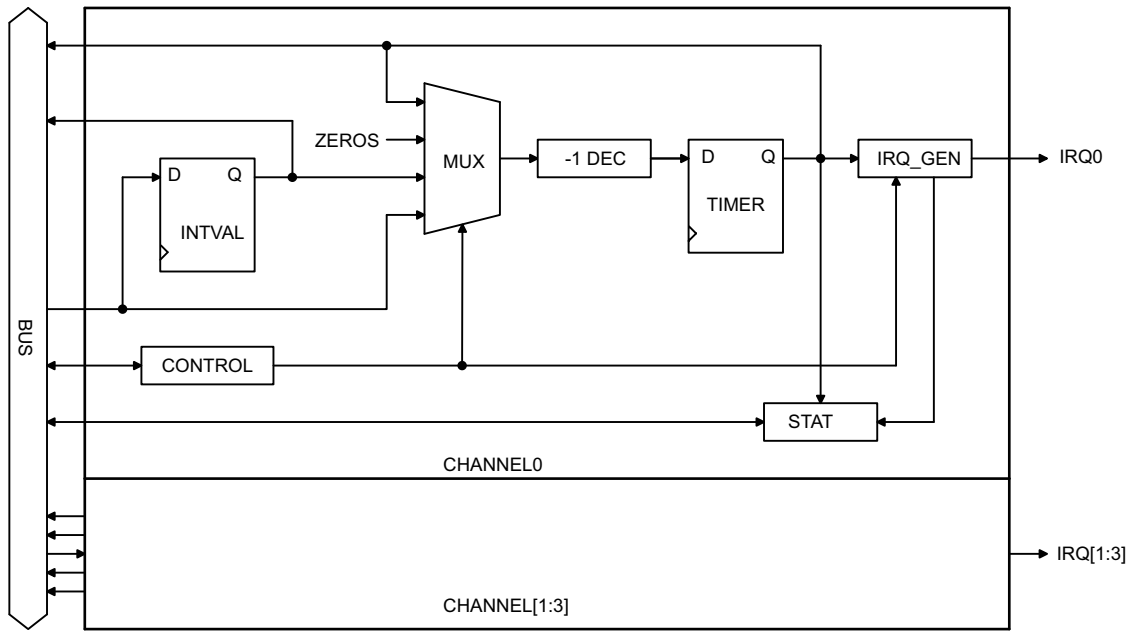


Fig 52. MRT block diagram

19.5.1 Repeat interrupt mode

The repeat interrupt mode generates repeated interrupts after a selected time interval. This mode can be used for software-based PWM or PPM applications.

When the timer n is in idle state, writing a non-zero value IVALUE to the INTVALn register immediately loads the time interval value IVALUE - 1, and the timer begins to count down from this value. When the timer reaches zero, an interrupt is generated, the value in the INTVALn register IVALUE - 1 is reloaded automatically, and the timer starts to count down again.

While the timer is running in repeat interrupt mode, you can perform the following actions:

- Change the interval value on the next timer cycle by writing a new value (>0) to the INTVALn register and setting the LOAD bit to 0. An interrupt is generated when the timer reaches zero. On the next cycle, the timer counts down from the new value.
- Change the interval value on-the-fly immediately by writing a new value (>0) to the INTVALn register and setting the LOAD bit to 1. The timer immediately starts to count down from the new timer interval value. An interrupt is generated when the timer reaches 0.
- Stop the timer at the end of time interval by writing a 0 to the INTVALn register and setting the LOAD bit to 0. An interrupt is generated when the timer reaches zero.
- Stop the timer immediately by writing a 0 to the INTVALn register and setting the LOAD bit to 1. No interrupt is generated when the INTVALn register is written.

19.5.2 One-shot interrupt mode

The one-shot interrupt generates one interrupt after a one-time count. With this mode, you can generate a single interrupt at any point. This mode can be used to introduce a specific delay in a software task.

When the timer is in the idle state, writing a non-zero value IVALUE to the INTVALn register immediately loads the time interval value IVALUE - 1, and the timer starts to count down. When the timer reaches 0, an interrupt is generated and the timer stops and enters the idle state.

While the timer is running in the one-shot interrupt mode, you can perform the following actions:

- Update the INTVALn register with a new time interval value (>0) and set the LOAD bit to 1. The timer immediately reloads the new time interval, and starts counting down from the new value. No interrupt is generated when the TIME_INTVALn register is updated.
- Write a 0 to the INTVALn register and set the LOAD bit to 1. The timer immediately stops counting and moves to the idle state. No interrupt is generated when the INTVALn register is updated.

19.6 Register description

The reset values shown in Table 276 are POR reset values.

Table 276. Register overview: MRT (base address 0x400A 0000)

Name	Access	Address offset	Description	Reset value	Reference
INTVAL0	R/W	0x0	MRT0 Time interval value register. This value is loaded into the TIMER0 register.	0	Table 277
TIMER0	R	0x4	MRT0 Timer register. This register reads the value of the down-counter.	0xFF FFFF	Table 278
CTRL0	R/W	0x8	MRT0 Control register. This register controls the MRT0 modes.	0	Table 279
STAT0	R/W	0xC	MRT0 Status register.	0	Table 280
INTVAL1	R/W	0x10	MRT1 Time interval value register. This value is loaded into the TIMER1 register.	0	Table 277
TIMER1	R/W	0x14	MRT1 Timer register. This register reads the value of the down-counter.	0xFF FFFF	Table 278
CTRL1	R/W	0x18	MRT1 Control register. This register controls the MRT1 modes.	0	Table 279
STAT1	R/W	0x1C	MRT1 Status register.	0	Table 280
INTVAL2	R/W	0x20	MRT2 Time interval value register. This value is loaded into the TIMER2 register.	0	Table 277
TIMER2	R/W	0x24	MRT2 Timer register. This register reads the value of the down-counter.	0xFF FFFF	Table 278
CTRL2	R/W	0x28	MRT2 Control register. This register controls the MRT2 modes.	0	Table 279
STAT2	R/W	0x2C	MRT2 Status register.	0	Table 280

Table 276. Register overview: MRT (base address 0x400A 0000)

Name	Access	Address offset	Description	Reset value	Reference
INTVAL3	R/W	0x30	MRT3 Time interval value register. This value is loaded into the TIMER3 register.	0	Table 277
TIMER3	R/W	0x34	MRT3 Timer register. This register reads the value of the down-counter.	0xFF FFFF	Table 278
CTRL3	R/W	0x38	MRT3 Control register. This register controls the MRT modes.	0	Table 279
STAT3	R/W	0x3C	MRT3 Status register.	0	Table 280
IDLE_CH	R	0xF4	Idle channel register. This register returns the number of the first idle channel.	0	Table 281
IRQ_FLAG	R/W	0xF8	Global interrupt flag register	0	Table 282

19.6.1 Time interval register

This register contains the MRT load value and controls how the timer is reloaded. The load value is IVALUE -1.

Table 277. Time interval register (INTVAL[0:3], address 0x400A 0000 (INTVAL0) to 0x400A 0030 (INTVAL3)) bit description

Bit	Symbol	Value	Description	Reset value
23:0	IVALUE		Time interval load value. This value is loaded into the TIMERn register and the MRT channel n starts counting down from IVALUE -1. If the timer is idle, writing a non-zero value to this bit field starts the timer immediately. If the timer is running, writing a zero to this bit field does the following: • If LOAD = 1, the timer stops immediately. • If LOAD = 0, the timer stops at the end of the time interval.	0
30:24	-		Reserved.	-
31	LOAD		Determines how the timer interval value IVALUE -1 is loaded into the TIMERn register. This bit is write-only. Reading this bit always returns 0.	0
		0	No force load. The load from the INTVALn register to the TIMERn register is processed at the end of the time interval if the repeat mode is selected.	
		1	Force load. The INTVALn interval value IVALUE -1 is immediately loaded into the TIMERn register while TIMERn is running.	

19.6.2 Timer register

The timer register holds the current timer value. This register is read-only.

Table 278. Timer register (TIMER[0:3], address 0x400A 0004 (TIMER0) to 0x400A 0034 (TIMER3)) bit description

Bit	Symbol	Description	Reset value
23:0	VALUE	Holds the current timer value of the down-counter. The initial value of the TIMERn register is loaded as IVALUE - 1 from the INTVALn register either at the end of the time interval or immediately in the following cases: INTVALn register is updated in the idle state. INTVALn register is updated with LOAD = 1. When the timer is in idle state, reading this bit fields returns -1 (0x00FF FFFF).	0x00FF FFFF
31:24	-	Reserved.	0

19.6.3 Control register

The control register configures the mode for each MRT and enables the interrupt.

Table 279. Control register (CTRL[0:3], address 0x400A 0008 (CTRL0) to 0x400A 0038 (CTRL3)) bit description

Bit	Symbol	Value	Description	Reset value
0	INTEN		Enable the TIMERn interrupt.	0
		0	Disable.	
		1	Enable.	
2:1	MODE		Selects timer mode.	0
		0x0	Repeat interrupt mode.	
		0x1	One-shot interrupt mode.	
		0x2	Reserved.	
		0x3	Reserved.	
31:3	-		Reserved.	0

19.6.4 Status register

This register indicates the status of each MRT.

Table 280. Status register (STAT[0:3], address 0x400A 000C (STAT0) to 0x400A 003C (STAT3)) bit description

Bit	Symbol	Value	Description	Reset value
0	INTFLAG		Monitors the interrupt flag.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMERn has reached the end of the time interval. If the INTEN bit in the CONTROLn is also set to 1, the interrupt for timer channel n and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
1	RUN		Indicates the state of TIMERn. This bit is read-only.	0
		0	Idle state. TIMERn is stopped.	
		1	Running. TIMERn is running.	
31:2	-		Reserved.	0

19.6.5 Idle channel register

The idle channel register returns the lowest idle channel number. The channel is considered idle when both flags in the STATUS register (RUN and INTFLAG) are zero.

In an application with multiple timers running independently, you can calculate the register offset of the next idle timer by reading the idle channel number in this register. The idle channel register allows you set up the next idle timer without checking the idle state of each timer.

Table 281. Idle channel register (IDLE_CH, address 0x400A 00F4) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved.	0
7:4	CHAN	Idle channel. Reading the CHAN bits, returns the lowest idle timer channel. If all timer channels are running, CHAN = 0xF. To make sure that all outstanding interrupt requests have been serviced, a channel is considered idle only when both the corresponding RUN bit and the interrupt flag are zero in the STATUS register.	0
31:8	-	Reserved.	0

19.6.6 Global interrupt flag register

The global interrupt register combines the interrupt flags from the individual timer channels in one register. Setting and clearing each flag behaves in the same way as setting and clearing the INTFLAG bit in each of the STATUSn registers.

Table 282. Global interrupt flag register (IRQ_FLAG, address 0x400A 00F8) bit description

Bit	Symbol	Value	Description	Reset value
0	GFLAG0		Monitors the interrupt flag of TIMER0.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMER0 has reached the end of the time interval. If the INTEN bit in the CONTROL0 register is also set to 1, the interrupt for timer channel 0 and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
1	GFLAG1		Monitors the interrupt flag of TIMER1.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMER1 has reached the end of the time interval. If the INTEN bit in the CONTROL1 register is also set to 1, the interrupt for timer channel 1 and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
2	GFLAG2		Monitors the interrupt flag of TIMER2.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMER2 has reached the end of the time interval. If the INTEN bit in the CONTROL2 register is also set to 1, the interrupt for timer channel 2 and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
3	GFLAG3		Monitors the interrupt flag of TIMER3.	0
		0	No pending interrupt. Writing a zero is equivalent to no operation.	
		1	Pending interrupt. The interrupt is pending because TIMER3 has reached the end of the time interval. If the INTEN bit in the CONTROL3 register is also set to 1, the interrupt for timer channel 3 and the global interrupt are raised. Writing a 1 to this bit clears the interrupt request.	
31:4	-		Reserved.	0

20.1 How to read this chapter

20.2 Features

- 48-bit counter running from the main clock. Counter can be free-running or be reset by a generated interrupt.
- 48-bit compare value.
- 48-bit compare mask. An interrupt is generated when the counter value equals the compare value, after masking. This allows for combinations not possible with a simple compare.

20.3 Basic configuration

The RI timer is configured through the following registers:

- Power to the register interface (RI timer clock): In the SYSAHBCLKCTRL1 register, set bit 1 in Table 51.

20.4 General description

The Repetitive Interrupt Timer (RIT) provides a versatile means of generating interrupts at specified time intervals, without using a standard timer. It is intended for repeating interrupts that aren't related to Operating System interrupts. The RIT could also be used as an alternative to the Cortex-M3 System Tick Timer if there are different system requirements.

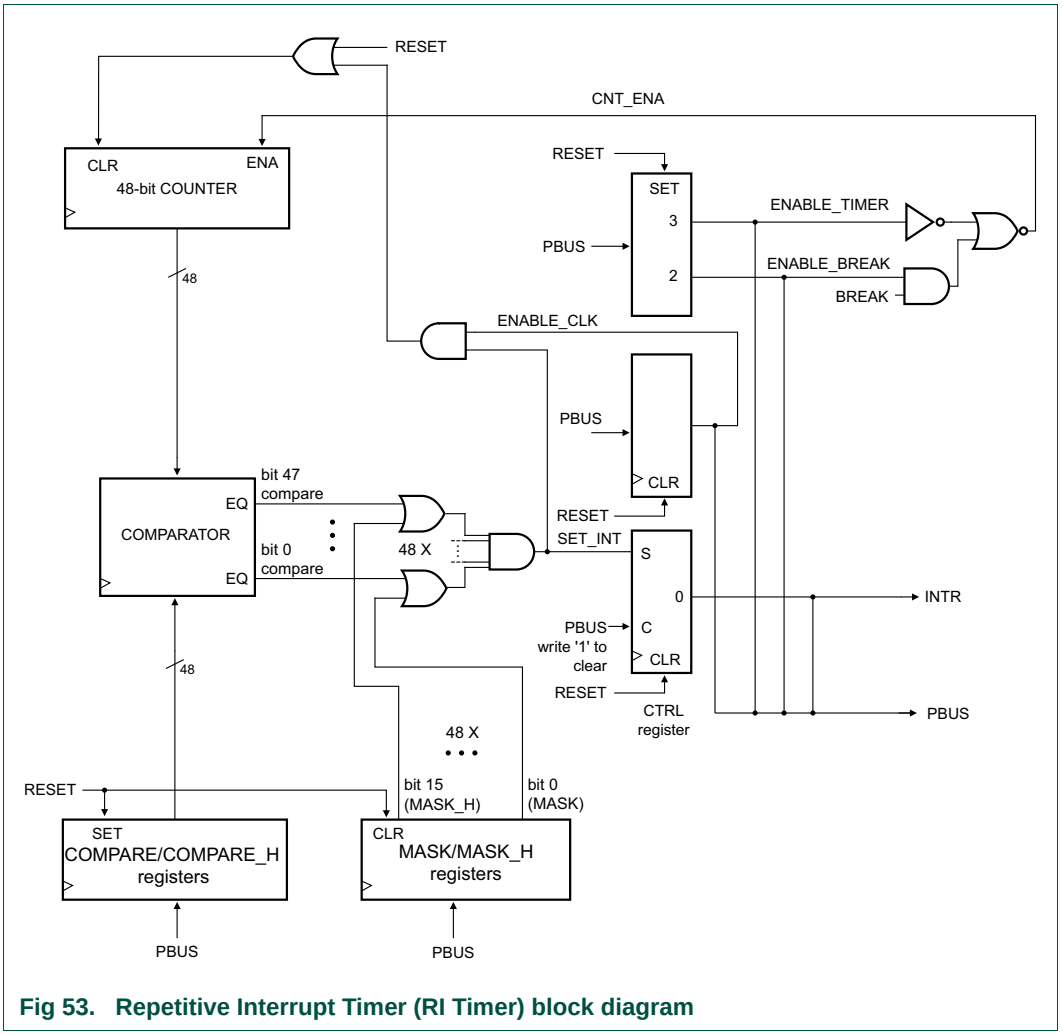


Fig 53. Repetitive Interrupt Timer (RI Timer) block diagram

20.5 Register description

Table 283. Register overview: Repetitive Interrupt Timer (RIT) (base address 0x400B 4000)

Name	Access	Address	Description	Reset value[1]	Reference
COMPVAL	R/W	0x000	Compare value LSB register. Holds the 32 LSBs of the compare value.	0xFFFF FFFF	Table 284
MASK	R/W	0x004	Mask LSB register. This register holds the 32 LSB s of the mask value. A 1 written to any bit will force the compare to be true for the corresponding bit of the counter and compare register.	0	Table 285
CTRL	R/W	0x008	Control register.	0xC	Table 286
COUNTER	R/W	0x00C	Counter LSB register. 32 LSBs of the counter.	0	Table 287

Table 283. Register overview: Repetitive Interrupt Timer (RIT) (base address 0x400B 4000)

Name	Access	Address	Description	Reset value[1]	Reference
COMPVAL_H	R/W	0x010	Compare value MSB register. Holds the 16 MSBs of the compare value.	0x0000 FFFF	Table 284
MASK_H	R/W	0x014	Mask MSB register. This register holds the 16 MSBs of the mask value. A '1' written to any bit will force a compare on the corresponding bit of the counter and compare register.	0	Table 285
COUNTER_H	R/W	0x01C	Counter MSB register. 16 MSBs of the counter.	0	Table 287

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

20.5.1 RI Compare Value LSB register

Table 284. RI Compare Value LSB register (COMPVAL, address 0x400B 4000) bit description

Bit	Symbol	Description	Reset value
31:0	RICOMP	Compare register. Holds the 32 LSBs of the compare value which is compared to the counter.	0xFFFF FFFF

20.5.2 RI Mask LSB register

Table 285. RI Mask LSB register (MASK, address 0x400B 4004) bit description

Bit	Symbol	Description	Reset value
31:0	RIMASK	Mask register. This register holds the 32 LSBs of the mask value. A one written to any bit overrides the result of the comparison for the corresponding bit of the counter and compare register (causes the comparison of the register bits to be always true).	0

20.5.3 RI Control register

Table 286. RI Control register (CTRL, address 0x400B 4008) bit description

Bit	Symbol	Value	Description	Reset value
0	RITINT		Interrupt flag	0
		1	This bit is set to 1 by hardware whenever the counter value equals the masked compare value specified by the contents of RICOMPVAL and RIMASK registers. Writing a 1 to this bit will clear it to 0. Writing a 0 has no effect.	
		0	The counter value does not equal the masked compare value.	
1	RITENCLR		Timer enable clear	
		1	The timer will be cleared to 0 whenever the counter value equals the masked compare value specified by the contents of COMPVAL/COMPVAL_H and MASK/MASK_H registers. This will occur on the same clock that sets the interrupt flag.	0
		0	The timer will not be cleared to 0.	

Table 286. RI Control register (CTRL, address 0x400B 4008) bit description

Bit	Symbol	Value	Description	Reset value
2	RITENBR		Timer enable for debug	1
		1	The timer is halted when the processor is halted for debugging.	
		0	Debug has no effect on the timer operation.	
3	RITEN		Timer enable.	1
		1	Timer enabled. **Remark:** This can be overruled by a debug halt if enabled in bit 2.	
		0	Timer disabled.	
31:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

20.5.4 RI Counter LSB register

Table 287. RI Counter register (COUNTER, address 0x400B 400C) bit description

Bit	Symbol	Description	Reset value
31:0	RICOUNTER	32 LSBs of the up counter. Counts continuously unless RITEN bit in CTRL register is cleared or debug mode is entered (if enabled by the RITNEBR bit in RICTRL). Can be loaded to any value in software.	0

20.5.5 RI Compare Value MSB register

Table 288. RI Compare Value MSB register (COMPVAL_H, address 0x400B 4010) bit description

Bit	Symbol	Description	Reset value
15:0	RICOMP	Compare value MSB register. Holds the 16 MSBs of the compare value which is compared to the counter.	0x0000 FFFF
31:16	-	Reserved.	-

20.5.6 RI Mask MSB register

Table 289. RI Mask MSB register (MASK_H, address 0x400B 4014) bit description

Bit	Symbol	Description	Reset value
15:0	RIMASK	Mask register. This register holds the 16 MSBs of the mask value. A one written to any bit overrides the result of the comparison for the corresponding bit of the counter and compare register (causes the comparison of the register bits to be always true).	0
31:16	-	Reserved.	-

20.5.7 RI Counter MSB register

Table 290. RI Counter MSB register (COUNTER_H, address 0x400B 401C) bit description

Bit	Symbol	Description	Reset value
15:0	RICOUNTER	16 LSBs of the up counter. Counts continuously unless RITEN bit in RICTRL register is cleared or debug mode is entered (if enabled by the RITNEBR bit in RICTRL). Can be loaded to any value in software.	0
31:16	-	Reserved.	-

20.6 RI timer operation

Following reset, the counter begins counting up from 0. (The RIT bit must be set in the SYSAHBCLKCTRL1 register to enable the clock to the RIT.) Whenever the counter value equals the 48-bit value programmed into the COMPVAL and COMPVAL_H registers, the interrupt flag will be set. Any bit or combination of bits can be removed from this comparison (i.e. forced to compare) by writing a 1 to the corresponding bit(s) in the MASK and MASK_H registers. If the RITENCLR bit is low (default state), a valid comparison ONLY causes the interrupt flag to be set. It has no effect on the count sequence. Counting continues as usual. When the counter reaches 0xFFFF FFFF FFFF it rolls-over to 0 on the next clock and continues counting. If the RITENCLR bit is set to 1 a valid comparison will also cause the counter to be reset to zero. Counting will resume from there on the next clock edge.

Counting can be halted in software by writing a ‘0’ to the RITEN bit. Counting will also be halted when the processor is halted for debugging provided the RITENBR bit is set. Both the RITEN and RITENBR bits are set on reset.

The interrupt flag can be cleared in software by writing a 1 to the RITINT bit.

Software must stop the counter before reloading it with a new value.

The counter (COUNTER/COUNTER_H), COMPVAL/COMPVAL_H registers, MASK/MASK_H registers, and the CTRL register can all be read by software at any time.

21.1 How to read this chapter

The system tick timer (SysTick timer) is part of the ARM Cortex-M3 core and is identical for all LPC15xx parts.

21.2 Basic configuration

The system tick timer is configured using the following registers:

1. Pins: The system tick timer uses no external pins.
2. Power: The system tick timer is enabled through the SysTick control register). The system tick timer clock is fixed to half the frequency of the system clock.
3. Enable the clock source for the SysTick timer in the SYST_CSR register.

21.3 Features

- Simple 24-bit timer.
- Uses dedicated exception vector.
- Clocked internally by the system clock or the SYSTICKCLK.

21.4 General description

The block diagram of the SysTick timer is shown below in the Figure 54.

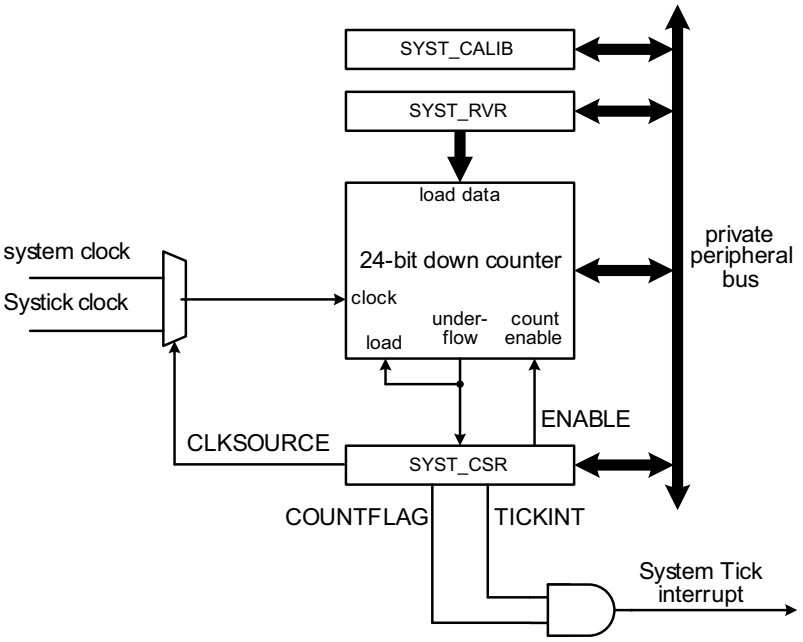


Fig 54. System tick timer block diagram

The SysTick timer is an integral part of the Cortex-M3. 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-M3, it facilitates porting of software by providing a standard timer that is available on Cortex-M3 based devices. The SysTick timer can be used for:

- An RTOS tick timer which fires at a programmable rate (for example 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the core clock.
- A simple counter. Software can use this to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNTFLAG bit-field in the control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

Refer to the *Cortex-M3 User Guide* for details.

21.5 Register description

The systick timer registers are located on the ARM Cortex-M3 private peripheral bus (see Figure 3), and are part of the ARM Cortex-M3 core peripherals. For details, see Ref. 1.

Table 291. Register overview: SysTick timer (base address 0xE000 E000)

Name	Access	Address offset	Description	Reset value[1]	Reference
SYST_CSR	R/W	0x010	System Timer Control and status register	0x000 0000	Table 292
SYST_RVR	R/W	0x014	System Timer Reload value register	0	Table 293
SYST_CVR	R/W	0x018	System Timer Current value register	0	Table 294
SYST_CALIB	R/W	0x01C	System Timer Calibration value register	0x0	Table 295

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

21.5.1 System Timer Control and status register

The SYST_CSR register contains control information for the SysTick timer and provides a status flag. This register is part of the ARM Cortex-M3 core system timer register block. For a detailed bit description of this register, see Ref. 1.

This register determines the clock source for the system tick timer.

Table 292. SysTick Timer Control and status register (SYST_CSR, 0xE000 E010) bit description

Bit	Symbol	Description	Reset value
0	ENABLE	System Tick counter enable. When 1, the counter is enabled. When 0, the counter is disabled.	0
1	TICKINT	System Tick interrupt enable. When 1, the System Tick interrupt is enabled. When 0, the System Tick interrupt is disabled. When enabled, the interrupt is generated when the System Tick counter counts down to 0.	0
2	CLKSOURCE	System Tick clock source selection. When 1, the system clock (CPU) clock is selected. When 0, the output clock from the system tick clock divider (SYSTICKDIV) is selected as the reference clock. In this case, the core clock must be at least 2.5 times faster than the reference clock otherwise the count values are unpredictable.	0
15:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
16	COUNTFLAG	Returns 1 if the SysTick timer counted to 0 since the last read of this register.	0
31:17	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

21.5.2 System Timer Reload value register

The SYST_RVR register is set to the value that will be loaded into the SysTick timer whenever it counts down to zero. This register is loaded by software as part of timer initialization. The SYST_CALIB register may be read and used as the value for SYST_RVR register if the CPU is running at the frequency intended for use with the SYST_CALIB value.

Table 293. System Timer Reload value register (SYST_RVR, 0xE000 E014) bit description

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

21.5.3 System Timer Current value register

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

Table 294. System Timer Current value register (SYST_CVR, 0xE000 E018) bit description

Bit	Symbol	Description	Reset value
23:0	CURRENT	Reading this register returns the current value of the System Tick counter. Writing any value clears the System Tick counter and the COUNTFLAG bit in SYST_CSR.	0
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

21.5.4 System Timer Calibration value register

The value of the SYST_CALIB register is driven by the value of the SYSTCKCAL register in the system configuration block (see Table 32).

Table 295. System Timer Calibration value register (SYST_CALIB, 0xE000 E01C) bit description

Bit	Symbol	Value	Description	Reset value
23:0	TENMS		Reload value set by the SYSCON block.	0x4
29:24	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	SKEW		Reload value set by the SYSCON block.	0
31	NOREF		Reload value set by the SYSCON block.	0

21.6 Functional description

The SysTick timer is a 24-bit timer that counts down to zero and generates an interrupt. The intent is to provide a fixed 10 millisecond time interval between interrupts. The SysTick timer is clocked from the CPU clock (the system clock, see Figure 3) or from the reference clock, which is fixed to half the frequency of the CPU clock. In order to generate recurring interrupts at a specific interval, the SYST_RVR register must be initialized with the correct value for the desired interval. A default value is provided in the SYST_CALIB register and may be changed by software.

21.7 Example timer calculations

To use the system tick timer, do the following:

1. Program the LOAD register with the reload value RELOAD to obtain the desired time interval.
2. Clear the VAL register by writing to it. This ensures that the timer will count from the LOAD value rather than an arbitrary value when the timer is enabled.

The following examples illustrate selecting SysTick timer reload values for different system configurations. All of the examples calculate an interrupt interval of 10 milliseconds, as the SysTick timer is intended to be used, and there are no rounding errors.

System clock = 72 MHz

Program the CTRL register with the value 0x7 which selects the system clock as the clock source and enables the SysTick timer and the SysTick timer interrupt.

RELOAD = (system clock frequency × 10 ms) −1 = (72 MHz × 10 ms) −1 = 720000 −1 = 719999 = 0x000AFC7F

System tick timer clock = 24 MHz

Program the CTRL register with the value 0x3 which selects the clock from the system tick clock divider (use DIV = 3) as the clock source and enables the SysTick timer and the SysTick timer interrupt.

RELOAD = (system tick timer clock frequency × 10 ms) −1 = (24 MHz × 10 ms) −1 = 240000 −1 = 239999 = 0x0003A97F

System clock = 12 MHz

Program the CTRL register with the value 0x7 which selects the system clock as the clock source and enables the SysTick timer and the SysTick timer interrupt.

In this case the system clock is derived from the IRC clock.

RELOAD = (system clock frequency × 10 ms) −1 = (12 MHz × 10 ms) −1 = 120000 −1 = 119999 = 0x0001D4BF

22.1 How to read this chapter

The quadrature encoder interface is available on all parts.

22.2 Features

- Tracks encoder position.
- Increments/ decrements depending on direction.
- Programmable for 2X or 4X position counting.
- Velocity capture using built-in timer.
- Velocity compare function with less than interrupt.
- Uses 32-bit registers for position and velocity.
- Three position compare registers with interrupts.
- Index counter for revolution counting.
- Index compare register with interrupts.
- Can combine index and position interrupts to produce an interrupt for whole and partial revolution displacement.
- Digital filter with programmable delays for encoder input signals.
- Can accept decoded signal inputs (clock and direction).

22.3 Basic configuration

- Use the SYSAHBCLKCTRL1 register (Table 51) to enable the clock to the QEI interface.
- Clear the peripheral reset for the entire comparator block using the PRESETCTRL1 register (Table 36).
- The QEI creates one interrupt connected to slot #44 in the NVIC.
- Use the switch matrix to assign the QEI inputs to pins. See Table 296.

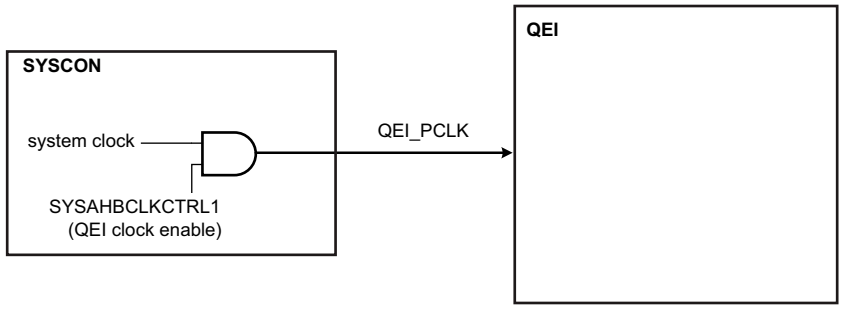


Fig 55. QEI timing

22.4 General description

A quadrature encoder, also known as a 2-channel incremental encoder, converts angular displacement into two pulse signals. By monitoring both the number of pulses and the relative phase of the two signals, you can track the position, direction of rotation, and velocity. In addition, a third channel, or index signal, can be used to reset the position counter. This quadrature encoder interface module decodes the digital pulses from a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.

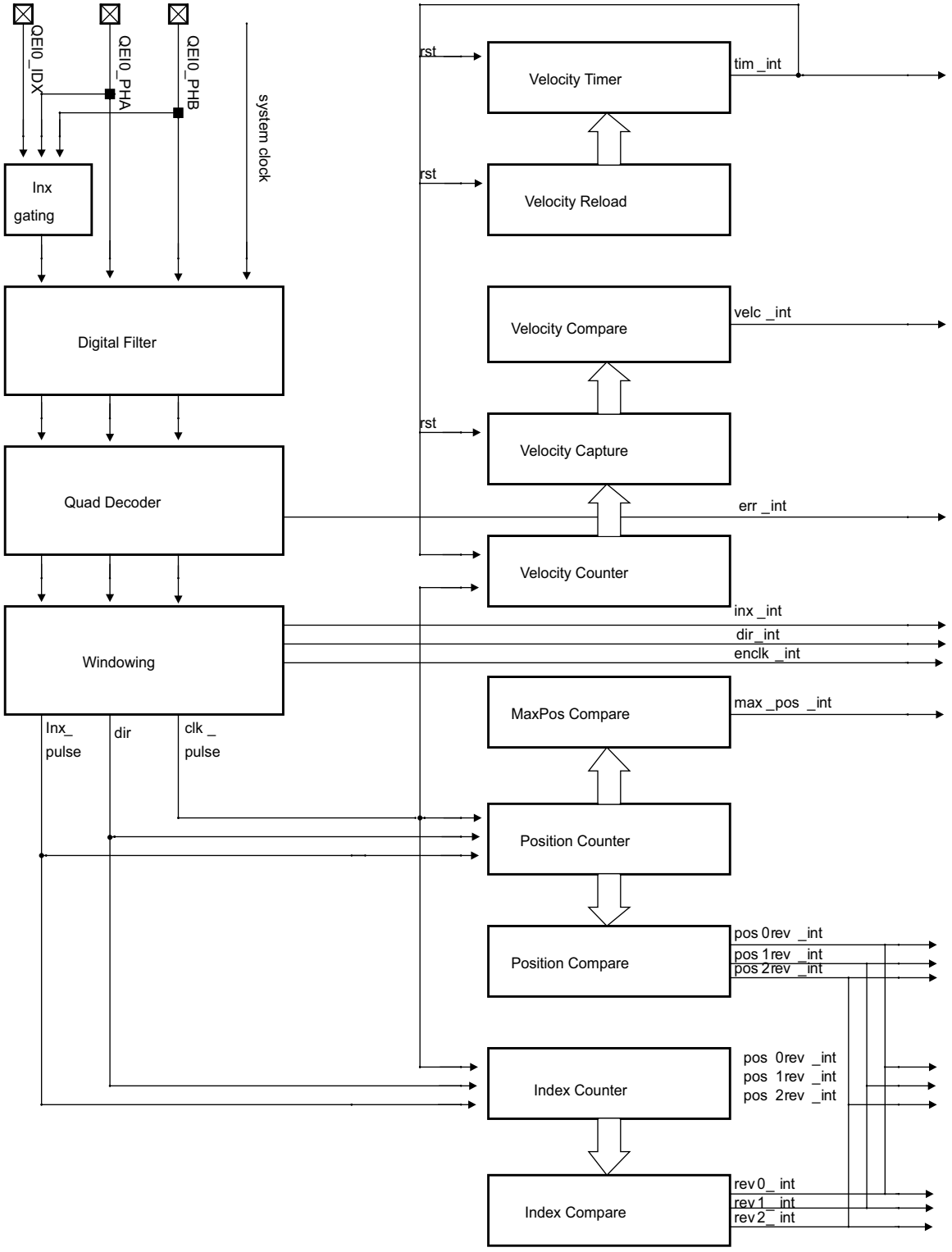


Fig 56. Encoder interface block diagram

22.5 Pin description

The QEI control signals are movable functions and are assigned to pins through the switch matrix.

Table 296. QEI pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
QEI0_PHA	I	external to pin	any pin	PINASSIGN14	Table 121	Phase A (PhA) input to the Quadrature Encoder Interface.
QEI0_PHB	I	external to pin	any pin	PINASSIGN14	Table 121	Phase B (PhB) input to the Quadrature Encoder Interface.
QEI0_IDX	I	external to pin	any pin	PINASSIGN14	Table 121	Index (IDX) input to the Quadrature Encoder Interface.

22.6 Register description

Table 297. Register overview: QEI (base address 0x4005 8000)

Name	Access	Address offset	Description	Reset value	Reference
Control registers					
CON	WO	0x000	Control register	0	Table 298
STAT	RO	0x004	Encoder status register	0	Table 299
CONF	R/W	0x008	Configuration register	0x000F 0000	Table 300
Position, index, and timer registers					
POS	RO	0x00C	Position register	0	Table 301
MAXPOS	R/W	0x010	Maximum position register	0	Table 302
CMPOS0	R/W	0x014	position compare register 0	0xFFFF FFFF	Table 303
CMPOS1	R/W	0x018	position compare register 1	0xFFFF FFFF	Table 304
CMPOS2	R/W	0x01C	position compare register 2	0xFFFF FFFF	Table 305
INXCNT	RO	0x020	Index count register	0	Table 306
INXCMP0	R/W	0x024	Index compare register 0	0xFFFF FFFF	Table 307
LOAD	R/W	0x028	Velocity timer reload register	0xFFFF FFFF	Table 308
TIME	RO	0x02C	Velocity timer register	0xFFFF FFFF	Table 309
VEL	RO	0x030	Velocity counter register	0	Table 310
CAP	RO	0x034	Velocity capture register	0xFFFF FFFF	Table 311
VELCOMP	R/W	0x038	Velocity compare register	0	Table 312
FILTERPHA	R/W	0x03C	Digital filter register on input phase A (QEI_A)	0	Table 313
FILTERPHB	R/W	0x040	Digital filter register on input phase B (QEI_B)	0	Table 314
FILTERINX	R/W	0x044	Digital filter register on input index (QEI_IDX)	0	Table 315
WINDOW	R/W	0x048	Index acceptance window register	0x0000 0000	Table 316
INXCMP1	R/W	0x04C	Index compare register 1	0xFFFF FFFF	Table 317

Table 297. Register overview: QEI (base address 0x4005 8000)

Name	Access	Address offset	Description	Reset value	Reference
INXCMP2	R/W	0x050	Index compare register 2	0xFFFF FFFF	Table 318
Interrupt registers					
IEC	WO	0xFD8	Interrupt enable clear register	0	Table 319
IES	WO	0xFDC	Interrupt enable set register	0	Table 320
INTSTAT	RO	0xFE0	Interrupt status register	0	Table 321
IE	RO	0xFE4	Interrupt enable register	0	Table 322
CLR	WO	0xFE8	Interrupt status clear register	0	Table 323
SET	WO	0xFEC	Interrupt status set register	0	Table 324

22.6.1 Control registers

22.6.1.1 QEI Control register

This register contains bits which control the operation of the position and velocity counters of the QEI module.

Table 298: QEI Control register (CON, address 0x4005 8000) bit description

Bit	Symbol	Description	Reset value
0	RESP	Reset position counter. When set = 1, resets the position counter to all zeros. Auto-clears when the position counter is cleared.	0
1	RESPI	Reset position counter on index. When set = 1, resets the position counter to all zeros when an index pulse occurs. Auto-clears when the position counter is cleared.	0
2	RESV	Reset velocity. When set = 1, resets the velocity counter to all zeros and reloads the velocity timer. Auto-clears when the velocity counter is cleared.	0
3	RESI	Reset index counter. When set = 1, resets the index counter to all zeros. Auto-clears when the index counter is cleared.	0
31:4	-	reserved	0

22.6.1.2 QEI Status register

This register provides the status of the encoder interface.

Table 299: QEI Interrupt Status register (STAT, address 0x4005 8004) bit description

Bit	Symbol	Description	Reset value
0	DIR	Direction bit. In combination with DIRINV bit indicates forward or reverse direction. See Table 327.	
31:1	-	reserved	0

22.6.1.3 QEI Configuration register

This register contains the configuration of the QEI module.

Table 300: QEI Configuration register (CONF, address 0x4005 8008) bit description

Bit	Symbol	Description	Reset value
0	DIRINV	Direction invert. When = 1, complements the DIR bit.	0
1	SIGMODE	Signal Mode. When = 0, PhA and PhB function as quadrature encoder inputs. When = 1, PhA functions as the direction signal and PhB functions as the clock signal.	0
2	CAPMODE	Capture Mode. When = 0, only PhA edges are counted (2X). When = 1, BOTH PhA and PhB edges are counted (4X), increasing resolution but decreasing range.	0
3	INVINX	Invert Index. When set, inverts the sense of the index input.	0
4	CRESPI	Continuously reset position counter on index. When set = 1, resets the position counter to all zeros when an index pulse occurs at the next position increase (recalibration). Auto-clears when the position counter is cleared.	0
15:5	-	Reserved	0
19:16	INXGATE	Index gating configuration: when INXGATE(19)=1, pass the index when Pha=0 and Phb=0, else block. when INXGATE(18)=1, pass the index when Pha=0 and Phb=1, else block. when INXGATE(17)=1, pass the index when Pha=1 and Phb=1, else block. when INXGATE(16)=1, pass the index when Pha=1 and Phb=0, else block.	1111
31:20	-	reserved	0

22.6.2 Position, index and timer registers

22.6.2.1 QEI Position register

This register contains the current value of the encoder position. Increments or decrements when encoder counts occur, depending on the direction of rotation.

Table 301. QEI Position register (POS, address 0x4005 800C) bit description

Bit	Symbol	Description	Reset value
31:0	POS	Current position value.	0

22.6.2.2 QEI Maximum Position register

This register contains the maximum value of the encoder position. In forward rotation the position register resets to zero when the position register exceeds this value. In reverse rotation the position register resets to this value when the position register decrements from zero.

Table 302. QEI Maximum Position register (MAXPOS, address 0x4005 8010) bit description

Bit	Symbol	Description	Reset value
31:0	MAXPOS	Maximum position value.	0

22.6.2.3 QEI Position Compare register 0

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 303. QEI Position Compare register 0 (CMPOS0, address 0x4005 8014) bit description

Bit	Symbol	Description	Reset value
31:0	PCMP0	Position compare value 0.	0xFFFF FFFF

22.6.2.4 QEI Position Compare register 1

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 304. QEI Position Compare register 1 (CMPOS1, address 0x4005 8018) bit description

Bit	Symbol	Description	Reset value
31:0	PCMP1	Position compare value 1.	0xFFFF FFFF

22.6.2.5 QEI Position Compare register 2

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 305. QEI Position Compare register 2 (CMPOS2, address 0x4005 801C) bit description

Bit	Symbol	Description	Reset value
31:0	PCMP2	Position compare value 2.	0xFFFF FFFF

22.6.2.6 QEI Index Count register

This register contains the current value of the encoder position. Increments or decrements when encoder counts occur, depending on the direction of rotation.

Table 306. QEI Index Count register (INXCNT, address 0x4005 8020) bit description

Bit	Symbol	Description	Reset value
31:0	ENCPOS	Current encoder position value.	0

22.6.2.7 QEI Index Compare register 0

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

Table 307. QEI Index Compare register 0 (INXCMP0, address 0x4005 8024) bit description

Bit	Symbol	Description	Reset value
31:0	ICMP0	Index compare value.	0xFFFF FFFF

22.6.2.8 QEI Timer Reload register

This register contains the reload value of the velocity timer. When the timer (TIME) reaches zero or the RESV bit is asserted, this value is loaded into the timer (TIME).

Table 308. QEI Timer Load register (LOAD, address 0x4005 8028) bit description

Bit	Symbol	Description	Reset value
31:0	VELLOAD	Current velocity timer pre-load value.The velocity timer counts down from this value.	0xFFFF FFFF

22.6.2.9 QEI Timer register

This register contains the current value of the velocity timer. When this timer reaches zero, the value of velocity register (VEL) is stored in the velocity capture register (CAP), the timer is reloaded with the value stored in the velocity reload register (LOAD), and the velocity interrupt (TIM_Int) is asserted.

Table 309. QEI Timer register (TIME, address 0x4005 802C) bit description

Bit	Symbol	Description	Reset value
31:0	VELVAL	Current velocity timer value.	0xFFFF FFFF

22.6.2.10 QEI Velocity register

This register contains the running count of velocity pulses for the current time period. When the velocity timer (TIME) reaches zero, the content of this register is captured in the velocity capture register (CAP). After capture, this register is set to zero. This register is also reset when the velocity reset bit (RESV) is asserted.

Table 310. QEI Velocity register (VEL, address 0x4005 8030) bit description

Bit	Symbol	Description	Reset value
31:0	VELPC	Current velocity pulse count.	0x0

22.6.2.11 QEI Velocity Capture register

This register contains the most recently measured velocity of the encoder. This corresponds to the number of velocity pulses counted in the previous velocity timer period.The current velocity count is latched into this register when the velocity timer overflows.

Table 311. QEI Velocity Capture register (CAP, address 0x4005 8034) bit description

Bit	Symbol	Description	Reset value
31:0	VELCAP	Velocity capture value.	0xFFFF FFFF

22.6.2.12 QEI Velocity Compare register

This register contains a velocity compare value. This value is compared against the captured velocity in the velocity capture register. If the capture velocity is less than the value in this compare register, a velocity compare interrupt (VELC_Int) will be asserted, if enabled.

Table 312. QEI Velocity Compare register (VELCOMP, address 0x4005 8038) bit description

Bit	Symbol	Description	Reset value
31:0	VELCMP	Velocity compare value.	0x0

22.6.2.13 QEI Digital filter on phase A input register

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

Table 313. QEI Digital filter on phase A input register (FILTERPHA, 0x4005 803C) bit description

Bit	Symbol	Description	Reset value
31:0	FILTA	Digital filter sampling delay	0x0

22.6.2.14 QEI Digital filter on phase B input register

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

Table 314. QEI Digital filter on phase B input register (FILTERPHB, 0x4005 8040) bit description

Bit	Symbol	Description	Reset value
31:0	FILTB	Digital filter sampling delay	0x0

22.6.2.15 QEI Digital filter on index input register

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

Table 315. QEI Digital filter on index input register (FILTERINX, 0x4005 8044) bit description

Bit	Symbol	Description	Reset value
31:0	FITLINX	Digital filter sampling delay	0x0

22.6.2.16 QEI Index acceptance window register

This register contains the width of the index acceptance window, when the index and the phase / clock edges fall nearly together. If the activating phase / clock edge falls before the Index, but within the window, the (re)calibration will be activated on that clock/phase edge.

Table 316. QEI Index acceptance window register (WINDOW, 0x4005 8048) bit description

Bit	Symbol	Description	Reset value
31:0	WINDOW	Index acceptance window width	0x0

22.6.2.17 QEI Index Compare register 1

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

Table 317. QEI Index Compare register 1 (INXCMP1, address 0x4005 804C) bit description

Bit	Symbol	Description	Reset value
31:0	ICMP1	Index compare value 1.	0xFFFF FFFF

22.6.2.18 QEI Index Compare register 2

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

Table 318. QEI Index Compare register 2 (INXCMP2, address 0x4005 8050) bit description

Bit	Symbol	Description	Reset value
31:0	ICMP2	Index compare value 2.	0xFFFF FFFF

22.6.3 Interrupt registers

22.6.3.1 QEI Interrupt Enable Clear register

Writing a 1 to a bit in this register clears the corresponding bit in the QEI Interrupt Enable register (QEIIE).

Table 319: QEI Interrupt Enable Clear register (IEC, address 0x4005 8FD8) bit description

Bit	Symbol	Description	Reset value
0	INX_EN	Indicates that an index pulse was detected.	0
1	TIM_EN	Indicates that a velocity timer overflow occurred	0
2	VELC_EN	Indicates that captured velocity is less than compare velocity.	0
3	DIR_EN	Indicates that a change of direction was detected.	0
4	ERR_EN	Indicates that an encoder phase error was detected.	0
5	ENCLK_EN	Indicates that and encoder clock pulse was detected.	0
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1compare value is equal to the current position.	0
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index 0 compare value is equal to the current index count.	0
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_Int is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	0
31:16	-	Reserved	0

22.6.3.2 QEI Interrupt Enable Set register

Writing a 1 to a bit in this register sets the corresponding bit in the QEI Interrupt Enable register (QEIIE).

Table 320: QEI Interrupt Enable Set register (IES, address 0x4005 8FDC) bit description

Bit	Symbol	Description	Reset value
0	INX_EN	Indicates that an index pulse was detected.	0
1	TIM_EN	Indicates that a velocity timer overflow occurred	0
2	VELC_EN	Indicates that captured velocity is less than compare velocity.	0
3	DIR_EN	Indicates that a change of direction was detected.	0
4	ERR_EN	Indicates that an encoder phase error was detected.	0
5	ENCLK_EN	Indicates that and encoder clock pulse was detected.	0
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1 compare value is equal to the current position.	0

Table 320: QEI Interrupt Enable Set register (IES, address 0x4005 8FDC) bit description

Bit	Symbol	Description	Reset value
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index compare value is equal to the current index count.	0
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_INT is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	0
31:16	-	Reserved	0

22.6.3.3 QEI Interrupt Status register

This register provides the status of the encoder interface and the current set of interrupt sources that are asserted to the controller. Bits set to 1 indicate the latched events that have occurred; a zero bit indicates that the event in question has not occurred. Writing a 0 to a bit position clears the corresponding interrupt.

Table 321: QEI Interrupt Status register (INTSTAT, address 0x4005 8FE0) bit description

Bit	Symbol	Description	Reset value
0	INX_INT	Indicates that an index pulse was detected.	0
1	TIM_INT	Indicates that a velocity timer overflow occurred	0
2	VELC_INT	Indicates that captured velocity is less than compare velocity.	0
3	DIR_INT	Indicates that a change of direction was detected.	0
4	ERR_INT	Indicates that an encoder phase error was detected.	0
5	ENCLK_INT	Indicates that and encoder clock pulse was detected.	0
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1compare value is equal to the current position.	0
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index compare value is equal to the current index count.	0
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_INT is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	0
31:16	-	Reserved	0

22.6.3.4 QEI Interrupt Enable register

This register enables interrupt sources. Bits set to 1 enable the corresponding interrupt; a 0 bit disables the corresponding interrupt.

Table 322: QEI Interrupt Enable register (IE, address 0x4005 8FE4) bit description

Bit	Symbol	Description	Reset value
0	INX_INT	Indicates that an index pulse was detected.	0
1	TIM_INT	Indicates that a velocity timer overflow occurred	0
2	VELC_INT	Indicates that captured velocity is less than compare velocity.	0
3	DIR_INT	Indicates that a change of direction was detected.	0
4	ERR_INT	Indicates that an encoder phase error was detected.	0
5	ENCLK_INT	Indicates that and encoder clock pulse was detected.	0
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1compare value is equal to the current position.	0
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index compare value is equal to the current index count.	0
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_INT is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	0
31:16	-	Reserved	0

22.6.3.5 QEI Interrupt Status Clear register

Writing a 1 to a bit in this register clears the corresponding bit in the QEI Interrupt Status register (QEISTAT).

Table 323: QEI Interrupt Status Clear register (CLR, 0x4005 8FE8) bit description

Bit	Symbol	Description	Reset value
0	INX_INT	Indicates that an index pulse was detected.	0
1	TIM_INT	Indicates that a velocity timer overflow occurred	0
2	VELC_INT	Indicates that captured velocity is less than compare velocity.	0
3	DIR_INT	Indicates that a change of direction was detected.	0
4	ERR_INT	Indicates that an encoder phase error was detected.	0
5	ENCLK_INT	Indicates that and encoder clock pulse was detected.	0
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1compare value is equal to the current position.	0
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index compare value is equal to the current index count.	0

Table 323: QEI Interrupt Status Clear register (CLR, 0x4005 8FE8) bit description

Bit	Symbol	Description	Reset value
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_INT is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	
31:16	-	Reserved	0

22.6.3.6 QEI Interrupt Status Set register

Writing a one to a bit in this register sets the corresponding bit in the QEI Interrupt Status register (STAT).

Table 324: QEI Interrupt Status Set register (SET, address 0x4005 8FEC) bit description

Bit	Symbol	Description	Reset value
0	INX_INT	Indicates that an index pulse was detected.	0
1	TIM_INT	Indicates that a velocity timer overflow occurred	0
2	VELC_INT	Indicates that captured velocity is less than compare velocity.	0
3	DIR_INT	Indicates that a change of direction was detected.	0
4	ERR_INT	Indicates that an encoder phase error was detected.	0
5	ENCLK_INT	Indicates that and encoder clock pulse was detected.	
6	POS0_INT	Indicates that the position 0 compare value is equal to the current position.	0
7	POS1_INT	Indicates that the position 1compare value is equal to the current position.	0
8	POS2_INT	Indicates that the position 2 compare value is equal to the current position.	0
9	REV0_INT	Indicates that the index compare value is equal to the current index count.	0
10	POS0REV_INT	Combined position 0 and revolution count interrupt. Set when both the POS0_INT bit is set and the REV0_INT is set.	0
11	POS1REV_INT	Combined position 1 and revolution count interrupt. Set when both the POS1_INT bit is set and the REV1_INT is set.	0
12	POS2REV_INT	Combined position 2 and revolution count interrupt. Set when both the POS2_INT bit is set and the REV2_INT is set.	0
13	REV1_INT	Indicates that the index 1 compare value is equal to the current index count.	0
14	REV2_INT	Indicates that the index 2 compare value is equal to the current index count.	0
15	MAXPOS_INT	Indicates that the current position count goes through the MAXPOS value to zero in forward direction, or through zero to MAXPOS in backward direction.	0
31:16	-	Reserved	0

22.7 Functional description

The QEI module interprets the two-bit gray code produced by a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.

22.7.1 Input signals

The QEI module supports two modes of signal operation: quadrature phase mode and clock/direction mode. In quadrature phase mode, the encoder produces two clocks that are 90 degrees out of phase; the edge relationship is used to determine the direction of rotation. In clock/direction mode, the encoder produces a clock signal to indicate steps and a direction signal to indicate the direction of rotation.).

This mode is determined by the SigMode bit of the QEI Control (CON) register (See Table 298). When the SigMode bit = 1, the quadrature decoder is bypassed and the PhA pin functions as the direction signal and PhB pin functions as the clock signal for the counters, etc. When the SigMode bit = 0, the PhA pin and PhB pins are decoded by the quadrature decoder. In this mode the quadrature decoder produces the direction and clock signals for the counters, etc. In both modes the direction signal is subject to the effects of the direction invert (DIRINV) bit.

The input signals are synchronized to the QEI peripheral clock QEI_CLK, which is identical to the system clock, see Figure 55. In order for the single-stage synchronizer to capture the input signal, the two phase inputs have to overlap at least one clock cycle of the QEI_PCLK. The signal minimum HIGH and LOW times should also be one clock cycle of the QEI_PCLK. Since the frequency of the input signals is typically much lower than the system clock, these conditions are always met in practical applications,

22.7.1.1 Quadrature input signals

When edges on PhA lead edges on PhB, the position counter is incremented. When edges on PhB lead edges on PhA, the position counter is decremented. When a rising and falling edge pair is seen on one of the phases without any edges on the other, the direction of rotation has changed.

Table 325. Encoder states

Phase A	Phase B	state
1	0	1
1	1	2
0	1	3
0	0	4

Table 326. Encoder state transitions[1]

from state	to state	Direction
1	2	positive
2	3	
3	4	
4	1	

Table 326. Encoder state transitions[1]

from state	to state	Direction
4	3	negative
3	2	
2	1	
1	4	

[1] All other state transitions are illegal and should set the ERR bit.

Interchanging of the PhA and PhB input signals are compensated by complementing the DIR bit. When set = 1, the direction inversion bit (DIRINV) complements the DIR bit.

Table 327. Encoder direction

DIR bit	DIRINV bit	direction
0	0	forward
1	0	reverse
0	1	reverse
1	1	forward

Figure 57 shows how quadrature encoder signals equate to direction and count.

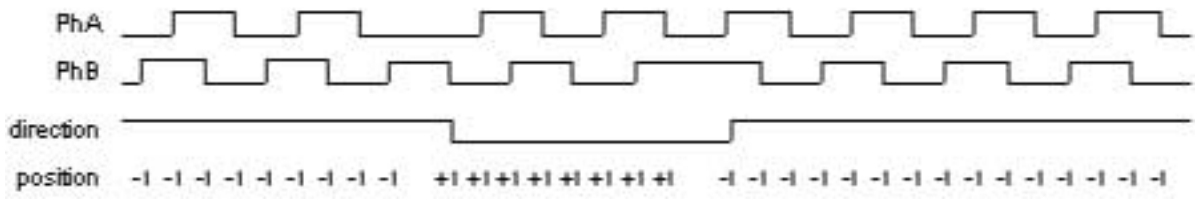


Fig 57. Quadrature Encoder basic operation

22.7.1.2 Digital input filtering

All three encoder inputs (PhA, PhB, and index) require digital filtering. The number of sample clocks is user programmable from 1 to 4,294,967,295 (0xFFFF FFFF). In order for a transition to be accepted, the input signal must remain in new state for the programmed number of sample clocks.

22.7.2 Position capture

The capture mode for the position integrator can be set to update the position counter on every edge of the PhA signal or to update on every edge of both PhA and PhB. Updating the position counter on every PhA and PhB provides more positional resolution at the cost of less range in the positional counter.

The position integrator and velocity capture can be independently enabled. Alternatively, the phase signals can be interpreted as a clock and direction signal as output by some encoders.

The position counter is automatically reset on one of three conditions. Incrementing past the maximum position value (MAXPOS) will reset the position counter to zero. If the reset on index bit (RESPI) is set, sensing the index pulse for the first time will once reset the position counter to zero after the next positional increase (calibrate). If the continuously reset on index bit (CRESPI) is set, sensing the index pulse will continuously reset the position counter to zero after the next positional increase (recalibrate).

22.7.3 Velocity capture

The velocity capture has a programmable timer and a capture register. It counts the number of phase edges (using the same configuration as for the position integrator) in a given time period. When the velocity timer (TIME) reaches zero, the contents of the velocity counter (VEL) are transferred to the capture (CAP) register. The velocity counter is then cleared. The velocity timer is loaded with the contents of the velocity reload register (LOAD). Finally, the velocity interrupt (TIM_Int) is asserted. The number of edges counted in a given time period is directly proportional to the velocity of the encoder.

Setting the reset velocity bit (RESV) will clear the velocity counter, reset the velocity capture register to 0xFFFF FFFF, and load the velocity timer with the contents of the velocity reload register (LOAD).

The following equation converts the velocity counter value into an RPM value:

RPM = (PCLK × Speed × 60) / Load × PPR × Edges)

where:

- PCLK is the QEI controller clock.
- PPR is the number of pulses per revolution of the physical encoder.
- Edges is 2 or 4, based on the capture mode set in the CON register (2 for CapMode set to 0 and 4 for CapMode set to 1)

For example, consider a motor running at 600 rpm. A 2048 pulse per revolution quadrature encoder is attached to the motor, producing 8192 phase edges per revolution. With clocking on both PhA and PhB edges, this results in 81,920 pulses per second (the motor turns 10 times per second). If the timer were clocked at 10,000 Hz, and the load value was 2,500 (¼ of a second), it would count 20,480 pulses per update. Using the above equation:

RPM = (10000 × 20480 × 60) / (2500 × 2048 × 4) = 600 RPM

Now, consider that the motor is sped up to 3000 RPM. This results in 409,600 pulses per second, or 102,400 every ¼ of a second. Again, the above equation gives:

RPM = (10000 × 102400 × 60) / (2500 × 2048 × 4) = 3000 RPM

22.7.4 Velocity compare

In addition to velocity capture, the velocity measurement system includes a programmable velocity compare register. After every velocity capture event the contents of the velocity capture register (CAP) is compared with the contents of the velocity

compare register (VELCOMP). If the captured velocity is less than the compare value an interrupt is asserted provided that the velocity compare interrupt enable bit is set. This can be used to determine if a motor shaft is either stalled or moving too slow.

23.1 How to read this chapter

The USB controller is only available on parts LPC1549/48/47.

23.2 Features

- USB2.0 full-speed device controller.
- Supports 10 physical (5 logical) endpoints including one control endpoint.
- Single and double-buffering supported.
- Each non-control endpoint supports bulk, interrupt, or isochronous endpoint types.
- Supports wake-up from Deep-sleep and Power-down modes on USB activity and remote wake-up.
- SoftConnect and serial 33 Ω resistors on USB_DP and USB_DM implemented internally.

23.3 Basic configuration

Configure the USB as follows:

- Use the SYSAHBCLKCTRL1 register (Table 51) to enable the clock to the USB register interface.
- Clear the USB peripheral reset using the PRESETCTRL1 register (Table 36).
- Turn on the USB PHY by setting the USB_PHY bit in the PDRUNCFG register (Table 75).
- Configure the USBPLL to create a 48 MHz USB clock.
- The USB block creates three interrupts which are connected to slot #28 (USB_INT), slot #29 (USB_FIQ), and slot #30 (USB_WAKE) in the NVIC.
- Use the switch matrix to connect the USB functions USB_VBUS, and USB_FTOGGLE to pins. The USB_DP and USB_DM pins are dedicated pins. Do not assign the USB_FTOGGLE function to a pin until a USB device is connected and the first SOF interrupt has been received by the device.
- The USB_FTOGGLE output is connected to the SCT2/3 input muxes.

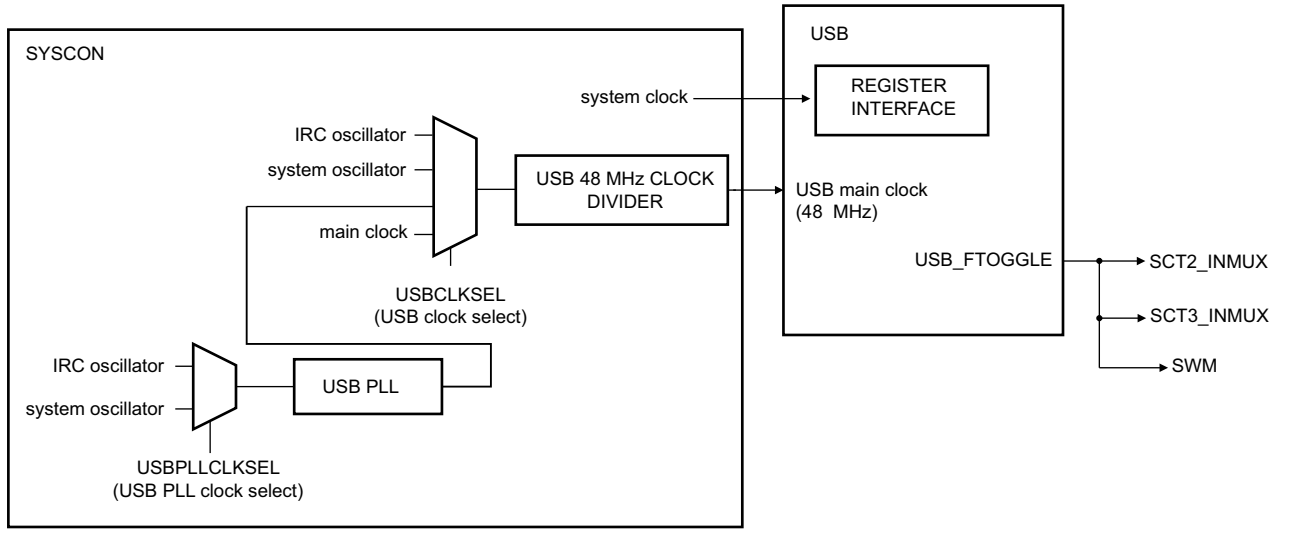


Fig 58. USB clocking

23.4 Pin description

The device controller can access one USB port.

The USB control signals are movable functions and are assigned to pins through the switch matrix.

Table 328. USB pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
VBUS	I	external to pin	any pin	PINASSIGN7	Table 114	V_{BUS} status input. When this function is not enabled via its corresponding IOCON register, it is driven HIGH internally.
USB_FTOGGLE	O	external to pin	any pin	PINASSIGN14	Table 121	USB 1 ms SoF signal. Do not assign this function to a pin until a USB device is connected and the first SOF interrupt has been received by the device.
		internal	SCT2	SCT2_INMUX[2:0]	Table 129	USB 1 ms SoF signal.
		internal	SCT3	SCT3_INMUX[2:0]	Table 130	USB 1 ms SoF signal.
USB_DP	I/O	pin	-	-	-	Positive differential data. Pad includes internal 33 Ω series termination resistor.
USB_DM	I/O	pin		-	-	Negative differential data. Pad includes internal 33 Ω series termination resistor.

23.5 General description

The Universal Serial Bus (USB) is a four-wire bus that supports communication between a host and one or more (up to 127) peripherals. The host controller allocates the USB bandwidth to attached devices through a token-based protocol. The bus supports hot plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The host schedules transactions in 1 ms frames. Each frame contains a Start-Of-Frame (SOF) marker and transactions that transfer data to or from device endpoints. Each device can have a maximum of 16 logical or 32 physical endpoints. The device controller supports up to 10 physical endpoints. There are four types of transfers defined for the endpoints. Control transfers are used to configure the device.

Interrupt transfers are used for periodic data transfer. Bulk transfers are used when the latency of transfer is not critical. Isochronous transfers have guaranteed delivery time but no error correction.

For more information on the Universal Serial Bus, see the USB Implementers Forum website.

The USB device controller on the LPC15xx enables full-speed (12 Mb/s) data exchange with a USB host controller.

Figure 59 shows the block diagram of the USB device controller.

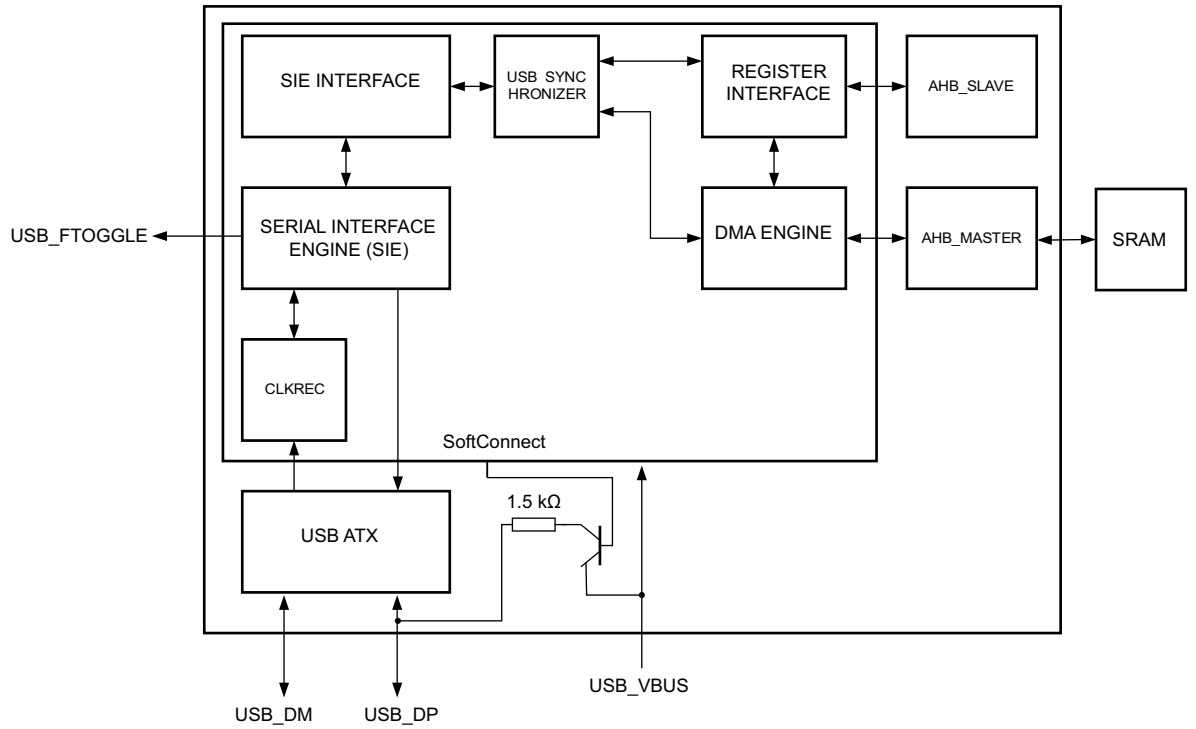


Fig 59. USB block diagram

The USB Device Controller has a built-in analog transceiver (ATX). The USB ATX sends/receives the bi-directional USB_DP and USB_DM signals of the USB bus.

The SIE implements the full USB protocol layer. It is completely hardwired for speed and needs no software intervention. It handles transfer of data between the endpoint buffers in USB RAM and the USB bus. The functions of this block include: synchronization pattern recognition, parallel/serial conversion, bit stuffing/de-stuffing, CRC checking/generation, PID verification/generation, address recognition, and handshake evaluation/generation.

23.5.1 USB software interface

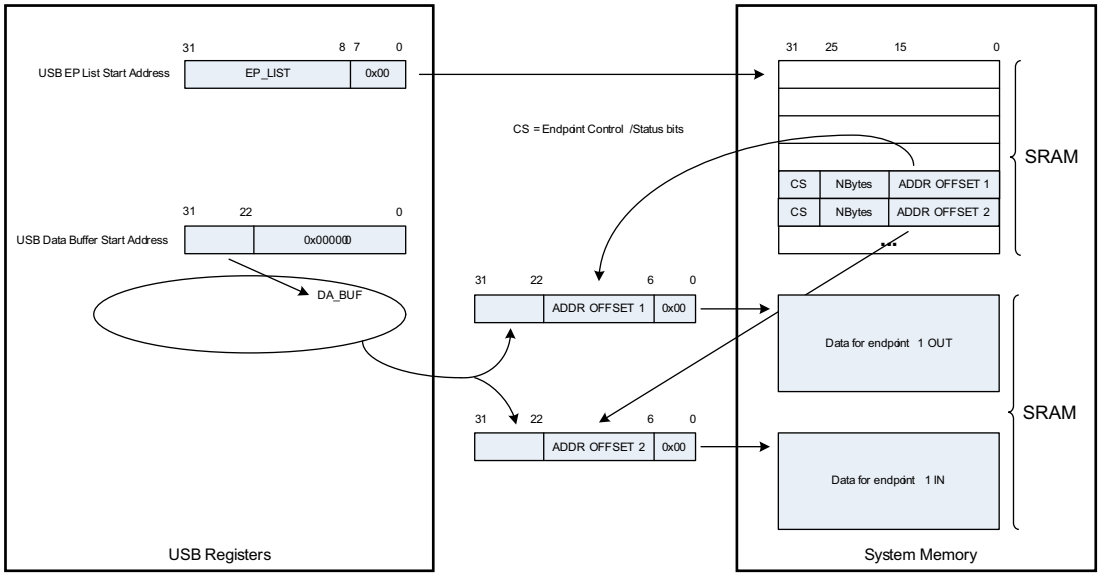


Fig 60. USB software interface

23.5.2 Fixed endpoint configuration

Table 329 shows the supported endpoint configurations. The packet size is configurable up to the maximum value shown in Table 329 for each type of end point.

Table 329. Fixed endpoint configuration

Logical endpoint	Physical endpoint	Endpoint type	Direction	Max packet size (byte)	Double buffer
0	0	Control	Out	64	No
0	1	Control	In	64	No
1	2	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
1	3	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
2	4	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
2	5	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
3	6	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
3	7	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes
4	8	Interrupt/Bulk/Isochronous	Out	64/64/1023	Yes
4	9	Interrupt/Bulk/Isochronous	In	64/64/1023	Yes

23.5.3 SoftConnect

The softConnect signal is implemented internally. An external pull-up resistor between USB_DP and V_{DD} is not necessary.

Software can control the USB_CONNECT signal by setting the DCON bit in the DEVCMDSTAT register. If the DCON bit is set to 1, the USB_DP line is pulled up to VDD through an internal 1.5 KOhm pull-up resistor.

The purpose of the soft connect feature using USB_CONNECT is to control when the device connects to the bus. When the device detects a USB_VBUS signal on the bus, it can finish processing if necessary, and then under software control indicate its presence to the host by pulling the USB_DP line HIGH. In a similar way, software can re-initialize a USB connection without the necessity to unplug the USB cable.

23.5.4 Interrupts

The USB controller has two interrupt lines USB_Int_Req_IRQ and USB_Int_Req_FIQ. Software can program the corresponding bit in the USB interrupt routing register to route the interrupt condition to one of these entries in the NVIC table Table 2. An interrupt is generated by the hardware if both the interrupt status bit and the corresponding interrupt enable bit are set. The interrupt status bit is set by hardware if the interrupt condition occurs (regardless of the interrupt enable bit setting).

23.5.5 Suspend and resume

The USB protocol insists on power management by the USB device. This becomes even more important if the device draws power from the bus (bus-powered device). The following constraints should be met by the bus-powered device.

- A device in the non-configured state should draw a maximum of 100 mA from the USB bus.
- A configured device can draw only up to what is specified in the Max Power field of the configuration descriptor. The maximum value is 500 mA.
- A suspended device should draw a maximum of 500 μA.

A device will go into the L2 suspend state if there is no activity on the USB bus for more than 3 ms. A suspended device wakes up, if there is transmission from the host (host-initiated wake up). The USB controller also supports software initiated remote wake-up. To initiate remote wake-up, software on the device must enable all clocks and clear the suspend bit. This will cause the hardware to generate a remote wake-up signal upstream.

The USB controller supports Link Power Management. Link Power Management defines an additional link power management state L1 that supplements the existing L2 state by utilizing most of the existing suspend/resume infrastructure but provides much faster transitional latencies between L1 and L0 (On).

The assertion of USB suspend signal indicates that there was no activity on the USB bus for the last 3 ms. At this time an interrupt is sent to the processor on which the software can start preparing the device for suspend.

If there is no activity for the next 2 ms, the USB need_clock signal will go low. This indicates that the USB main clock can be switched off.

When activity is detected on the USB bus, the USB suspend signal is deactivated and USB need_clock signal is activated. This process is fully combinatorial and hence no USB main clock is required to activate the US B need_clock signal.

23.5.6 Frame toggle output

The USB_FTOGGLE output pin reflects the 1 kHz clock derived from the incoming Start-of-Frame (SOF) tokens sent by the USB host. When the USB is connected to a host, the rising edge of the USB_FTOGGLE signal is aligned with the middle of the SOF token which is received on the USB bus. The signal can be monitored on a pin (connected through the switch matrix) or on the inputs of timers SCT2 or SCT3.

When no tokens are coming in, the USB_FTOGGLE input is a 1 KHz signal based on the USB main clock.

23.5.7 Clocking

The LPC15xx USB device controller has the following clock connections:

- USB main clock: The USB main clock is the 48 MHz +/- 500 ppm clock from the dedicated USB PLL or the main clock (see Table 42). If the main clock is used, the system PLL output must be 48 MHz and derived from the system oscillator. The USB main clock is used to recover the 12 MHz clock from the USB bus.
- AHB clock: This is the AHB system bus clock. The minimum frequency of the AHB clock is 6 MHz when the USB device controller is receiving or transmitting USB packets.

23.6 Register description

Table 330. Register overview: USB (base address: 0x1C00 C000)

Name	Access	Address offset	Description	Reset value	Reference
DEVCMDSTAT	R/W	0x000	USB Device Command/Status register	0x00000800	Table 331
INFO	R/W	0x004	USB Info register	0	Table 332
EPLISTSTART	R/W	0x008	USB EP Command/Status List start address	0	Table 333
DATABUFSTART	R/W	0x00C	USB Data buffer start address	0	Table 334
LPM	R/W	0x010	Link Power Management register	0	Table 335
EPSKIP	R/W	0x014	USB Endpoint skip	0	Table 336
EPINUSE	R/W	0x018	USB Endpoint Buffer in use	0	Table 337
EPBUFCFG	R/W	0x01C	USB Endpoint Buffer Configuration register	0	Table 338
INTSTAT	R/W	0x020	USB interrupt status register	0	Table 339
INTEN	R/W	0x024	USB interrupt enable register	0	Table 340
INTSETSTAT	R/W	0x028	USB set interrupt status register	0	Table 341
INTROUTING	R/W	0x02C	USB interrupt routing register	0	Table 342
EPTOGGLE	R	0x034	USB Endpoint toggle register	0	Table 343

23.6.1 USB Device Command/Status register

Table 331. USB Device Command/Status register (DEVCMDSTAT, address 0x1C00 C000) bit description

Bit	Symbol	Value	Description	Reset value	Access
6:0	DEV_ADDR		USB device address. After bus reset, the address is reset to 0x00. If the enable bit is set, the device will respond on packets for function address DEV_ADDR. When receiving a SetAddress Control Request from the USB host, software must program the new address before completing the status phase of the SetAddress Control Request.	0	RW
7	DEV_EN		USB device enable. If this bit is set, the USB block will start responding on packets for function address DEV_ADDR.	0	RW
		0	Disabled.		
		1	Enabled. USB device operation enabled.		
8	SETUP		SETUP token received. If a SETUP token is received and acknowledged by the device, this bit is set. As long as this bit is set all received IN and OUT tokens will be NAKed by HW. SW must clear this bit by writing a one. If this bit is zero, HW will handle the tokens to the CTRL EP0 as indicated by the CTRL EP0 IN and OUT data information programmed by SW.	0	RWC
9	PLL_ON		Always PLL Clock on:	0	RW
		0	Functional. USB_NeedClk functional.		
		1	High. USB_NeedClk always 1. Clock will not be stopped in case of suspend.		
10	-		Reserved.	0	RO
11	LPM_SUP		LPM Support.:	1	RW
		0	No. LPM not supported.		
		1	Yes.LPM supported.		
12	INTONNAK_AO		Interrupt on NAK for interrupt and bulk OUT EP	0	RW
		0	AK only. Only acknowledged packets generate an interrupt		
		1	Ak and Nak. Both acknowledged and NAKed packets generate interrupts.		
13	INTONNAK_AI		Interrupt on NAK for interrupt and bulk IN EP	0	RW
		0	AK only. Only acknowledged packets generate an interrupt		
		1	Ak and NAK. Both acknowledged and NAKed packets generate interrupts.		
14	INTONNAK_CO		Interrupt on NAK for control OUT EP	0	RW
		0	AK only. Only acknowledged packets generate an interrupt		
		1	AK and NAK. Both acknowledged and NAKed packets generate interrupts.		
15	INTONNAK_CI		Interrupt on NAK for control IN EP	0	RW
		0	AK only. Only acknowledged packets generate an interrupt		
		1	AK and NAK. Both acknowledged and NAKed packets generate interrupts.		

Table 331. USB Device Command/Status register (DEVCMDSTAT, address 0x1C00 C000) bit description

Bit	Symbol	Value	Description	Reset value	Access
16	DCON		Device status - connect.	0	RW
		0	Not connected.		
		1	Connect. The connect bit must be set by software to indicate that the device must signal a connect. The pull-up resistor on USB_DP will be enabled when this bit is set and the VBUSDEBOUNCED bit is one.		
17	DSUS		Device status - suspend. The suspend bit indicates the current suspend state. It is set to 1 when the device hasn't seen any activity on its upstream port for more than 3 milliseconds. It is reset to 0 on any activity. When the device is suspended (Suspend bit DSUS = 1) and the software writes a 0 to it, the device will generate a remote wake-up. This will only happen when the device is connected (Connect bit = 1). When the device is not connected or not suspended, a writing a 0 has no effect. Writing a 1 never has an effect.	0	RW
18	-		Reserved.	0	RO
19	LPM_SUS		Device status - LPM Suspend. This bit represents the current LPM suspend state. It is set to 1 by HW when the device has acknowledged the LPM request from the USB host and the Token Retry Time of 10us has elapsed. When the device is in the LPM suspended state (LPM suspend bit = 1) and the software writes a zero to this bit, the device will generate a remote walk-up. Software can only write a zero to this bit when the LPM_REWP bit is set to 1. HW resets this bit when it receives a host initiated resume. HW only updates the LPM_SUS bit when the LPM_SUPP bit is equal to one.	0	RW
20	LPM_REWP		LPM Remote Wake-up Enabled by USB host. HW sets this bit to one when the bRemoteWake bit in the LPM extended token is set to 1. HW will reset this bit to 0 when it receives the host initiated LPM resume, when a remote wake-up is sent by the device or when a USB bus reset is received. Software can use this bit to check if the remote wake-up feature is enabled by the host for the LPM transaction.	0	RO
23:21	-		Reserved.	0	RO
24	DCON_C		Device status - connect change. The Connect Change bit is set when the device's pull-up resistor is disconnected because VBus disappeared. The bit is reset by writing a one to it.	0	RWC
25	DSUS_C		Device status - suspend change. The suspend change bit is set to 1 when the suspend bit toggles. The suspend bit can toggle because: - The device goes in the suspended state - The device is disconnected - The device receives resume signaling on its upstream port. The bit is reset by writing a one to it.	0	RWC

Table 331. USB Device Command/Status register (DEVCMDSTAT, address 0x1C00 C000) bit description

Bit	Symbol	Value	Description	Reset value	Access
26	DRES_C		Device status - reset change. This bit is set when the device received a bus reset. On a bus reset the device will automatically go to the default state (unconfigured and responding to address 0). The bit is reset by writing a one to it.	0	RWC
27	-		Reserved.	0	RO
28	VBUSDEBOUNCED		This bit indicates if VBUS is detected or not. The bit raises immediately when VBUS becomes high. It drops to zero if VBUS is low for at least 3 ms. If this bit is high and the DCon bit is set, the HW will enable the pull-up resistor to signal a connect.	0	RO
31:29	-		Reserved.	0	RO

23.6.2 USB Info register

Table 332. USB Info register (INFO, address 0x1C00 C004) bit description

Bit	Symbol	Value	Description	Reset value	Access
10:0	FRAME_NR		Frame number. This contains the frame number of the last successfully received SOF. In case no SOF was received by the device at the beginning of a frame, the frame number returned is that of the last successfully received SOF. In case the SOF frame number contained a CRC error, the frame number returned will be the corrupted frame number as received by the device.	0	RO
14:11	ERR_CODE		The error code which last occurred:	0	RW
		0x0	No error		
		0x1	PID encoding error		
		0x2	PID unknown		
		0x3	Packet unexpected		
		0x4	Token CRC error		
		0x5	Data CRC error		
		0x6	Time out		
		0x7	Babble		
		0x8	Truncated EOP		
		0x9	Sent/Received NAK		
		0xA	Sent Stall		
		0xB	Overrun		
		0xC	Sent empty packet		
		0xD	Bitstuff error		
		0xE	Sync error		
		0xF	Wrong data toggle		
15	-		Reserved.	0	RO
31:16	-	-	Reserved	-	RO

23.6.3 USB EP Command/Status List start address

This 32-bit register indicates the start address of the USB EP Command/Status List.

Only a subset of these bits is programmable by software. The 8 least-significant bits are hardcoded to zero because the list must start on a 256 byte boundary. The bits 31 to 8 can be programmed by software.

Table 333. USB EP Command/Status List start address (EPLISTSTART, address 0x1C00 C008) bit description

Bit	Symbol	Description	Reset value	Access
7:0	-	Reserved	0	RO
31:8	EP_LIST	Start address of the USB EP Command/Status List.	0	R/W

23.6.4 USB Data buffer start address

This register indicates the page of the AHB address where the endpoint data can be located.

Table 334. USB Data buffer start address (DATABUFSTART, address 0x1C00 C00C) bit description

Bit	Symbol	Description	Reset value	Access
21:0	-	Reserved	0	R
31:22	DA_BUF	Start address of the buffer pointer page where all endpoint data buffers are located.	0	R/W

23.6.5 Link Power Management register

Table 335. Link Power Management register (LPM, address 0x1C00 C010) bit description

Bit	Symbol	Description	Reset value	Access
3:0	HIRD_HW	Host Initiated Resume Duration - HW. This is the HIRD value from the last received LPM token	0	RO
7:4	HIRD_SW	Host Initiated Resume Duration - SW. This is the time duration required by the USB device system to come out of LPM initiated suspend after receiving the host initiated LPM resume.	0	R/W
8	DATA_PENDING	As long as this bit is set to one and LPM supported bit is set to one, HW will return a NYET handshake on every LPM token it receives. If LPM supported bit is set to one and this bit is zero, HW will return an ACK handshake on every LPM token it receives. If SW has still data pending and LPM is supported, it must set this bit to 1.	0	R/W
31:9	-	Reserved	0	RO

23.6.6 USB Endpoint skip

Table 336. USB Endpoint skip (EPSKIP, address 0x1C00 C014) bit description

Bit	Symbol	Description	Reset value	Access
29:0	SKIP	Endpoint skip: Writing 1 to one of these bits, will indicate to HW that it must deactivate the buffer assigned to this endpoint and return control back to software. When HW has deactivated the endpoint, it will clear this bit, but it will not modify the EPINUSE bit. An interrupt will be generated when the Active bit goes from 1 to 0. Note: In case of double-buffering, HW will only clear the Active bit of the buffer indicated by the EPINUSE bit.	0	R/W
31:30	-	Reserved	0	R

23.6.7 USB Endpoint Buffer in use

Table 337. USB Endpoint Buffer in use (EPINUSE, address 0x1C00 C018) bit description

Bit	Symbol	Description	Reset value	Access
1:0	-	Reserved. Fixed to zero because the control endpoint zero is fixed to single-buffering for each physical endpoint.	0	R
9:2	BUF	Buffer in use: This register has one bit per physical endpoint. 0: HW is accessing buffer 0. 1: HW is accessing buffer 1.	0	R/W
31:10	-	Reserved	0	R

23.6.8 USB Endpoint Buffer Configuration

Table 338. USB Endpoint Buffer Configuration (EPBUFCFG, address 0x1C00 C01C) bit description

Bit	Symbol	Description	Reset value	Access
1:0	-	Reserved. Fixed to zero because the control endpoint zero is fixed to single-buffering for each physical endpoint.	0	R
9:2	BUF_SB	Buffer usage: This register has one bit per physical endpoint. 0: Single-buffer. 1: Double-buffer. If the bit is set to single-buffer (0), it will not toggle the corresponding EPINUSE bit when it clears the active bit. If the bit is set to double-buffer (1), HW will toggle the EPINUSE bit when it clears the Active bit for the buffer.	0	R/W
31:10	-	Reserved	0	R

23.6.9 USB interrupt status register

Table 339. USB interrupt status register (INTSTAT, address 0x1C00 C020) bit description

Bit	Symbol	Description	Reset value	Access
0	EP0OUT	Interrupt status register bit for the Control EP0 OUT direction. This bit will be set if NBytes transitions to zero or the skip bit is set by software or a SETUP packet is successfully received for the control EP0. If the IntOnNAK_CO is set, this bit will also be set when a NAK is transmitted for the Control EP0 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
1	EP0IN	Interrupt status register bit for the Control EP0 IN direction. This bit will be set if NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_CI is set, this bit will also be set when a NAK is transmitted for the Control EP0 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
2	EP1OUT	Interrupt status register bit for the EP1 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP1 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
3	EP1IN	Interrupt status register bit for the EP1 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP1 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
4	EP2OUT	Interrupt status register bit for the EP2 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP2 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
5	EP2IN	Interrupt status register bit for the EP2 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP2 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
6	EP3OUT	Interrupt status register bit for the EP3 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP3 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC

Table 339. USB interrupt status register (INTSTAT, address 0x1C00 C020) bit description

Bit	Symbol	Description	Reset value	Access
7	EP3IN	Interrupt status register bit for the EP3 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP3 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
8	EP4OUT	Interrupt status register bit for the EP4 OUT direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AO is set, this bit will also be set when a NAK is transmitted for the EP4 OUT direction. Software can clear this bit by writing a one to it.	0	R/WC
9	EP4IN	Interrupt status register bit for the EP4 IN direction. This bit will be set if the corresponding Active bit is cleared by HW. This is done in case the programmed NBytes transitions to zero or the skip bit is set by software. If the IntOnNAK_AI is set, this bit will also be set when a NAK is transmitted for the EP4 IN direction. Software can clear this bit by writing a one to it.	0	R/WC
29:10	-	Reserved	0	RO
30	FRAME_INT	Frame interrupt. This bit is set to one every millisecond when the VbusDebounced bit and the DCON bit are set. This bit can be used by software when handling isochronous endpoints. Software can clear this bit by writing a one to it.	0	R/WC
31	DEV_INT	Device status interrupt. This bit is set by HW when one of the bits in the Device Status Change register are set. Software can clear this bit by writing a one to it.	0	R/WC

23.6.10 USB interrupt enable register

Table 340. USB interrupt enable register (INTEN, address 0x1C00 C024) bit description

Bit	Symbol	Description	Reset value	Access
9:0	EP_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W
29:10	-	Reserved	0	RO
30	FRAME_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W
31	DEV_INT_EN	If this bit is set and the corresponding USB interrupt status bit is set, a HW interrupt is generated on the interrupt line indicated by the corresponding USB interrupt routing bit.	0	R/W

23.6.11 USB set interrupt status register

Table 341. USB set interrupt status register (INTSETSTAT, address 0x1C00 C028) bit description

Bit	Symbol	Description	Reset value	Access
9:0	EP_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W
29:10	-	Reserved	0	RO
30	FRAME_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W
31	DEV_SET_INT	If software writes a one to one of these bits, the corresponding USB interrupt status bit is set. When this register is read, the same value as the USB interrupt status register is returned.	0	R/W

23.6.12 USB interrupt routing register

Table 342. USB interrupt routing register (INTROUTING, address 0x1C00 C02C) bit description

Bit	Symbol	Description	Reset value	Access
9:0	ROUTE_INT	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W
29:10	-	Reserved	0	RO
30	ROUTE_INT30	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W
31	ROUTE_INT31	This bit can control on which hardware interrupt line the interrupt will be generated: 0: IRQ interrupt line is selected for this interrupt bit 1: FIQ interrupt line is selected for this interrupt bit	0	R/W

23.6.13 USB Endpoint toggle

Table 343. USB Endpoint toggle (EPTOGGLE, address 0x1C00 C034) bit description

Bit	Symbol	Description	Reset value	Access
9:0	TOGGLE	Endpoint data toggle: This field indicates the current value of the data toggle for the corresponding endpoint.	0	R
31:10	-	Reserved	0	R

23.7 Functional description

23.7.1 Endpoint command/status list

Figure 61 gives an overview on how the Endpoint List is organized in memory. The USB EP Command/Status List start register points to the start of the list that contains all the endpoint information in memory. The order of the endpoints is fixed as shown in the picture.

USB EP Command/Status FIFO start

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset
A	R	S	TR	TV	R	EP0 OUT Buffer NBytes										EP0 OUT Buffer Address Offset																	0x00
R	R	R	R	R	R	Reserved										SETUP bytes Buffer Address Offset																	0x04
A	R	S	TR	TV	R	EP0 IN Buffer NBytes										EP0 IN Buffer Address Offset																	0x08
R	R	R	R	R	R	Reserved										Reserved																	0x0C
A	D	S	TR	RF TV	T	EP1 OUT Buffer 0 NBytes										EP1 OUT Buffer 0 Address Offset																	0x10
A	D	S	TR	RF TV	T	EP1 OUT Buffer 1 NBytes										EP1 OUT Buffer 1 Address Offset																	0x14
A	D	S	TR	RF TV	T	EP1 IN Buffer 0 NBytes										EP1 IN Buffer 0 Address Offset																	0x18
A	D	S	TR	RF TV	T	EP1 IN Buffer 1 NBytes										EP1 IN Buffer 1 Address Offset																	0x1C
A	D	S	TR	RF TV	T	EP2 OUT Buffer 0 NBytes										EP2 OUT Buffer 0 Address Offset																	0x20
A	D	S	TR	RF TV	T	EP2 OUT Buffer 1 NBytes										EP2 OUT Buffer 1 Address Offset																	0x24
A	D	S	TR	RF TV	T	EP2 IN Buffer 0 NBytes										EP2 IN Buffer 0 Address Offset																	0x28
A	D	S	TR	RF TV	T	EP2 IN Buffer 1 NBytes										EP2 IN Buffer 1 Address Offset																	0x2C
...																																
A	D	S	TR	RF TV	T	EP4 OUT Buffer 0 NBytes										EP4 OUT Buffer 0 Address Offset																	0x40
A	D	S	TR	RF TV	T	EP4 OUT Buffer 1 NBytes										EP4 OUT Buffer 1 Address Offset																	0x44
A	D	S	TR	RF TV	T	EP4 IN Buffer 0 NBytes										EP4 IN Buffer 0 Address Offset																	0x48
A	D	S	TR	RF TV	T	EP4 IN Buffer 1 NBytes										EP4 IN Buffer 1 Address Offset																	0x4C

Fig 61. Endpoint command/status list (see also Table 344)

Table 344. Endpoint commands

Symbol	Access	Description
A	RW	Active The buffer is enabled. HW can use the buffer to store received OUT data or to transmit data on the IN endpoint. Software can only set this bit to '1'. As long as this bit is set to one, software is not allowed to update any of the values in this 32-bit word. In case software wants to deactivate the buffer, it must write a one to the corresponding "skip" bit in the USB Endpoint skip register. Hardware can only write this bit to zero. It will do this when it receives a short packet or when the NBytes field transitions to zero or when software has written a one to the "skip" bit.
D	RW	Disabled 0: The selected endpoint is enabled. 1: The selected endpoint is disabled. If a USB token is received for an endpoint that has the disabled bit set, hardware will ignore the token and not return any data or handshake. When a bus reset is received, software must set the disable bit of all endpoints to 1. Software can only modify this bit when the active bit is zero.
S	RW	Stall 0: The selected endpoint is not stalled 1: The selected endpoint is stalled The Active bit has always higher priority than the Stall bit. This means that a Stall handshake is only sent when the active bit is zero and the stall bit is one. Software can only modify this bit when the active bit is zero.
TR	RW	Toggle Reset When software sets this bit to one, the HW will set the toggle value equal to the value indicated in the "toggle value" (TV) bit. For the control endpoint zero, this is not needed to be used because the hardware resets the endpoint toggle to one for both directions when a setup token is received. For the other endpoints, the toggle can only be reset to zero when the endpoint is reset.
RF / TV	RW	Rate Feedback mode / Toggle value For bulk endpoints and isochronous endpoints this bit is reserved and must be set to zero. For the control endpoint zero this bit is used as the toggle value. When the toggle reset bit is set, the data toggle is updated with the value programmed in this bit. When the endpoint is used as an interrupt endpoint, it can be set to the following values. 0: Interrupt endpoint in 'toggle mode' 1: Interrupt endpoint in 'rate feedback mode'. This means that the data toggle is fixed to zero for all data packets. When the interrupt endpoint is in 'rate feedback mode', the TR bit must always be set to zero.

Table 344. Endpoint commands

Symbol	Access	Description
T	RW	Endpoint Type 0: Generic endpoint. The endpoint is configured as a bulk or interrupt endpoint 1: Isochronous endpoint
NBytes	RW	For OUT endpoints this is the number of bytes that can be received in this buffer. For IN endpoints this is the number of bytes that must be transmitted. HW decrements this value with the packet size every time when a packet is successfully transferred. Note: If a short packet is received on an OUT endpoint, the active bit will be cleared and the NBytes value indicates the remaining buffer space that is not used. Software calculates the received number of bytes by subtracting the remaining NBytes from the programmed value.
Address Offset	RW	Bits 21 to 6 of the buffer start address. If the endpoint type is set to ‘0’ (generic endpoint) this address is incremented every time a packet has been successfully received/transmitted. If the endpoint type is set to ‘1’ (isochronous endpoint), the address is not incremented.

Remark: When receiving a SETUP token for endpoint zero, the HW will only read the SETUP bytes Buffer Address offset to know where it has to store the received SETUP bytes. The hardware will ignore all other fields. In case the SETUP stage contains more than 8 bytes, it will only write the first 8 bytes to memory. A USB compliant host must never send more than 8 bytes during the SETUP stage.

23.7.2 Control endpoint 0

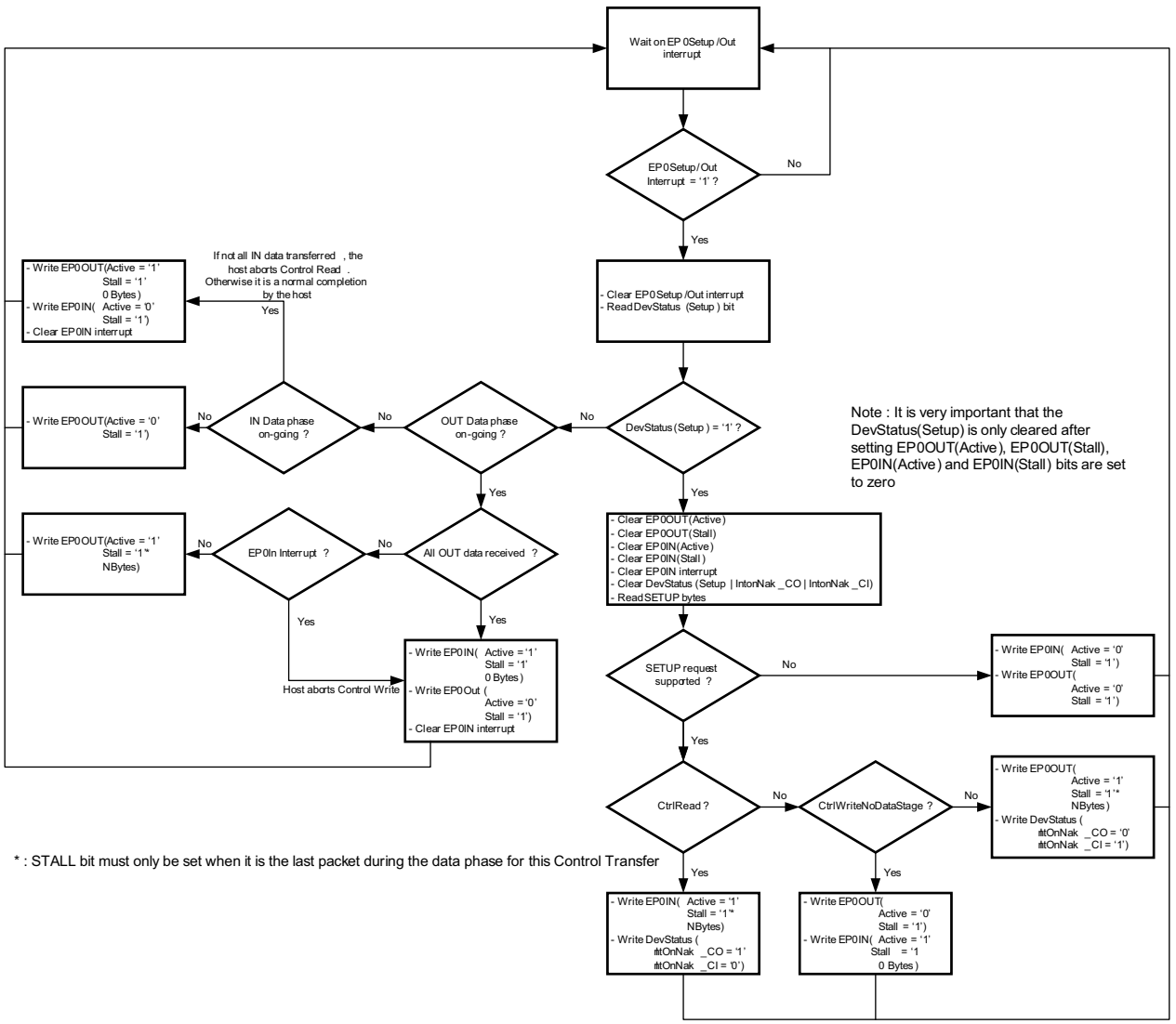
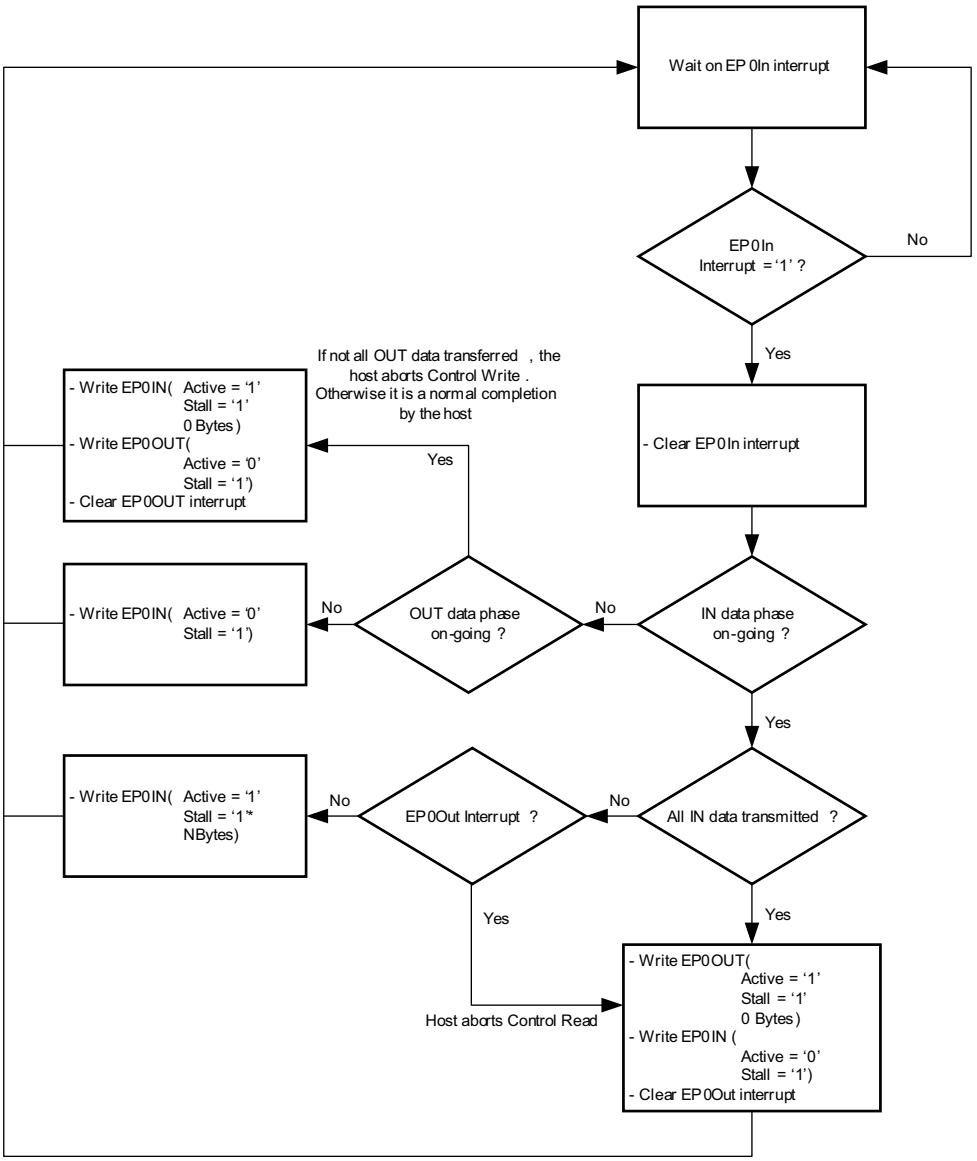


Fig 62. Flowchart of control endpoint 0 - OUT direction



* : STALL bit must only be set when it is the last packet during the data phase for this Control Transfer

Fig 63. Flowchart of control endpoint 0 - IN direction

23.7.3 Generic endpoint: single-buffering

To enable single-buffering, software must set the corresponding "USB EP Buffer Config" bit to zero. In the "USB EP Buffer in use" register, software can indicate which buffer is used in this case.

When software wants to transfer data, it programs the different bits in the Endpoint command/status entry and sets the active bits. The hardware will transmit/receive multiple packets for this endpoint until the NBytes value is equal to zero. When NBytes goes to zero, hardware clears the active bit and sets the corresponding interrupt status bit.

Software must wait until hardware has cleared the Active bit to change some of the command/status bits. This prevents hardware from overwriting a new value programmed by software with some old values that were still cached.

If software wants to disable the active bit before the hardware has finished handling the complete buffer, it can do this by setting the corresponding endpoint skip bit in USB endpoint skip register.

23.7.4 Generic endpoint: double-buffering

To enable double-buffering, software must set the corresponding "USB EP Buffer Config" bit to one. The "USB EP Buffer in use" register indicates which buffer will be used by HW when the next token is received.

When HW clears the active bit of the current buffer in use, it will switch the buffer in use. Software can also force HW to use a certain buffer by writing to the "USB EP Buffer in use" bit.

23.7.5 Special cases

23.7.5.1 Use of the Active bit

The use of the Active bit is a bit different between OUT and IN endpoints.

When data must be received for the OUT endpoint, the software will set the Active bit to one and program the NBytes field to the maximum number of bytes it can receive.

When data must be transmitted for an IN endpoint, the software sets the Active bit to one and programs the NBytes field to the number of bytes that must be transmitted.

23.7.5.2 Generation of a STALL handshake

Special care must be taken when programming the endpoint to send a STALL handshake. A STALL handshake is only sent in the following situations:

- The endpoint is enabled (Disabled bit = 0)
- The active bit of the endpoint is set to 0. (No packet needs to be received/transmitted for that endpoint).
- The stall bit of the endpoint is set to one.

23.7.5.3 Clear Feature (endpoint halt)

When a non-control endpoint has returned a STALL handshake, the host will send a Clear Feature (Endpoint Halt) for that endpoint. When the device receives this request, the endpoint must be unstalled and the toggle bit for that endpoint must be reset back to zero. In order to do that the software must program the following items for the endpoint that is indicated.

If the endpoint is used in single-buffer mode, program the following:

- Set STALL bit (S) to 0.
- Set toggle reset bit (TR) to 1 and set toggle value bit (TV) to 0.

If the endpoint is used in double-buffer mode, program the following:

- Set the STALL bit of both buffer 0 and buffer 1 to 0.
- Read the buffer in use bit for this endpoint.
- Set the toggle reset bit (TR) to 1 and set the toggle value bit (TV) to 0 for the buffer indicated by the buffer in use bit.

23.7.5.4 Set configuration

When a SetConfiguration request is received with a configuration value different from zero, the device software must enable all endpoints that will be used in this configuration and reset all the toggle values. To do so, it must generate the procedure explained in Section 23.7.5.3 for every endpoint that will be used in this configuration.
For all endpoints that are not used in this configuration, it must set the Disabled bit (D) to one.

23.7.6 USB wake-up

23.7.6.1 Waking up from Deep-sleep and Power-down modes on USB activity

To allow the part to wake up from Deep-sleep or Power-down mode on USB activity, complete the following steps:

1. Set bit AP_CLK in the USBCLKCTRL register (Table 62) to 0 (default) to enable automatic control of the USB need_clock signal.
2. Wait until USB activity is suspended by polling the DSUS bit in the DSVCMD_STAT register (DSUS = 1).
3. The USB need_clock signal will be deasserted after another 2 ms. Poll the USBCLKST register until the USB need_clock status bit is 0 (Table 63).
4. Once the USBCLKST register returns 0, enable the USB activity wake-up interrupt in the NVIC (# 30) and clear it.
5. Set bit 1 in the USBCLKCTRL register to 1 to trigger the USB activity wake-up interrupt on the rising edge of the USB need_clock signal.
6. Enable the wake-up from Deep-sleep or Power-down modes on this interrupt by enabling the USB need_clock signal in the STARTERP0 register (Table 76, bit 30).
7. Enter Deep-sleep or Power-down modes by writing to the power configuration API.
8. Execute a WFI instruction.

The part will automatically wake up and resume execution on USB activity.

23.7.6.2 Remote wake-up

To issue a remote wake-up when the USB activity is suspended, complete the following steps:

1. Set bit AP_CLK in the USBCLKCTRL register to 0 (Table 62, default) to enable automatic control of the USB need_clock signal.
2. When it is time to issue a remote wake-up, turn on the USB clock and enable the USB clock source.
3. Force the USB clock on by writing a 1 to bit AP_CLK (Table 62, bit 0) in the USBCLKCTRL register.
4. Write a 0 to the DSUS bit in the DSVCMD_STAT register.

5. Wait until the USB leaves the suspend state by polling the DSUS bit in the DSVCMD_STAT register (DSUS =0).
6. Clear the AP_CLK bit (Table 62, bit 0) in the USBCLKCTRL to enable automatic USB clock control.

24.1 How to read this chapter

Read this chapter for a description of the USART peripheral and the software interface.

The LPC15xx also supports an on-chip ROM-based USART API to configure and operate the USART. See Chapter 36.

24.2 Features

- 7, 8, or 9 data bits and 1 or 2 stop bits
- Synchronous mode with master or slave operation. Includes data phase selection and continuous clock option.
- Multiprocessor/multidrop (9-bit) mode with software address compare.
- RS-485 transceiver output enable.
- Parity generation and checking: odd, even, or none.
- Software selectable oversampling from 5 to 16 clocks in asynchronous mode.
- One transmit and one receive data buffer.
- RTS/CTS for hardware signaling for automatic flow control. Software flow control can be performed using Delta CTS detect, Transmit Disable control, and any GPIO as an RTS output.
- Received data and status can optionally be read from a single register
- Break generation and detection.
- Receive data is 2 of 3 sample "voting". Status flag set when one sample differs.
- Built-in Baud Rate Generator with auto-baud function.
- A fractional rate divider is shared among all USARTs.
- Interrupts available for Receiver Ready, Transmitter Ready, Receiver Idle, change in receiver break detect, Framing error, Parity error, Overrun, Underrun, Delta CTS detect, and receiver sample noise detected.
- Loopback mode for testing of data and flow control.
- UART transmit and receive functions can operated with the system DMA controller.
- Special operating mode allows operation at up to 9600 baud using the 32 kHz RTC oscillator as the UART clock. This mode can be used while the device is in Power-down mode and can wake-up the device when a character is received.
- UARTn transmit and receive functions can operated with the system DMA controller.

24.3 Basic configuration

Remark: The on-chip USART API provides software routines to configure and use the USART. See Chapter 36.

Configure USART0/1/2 for receiving and transmitting data:

- In the SYSAHBCLKCTRL1 register, set bit 17 to 19 (Table 51) to enable the clock to the register interface.
- Clear the USART0/1/2 peripheral resets using the PRESETCTRL1 register (Table 36).
- Enable or disable the USART0/1/2 interrupts in slots #21 to 23 in the NVIC.
- Configure the USART0/1/2 pin functions through the switch matrix. See Section 24.4.
- Configure the USART clock and baud rate. See Section 24.3.1.
- Send and receive lines are connected to DMA request lines. See Table 167.

Configure the USART0/1/2 to wake up the part from low power modes:

- Configure the USART to receive and transmit data in synchronous slave mode. See Section 24.3.2.

24.3.1 Configure the USART clock and baud rate

All three USARTs use a common peripheral clock (U_PCLK) and, if needed, a fractional baud rate generator. The peripheral clock and the fractional divider for the baud rate calculation are set up in the SYSCON block as follows (see Figure 64):

1. Configure the UART clock by writing a value UARTCLKDIV > 0 in the USART peripheral clock divider register. This is the divided main clock common to all USARTs.

   Section 3.6.25 “USART clock divider register”
2. If a fractional value is needed to obtain a particular baud rate, program the fractional divider. The fractional divider value is the fraction of MULT/DIV. The MULT and DIV values are programmed in the FRGCTRL register. The DIV value must be programmed with the fixed value of 256.

   U_PCLK = UARTCLKDIV/(1+(MULT/DIV))

   The following rules apply for MULT and DIV:

   – Always set DIV to 256 by programming the FRGCTRL register with the value of 0xFF.

   – Set the MULT to any value between 0 and 255.

   Table 61 “USART fractional baud rate generator register (FRGCTRL, address 0x4007 4128) bit description”
3. In asynchronous mode: Configure the baud rate divider BRGVAL in the USARTn BRG register. The baud rate divider divides the common USART peripheral clock by a factor of 16 multiplied by the baud rate value to provide the baud rate = U_PCLK/16 x BRGVAL.

   Section 24.6.9 “USART Baud Rate Generator register”
4. In synchronous mode: The serial clock is Un_SCLK = U_PCLK/BRGVAL.

The USART can also be clocked by the 32 kHz RTC oscillator. Set the MODE32K bit to enable this 32 kHz mode. See also Section 24.7.1.4 “32 kHz mode”.

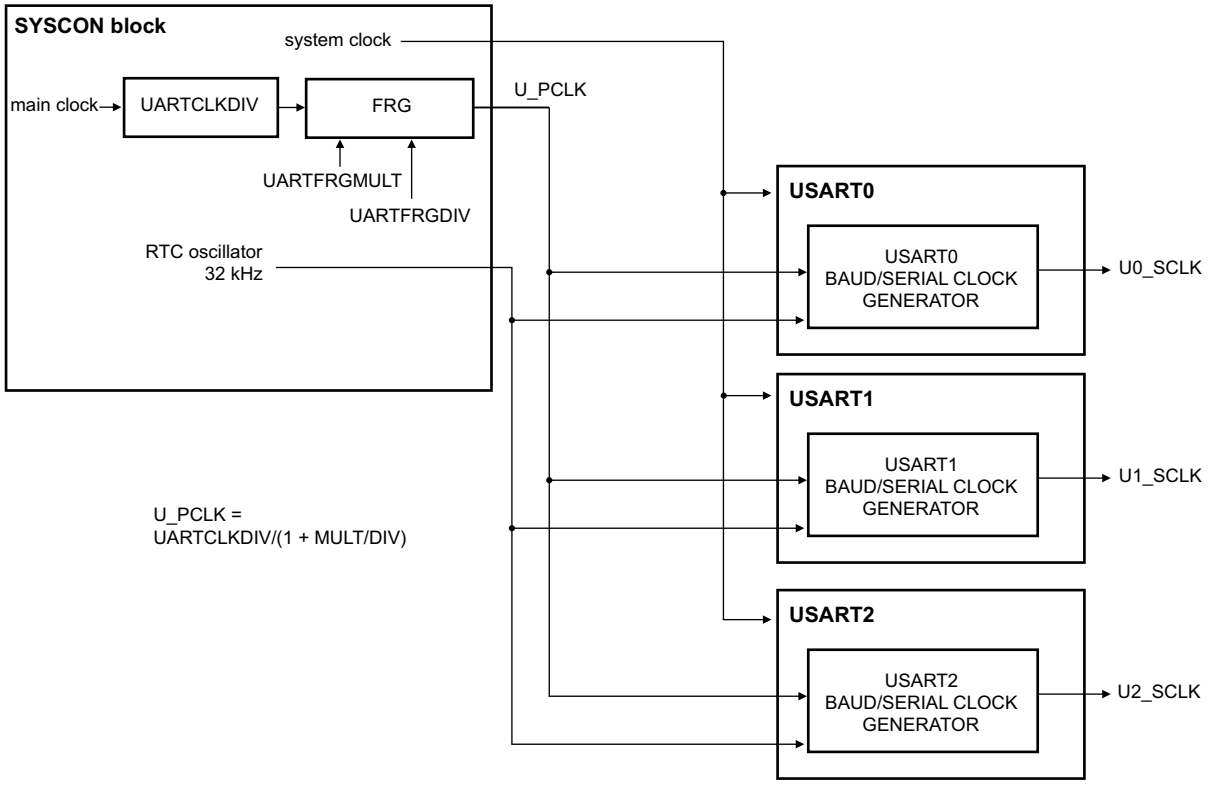


Fig 64. USART clocking

For details on the clock configuration see:

Section 24.7.1 “Clocking and baud rates”

24.3.2 Configure the USART for wake-up

The USART can wake up the system from sleep mode in asynchronous or synchronous mode on any enabled USART interrupt.

In Deep-sleep or power-down mode, you have two options for configuring USART for wake-up:

- If the USART is configured for synchronous slave mode, the USART block can create an interrupt on a received signal even when the USART block receives no clocks from the ARM Cortex-M3 core - that is in Deep-sleep or Power-down mode.

  As long as the USART receives a clock signal from the master, it can receive up to one byte in the RXDAT register while in Deep-sleep or Power-down mode. Any interrupt raised as part of the receive data process can then wake up the part.
- If the 32 kHz mode is enabled, the USART can run in asynchronous mode using the 32 kHz RTC oscillator and create interrupts.

24.3.2.1 Wake-up from Sleep mode

- Configure the USART in either asynchronous mode or synchronous mode. See Table 347.
- Enable the USART interrupt in the NVIC.
- Any USART interrupt wakes up the part from sleep mode. Enable the USART interrupt in the INTENSET register (Table 350).

24.3.2.2 Wake-up from Deep-sleep or Power-down mode

- Configure the USART in synchronous slave mode. See Table 347. You must connect the SCLK function to a pin and connect the pin to the master. Alternatively, you can enable the 32 kHz mode and use the USART in asynchronous mode with the 32 kHz RTC oscillator.
- Enable the USART interrupt in the STARTERP1 register. See Table 76 “Start logic 0 wake-up enable register 0 (STARTERP0, address 0x4007 4218) bit description”.
- Enable the USART interrupt in the NVIC.
- In the PDAWAKE register, configure all peripherals that need to be running when the part wakes up.
- The USART wakes up the part from Deep-sleep or Power-down mode on all events that cause an interrupt and are also enabled in the INTENSET register. Typical wake-up events are:
  - A start bit has been received.
  - The RXDAT buffer has received a byte.
  - Data is ready to be transmitted in the TXDAT buffer and a serial clock from the master has been received.
  - A change in the state of the CTS pin if the CTS function is connected.
  - **Remark:** By enabling or disabling the interrupt in the INTENSET register (Table 350), you can customize when the wake-up occurs in the USART receive/transmit protocol.

24.4 Pin description

The USART receive, transmit, and control signals are movable functions and are assigned to external pins through the switch matrix.

See Section 8.3.1 “Connect an internal signal to a package pin” to assign the USART functions to pins on the LPC15xx package.

Table 345. USART pin description

Function	Direction	Pin	Description	SWM register	Reference
U0_TXD	O	any	Transmitter output for USART0. Serial transmit data.	PINASSIGN0	Table 107
U0_RXD	I	any	Receiver input for USART0. Serial receive data.	PINASSIGN0	Table 107
$\overline{\text{U0\_RTS}}$	O	any	Request To Send output for USART0. This signal supports inter-processor communication through the use of hardware flow control. This feature is active when the USART RTS signal is configured to appear on a device pin.	PINASSIGN0	Table 107

Table 345. USART pin description

Function	Direction	Pin	Description	SWM register	Reference
$\overline{\text{U0\_CTS}}$	I	any	Clear To Send input for USART0. Active low signal indicates that the external device that is in communication with the USART is ready to accept data. This feature is active when enabled by the CTSEn bit in CFG register and when configured to appear on a device pin. When deasserted (high) by the external device, the USART will complete transmitting any character already in progress, then stop until CTS is again asserted (low).	PINASSIGN0	Table 107
U0_SCLK	I/O	any	Serial clock input/output for USART0 in synchronous mode. Clock input or output in synchronous mode.	PINASSIGN1	Table 108
U1_TXD	O	any	Transmitter output for USART1. Serial transmit data.	PINASSIGN1	Table 108
U1_RXD	I	any	Receiver input for USART1.	PINASSIGN1	Table 108
$\overline{\text{U1\_RTS}}$	O	any	Request To Send output for USART1.	PINASSIGN1	Table 108
$\overline{\text{U1\_CTS}}$	I	any	Clear To Send input for USART1.	PINASSIGN2	Table 109
U1_SCLK	I/O	any	Serial clock input/output for USART1 in synchronous mode.	PINASSIGN2	Table 109
U2_TXD	O	any	Transmitter output for USART2. Serial transmit data.	PINASSIGN2	Table 109
U2_RXD	I	any	Receiver input for USART2.	PINASSIGN2	Table 109
U2_SCLK	I/O	any	Serial clock input/output for USART2 in synchronous mode.	PINASSIGN3	Table 110

24.5 General description

The USART receiver block monitors the serial input line, Un_RXD, for valid input. The receiver shift register assembles characters as they are received, after which they are passed to the receiver buffer register to await access by the CPU or the DMA controller.

When RTS signal is configured as an RS-485 output enable, it is asserted at the beginning of an transmitted character, and deasserted either at the end of the character, or after a one character delay (selected by software).

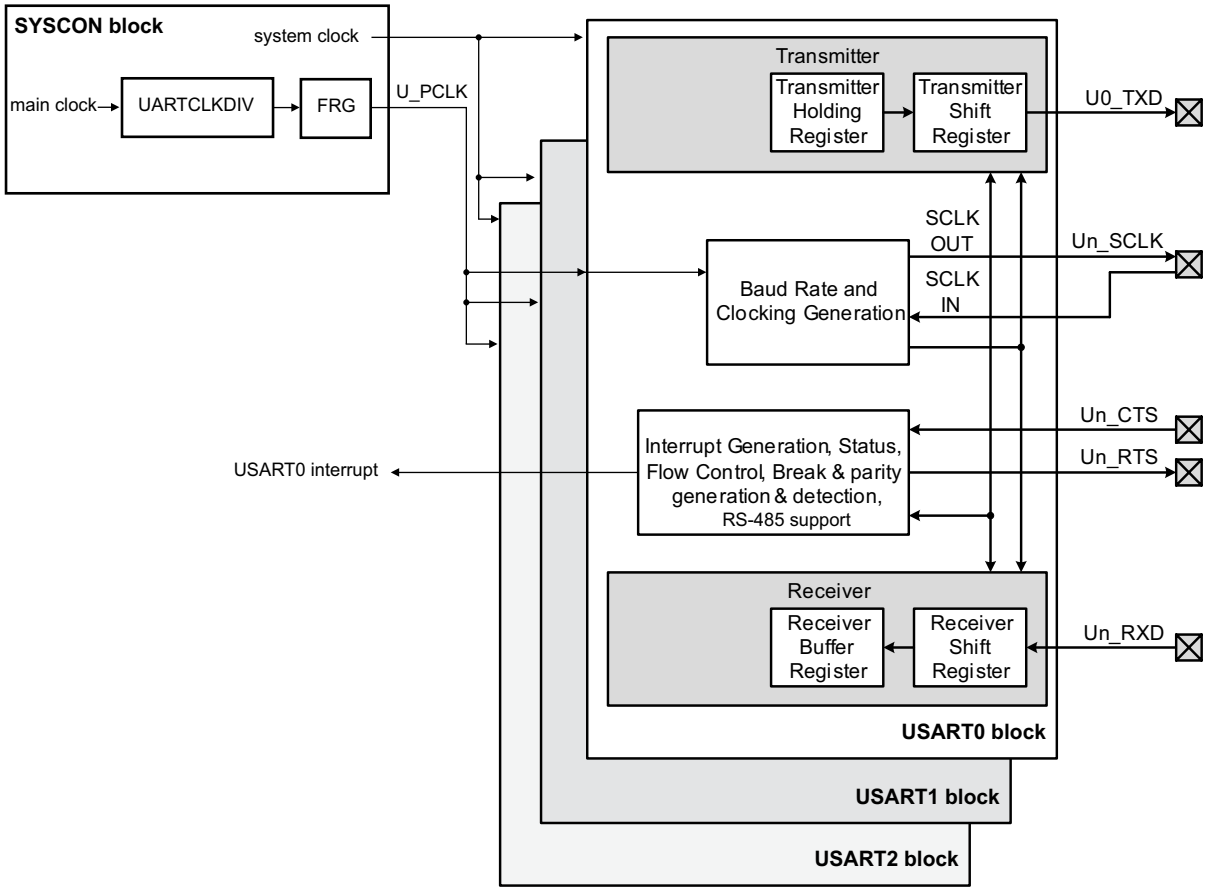
The USART transmitter block accepts data written by the CPU or DMA controllers and buffers the data in the transmit holding register. When the transmitter is available, the transmit shift register takes that data, formats it, and serializes it to the serial output, Un_TXD.

The Baud Rate Generator block divides the incoming clock to create a 16x baud rate clock in the standard asynchronous operating mode. The BRG clock input source is the shared Fractional Rate Generator that runs from the common USART peripheral clock U_PCLK). The 32 kHz operating mode generates a specially timed internal clock based on the RTC oscillator frequency.

In synchronous slave mode, data is transmitted and received using the serial clock directly. In synchronous master mode, data is transmitted and received using the baud rate clock without division.

Status information from the transmitter and receiver is saved and provided via the Stat register. Many of the status flags are able to generate interrupts, as selected by software.

Remark: The fractional value and the USART peripheral clock are shared between all USARTs.



U_PCLK = UARTCLKDIV/(1+MULT/DIV)

Fig 65. USART block diagram

24.6 Register description

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

Table 346: Register overview: USART (base address 0x4004 0000 (USART0), 0x4004 4000 (USART1), 0x400C 0000 (USART2))

Name	Access	Offset	Description	Reset value	Reference
CFG	R/W	0x000	USART Configuration register. Basic USART configuration settings that typically are not changed during operation.	0	Table 347
CTL	R/W	0x004	USART Control register. USART control settings that are more likely to change during operation.	0	Table 348
STAT	R/W	0x008	USART Status register. The complete status value can be read here. Writing ones clears some bits in the register. Some bits can be cleared by writing a 1 to them.	0x000E	Table 349
INTENSET	R/W	0x00C	Interrupt Enable read and Set register. Contains an individual interrupt enable bit for each potential USART interrupt. A complete value may be read from this register. Writing a 1 to any implemented bit position causes that bit to be set.	0	Table 350
INTENCLR	W	0x010	Interrupt Enable Clear register. Allows clearing any combination of bits in the INTENSET register. Writing a 1 to any implemented bit position causes the corresponding bit to be cleared.	-	Table 351
RXDAT	R	0x014	Receiver Data register. Contains the last character received.	-	Table 352
RXDATSTAT	R	0x018	Receiver Data with Status register. Combines the last character received with the current USART receive status. Allows DMA or software to recover incoming data and status together.	-	Table 353
TXDAT	R/W	0x01C	Transmit Data register. Data to be transmitted is written here.	0	Table 354
BRG	R/W	0x020	Baud Rate Generator register. 16-bit integer baud rate divisor value.	0	Table 355
INTSTAT	R	0x024	Interrupt status register. Reflects interrupts that are currently enabled.	0x0005	Table 356
OSR	R/W	0x028	Oversample selection register for asynchronous communication.	0xF	Table 357
ADDR	R/W	0x02C	Address register for automatic address matching.	0	Table 358

24.6.1 USART Configuration register

The CFG register contains communication and mode settings for aspects of the USART that would normally be configured once in an application.

Remark: If software needs to change configuration values, the following sequence should be used: 1) Make sure the USART is not currently sending or receiving data. 2) Disable the USART by writing a 0 to the Enable bit (0 may be written to the entire register). 3) Write the new configuration value, with the ENABLE bit set to 1.

Table 347. USART Configuration register (CFG, address 0x4004 0000 (USART0), 0x4004 4000 (USART1), 0x400C 0000 (USART2)) bit description

Bit	Symbol	Value	Description	Reset Value
0	ENABLE		USART Enable.	0
		0	Disabled. The USART is disabled and the internal state machine and counters are reset. While Enable = 0, all USART interrupts and DMA transfers are disabled. When Enable is set again, CFG and most other control bits remain unchanged. For instance, when re-enabled, the USART will immediately generate a TxRdy interrupt (if enabled in the INTENSET register) or a DMA transfer request because the transmitter has been reset and is therefore available.	
		1	Enabled. The USART is enabled for operation.	
1	-		Reserved. Read value is undefined, only zero should be written.	NA
3:2	DATALEN		Selects the data size for the USART.	00
		0x0	7 bit Data length.	
		0x1	8 bit Data length.	
		0x2	9 bit data length. The 9th bit is commonly used for addressing in multidrop mode. See the ADDRDET bit in the CTL register.	
		0x3	Reserved.	
5:4	PARITYSEL		Selects what type of parity is used by the USART.	00
		0x0	No parity.	
		0x1	Reserved.	
		0x2	Even parity. Adds a bit to each character such that the number of 1s in a transmitted character is even, and the number of 1s in a received character is expected to be even.	
		0x3	Odd parity. Adds a bit to each character such that the number of 1s in a transmitted character is odd, and the number of 1s in a received character is expected to be odd.	
6	STOPLEN		Number of stop bits appended to transmitted data. Only a single stop bit is required for received data.	0
		0	1 stop bit.	
		1	2 stop bits. This setting should only be used for asynchronous communication.	

Table 347. USART Configuration register (CFG, address 0x4004 0000 (USART0), 0x4004 4000 (USART1), 0x400C 0000 (USART2)) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
7	MODE32K		Selects standard or 32 kHz clocking mode.	0
		0	UART uses standard clocking.	
		1	UART uses the 32 kHz clock from the RTC oscillator as the clock source to the BRG, and uses a special bit clocking scheme.	
8	-		Reserved. Read value is undefined, only zero should be written.	NA
9	CTSEN		CTS Enable. Determines whether CTS is used for flow control. CTS can be from the input pin, or from the USART's own RTS if loopback mode is enabled. See Section 24.7.4 for more information.	0
		0	No flow control. The transmitter does not receive any automatic flow control signal.	
		1	Flow control enabled. The transmitter uses the CTS input (or RTS output in loopback mode) for flow control purposes.	
10	-		Reserved. Read value is undefined, only zero should be written.	NA
11	SYNCEN		Selects synchronous or asynchronous operation.	0
		0	Asynchronous mode is selected.	
		1	Synchronous mode is selected.	
12	CLKPOL		Selects the clock polarity and sampling edge of received data in synchronous mode.	0
		0	Falling edge. Un_RXD is sampled on the falling edge of SCLK.	
		1	Rising edge. Un_RXD is sampled on the rising edge of SCLK.	
13	-		Reserved. Read value is undefined, only zero should be written.	NA
14	SYNCMST		Synchronous mode Master select.	0
		0	Slave. When synchronous mode is enabled, the USART is a slave.	
		1	Master. When synchronous mode is enabled, the USART is a master.	
15	LOOP		Selects data loopback mode.	0
		0	Normal operation.	
		1	Loopback mode. This provides a mechanism to perform diagnostic loopback testing for USART data. Serial data from the transmitter (Un_TXD) is connected internally to serial input of the receive (Un_RXD). Un_TXD and Un_RTS activity will also appear on external pins if these functions are configured to appear on device pins. The receiver RTS signal is also looped back to CTS and performs flow control if enabled by CTSEN.	
17:16	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 347. USART Configuration register (CFG, address 0x4004 0000 (USART0), 0x4004 4000 (USART1), 0x400C 0000 (USART2)) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
18	OETA		Output Enable Turnaround time enable for RS-485 operation.	0
		0	Disabled. If selected by OESEL, the Output Enable signal deasserted at the end of the last stop bit of a transmission.	
		1	Enabled. If selected by OESEL, the Output Enable signal remains asserted for one character time after the end of the last stop bit of a transmission. OE will also remain asserted if another transmit begins before it is deasserted.	
19	AUTOADDR		Automatic Address matching enable.	0
		0	Disabled. When addressing is enabled by ADDRDET, address matching is done by software. This provides the possibility of versatile addressing (e.g. respond to more than one address).	
		1	Enabled. When addressing is enabled by ADDRDET, address matching is done by hardware, using the value in the ADDR register as the address to match.	
20	OESEL		Output Enable Select.	0
		0	Standard. The RTS signal is used as the standard flow control function.	
		1	RS-485. The RTS signal configured to provide an output enable signal to control an RS-485 transceiver.	
21	OEPOL		Output Enable Polarity.	0
		0	Low. If selected by OESEL, the output enable is active low.	
		1	High. If selected by OESEL, the output enable is active high.	
22	RXPOL		Receive data polarity.	0
		0	Standard. The RX signal is used as it arrives from the pin. This means that the RX rest value is 1, start bit is 0, data is not inverted, and the stop bit is 1.	
		1	Inverted. The RX signal is inverted before being used by the UART. This means that the RX rest value is 0, start bit is 1, data is inverted, and the stop bit is 0.	
23	TXPOL		Transmit data polarity.	0
		0	Standard. The TX signal is sent out without change. This means that the TX rest value is 1, start bit is 0, data is not inverted, and the stop bit is 1.	
		1	Inverted. The TX signal is inverted by the UART before being sent out. This means that the TX rest value is 0, start bit is 1, data is inverted, and the stop bit is 0.	
31:24	-		Reserved. Read value is undefined, only zero should be written.	NA

24.6.2 USART Control register

The CTL register controls aspects of USART operation that are more likely to change during operation.

Table 348. USART Control register (CTL, address 0x4004 0004 (USART0), 0x4004 4004 (USART1), 0x400C 0004 (USART2)) bit description

Bit	Symbol	Value	Description	Reset Value
0	-		Reserved. Read value is undefined, only zero should be written.	NA
1	TXBRKEN		Break Enable.	0
		0	Normal operation.	
		1	Continuous break is sent immediately when this bit is set, and remains until this bit is cleared. A break may be sent without danger of corrupting any currently transmitting character if the transmitter is first disabled (TXDIS in CTL is set) and then waiting for the transmitter to be disabled (TXDISINT in STAT = 1) before writing 1 to TXBRKEN.	
2	ADDRDET		Enable address detect mode.	0
		0	Disabled. The USART presents all incoming data.	
		1	Enabled. The USART receiver ignores incoming data that does not have the most significant bit of the data (typically the 9th bit) = 1. When the data MSB bit = 1, the receiver treats the incoming data normally, generating a received data interrupt. Software can then check the data to see if this is an address that should be handled. If it is, the ADDRDET bit is cleared by software and further incoming data is handled normally.	
5:3	-		Reserved. Read value is undefined, only zero should be written.	NA
6	TXDIS		Transmit Disable.	0
		0	Not disabled. USART transmitter is not disabled.	
		1	Disabled. USART transmitter is disabled after any character currently being transmitted is complete. This feature can be used to facilitate software flow control.	
7	-		Reserved. Read value is undefined, only zero should be written.	NA
8	CC		Continuous Clock generation. By default, SCLK is only output while data is being transmitted in synchronous mode.	0
		0	Clock on character. In synchronous mode, SCLK cycles only when characters are being sent on Un_TXD or to complete a character that is being received.	
		1	Continuous clock. SCLK runs continuously in synchronous mode, allowing characters to be received on Un_RxD independently from transmission on Un_TXD).	
9	CLRCCONRX		Clear Continuous Clock.	0
		0	No affect on the CC bit.	
		1	Auto-clear. The CC bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.	
15:10	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 348. USART Control register (CTL, address 0x4004 0004 (USART0), 0x4004 4004 (USART1), 0x400C 0004 (USART2)) bit description

Bit	Symbol	Value	Description	Reset Value
16	AUTOBAUD		Autobaud enable.	0
		0	Disabled. UART is in normal operating mode.	
		1	Enabled. UART is in autobaud mode. This bit should only be set when the UART receiver is idle. The first start bit of RX is measured and used the update the BRG register to match the received data rate. AUTOBAUD is cleared once this process is complete, or if there is an AERR.	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

24.6.3 USART Status register

The STAT register primarily provides a complete set of USART status flags for software to read. Flags other than read-only flags may be cleared by writing ones to corresponding bits of STAT. Interrupt status flags that are read-only and cannot be cleared by software, can be masked using the INTENCLR register (see Table 351).

The error flags for received noise, parity error, framing error, and overrun are set immediately upon detection and remain set until cleared by software action in STAT.

Table 349. USART Status register (STAT, address 0x4004 0008 (USART0), 0x4004 4008 (USART1), 0x400C 0008 (USART2)) bit description

Bit	Symbol	Description	Reset value	Access [1]
0	RXRDY	Receiver Ready flag. When 1, indicates that data is available to be read from the receiver buffer. Cleared after a read of the RXDAT or RXDATSTAT registers.	0	RO
1	RXIDLE	Receiver Idle. When 0, indicates that the receiver is currently in the process of receiving data. When 1, indicates that the receiver is not currently in the process of receiving data.	1	RO
2	TXRDY	Transmitter Ready flag. When 1, this bit indicates that data may be written to the transmit buffer. Previous data may still be in the process of being transmitted. Cleared when data is written to TXDAT. Set when the data is moved from the transmit buffer to the transmit shift register.	1	RO
3	TXIDLE	Transmitter Idle. When 0, indicates that the transmitter is currently in the process of sending data.When 1, indicate that the transmitter is not currently in the process of sending data.	1	RO
4	CTS	This bit reflects the current state of the CTS signal, regardless of the setting of the CTSEN bit in the CFG register. This will be the value of the CTS input pin unless loopback mode is enabled.	NA	RO
5	DELTACTS	This bit is set when a change in the state is detected for the CTS flag above. This bit is cleared by software.	0	W1
6	TXDISSTAT	Transmitter Disabled Status flag. When 1, this bit indicates that the UART transmitter is fully idle after being disabled via the TXDIS bit in the CFG register (TXDIS = 1).	0	RO
7	-	Reserved. Read value is undefined, only zero should be written.	NA	NA

Table 349. USART Status register (STAT, address 0x4004 0008 (USART0), 0x4004 4008 (USART1), 0x400C 0008 (USART2)) bit description

Bit	Symbol	Description	Reset value	Access [1]
8	OVERRUNINT	Overrun Error interrupt flag. This flag is set when a new character is received while the receiver buffer is still in use. If this occurs, the newly received character in the shift register is lost.	0	W1
9	-	Reserved. Read value is undefined, only zero should be written.	NA	NA
10	RXBRK	Received Break. This bit reflects the current state of the receiver break detection logic. It is set when the Un_RXD pin remains low for 16 bit times. Note that FRAMERRINT will also be set when this condition occurs because the stop bit(s) for the character would be missing. RXBRK is cleared when the Un_RXD pin goes high.	0	RO
11	DELTARXBRK	This bit is set when a change in the state of receiver break detection occurs. Cleared by software.	0	W1
12	START	This bit is set when a start is detected on the receiver input. Its purpose is primarily to allow wake-up from Deep-sleep or Power-down mode immediately when a start is detected. Cleared by software.	0	W1
13	FRAMERRINT	Framing Error interrupt flag. This flag is set when a character is received with a missing stop bit at the expected location. This could be an indication of a baud rate or configuration mismatch with the transmitting source.	0	W1
14	PARITYERRINT	Parity Error interrupt flag. This flag is set when a parity error is detected in a received character..	0	W1
15	RXNOISEINT	Received Noise interrupt flag. Three samples of received data are taken in order to determine the value of each received data bit, except in synchronous mode. This acts as a noise filter if one sample disagrees. This flag is set when a received data bit contains one disagreeing sample. This could indicate line noise, a baud rate or character format mismatch, or loss of synchronization during data reception.	0	W1
16	ABERR	Auto baud Error. An auto baud error can occur if the BRG counts to its limit before the end of the start bit that is being measured, essentially an auto baud time-out.	0	W1
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA	NA

[1] RO = Read-only, W1 = write 1 to clear.

24.6.4 USART Interrupt Enable read and set register

The INTENSET register is used to enable various USART interrupt sources. Enable bits in INTENSET are mapped in locations that correspond to the flags in the STAT register. The complete set of interrupt enables may be read from this register. Writing ones to implemented bits in this register causes those bits to be set. The INTENCLR register is used to clear bits in this register.

Table 350. USART Interrupt Enable read and set register (INTENSET, address 0x4004 000C(USART0), 0x4004 400C (USART1), 0x400C 000C (USART2)) bit description

Bit	Symbol	Description	Reset Value
0	RXRDYEN	When 1, enables an interrupt when there is a received character available to be read from the RXDAT register.	0
1	-	Reserved. Read value is undefined, only zero should be written.	NA
2	TXRDYEN	When 1, enables an interrupt when the TXDAT register is available to take another character to transmit.	0
3	TXIDLEEN	When 1, enables an interrupt when the transmitter becomes idle (TXIDLE = 1).	0
4	-	Reserved. Read value is undefined, only zero should be written.	NA
5	DELTACTSEN	When 1, enables an interrupt when there is a change in the state of the CTS input.	0
6	TXDISEN	When 1, enables an interrupt when the transmitter is fully disabled as indicated by the TXDISINT flag in STAT. See description of the TXDISINT bit for details.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	OVERRUNEN	When 1, enables an interrupt when an overrun error occurred.	0
10:9	-	Reserved. Read value is undefined, only zero should be written.	NA
11	DELTARXBRKEN	When 1, enables an interrupt when a change of state has occurred in the detection of a received break condition (break condition asserted or deasserted).	0
12	STARTEN	When 1, enables an interrupt when a received start bit has been detected.	0
13	FRAMERREN	When 1, enables an interrupt when a framing error has been detected.	0
14	PARITYERREN	When 1, enables an interrupt when a parity error has been detected.	0
15	RXNOISEEN	When 1, enables an interrupt when noise is detected. See description of the RXNOISEINT bit in Table 349.	0
16	ABERREN	When 1, enables an interrupt when an auto baud error occurs.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

24.6.5 USART Interrupt Enable Clear register

The INTENCLR register is used to clear bits in the INTENSET register.

Table 351. USART Interrupt Enable clear register (INTENCLR, address 0x4000 0010 (USART0), 0x4004 4010 (USART1), 0x400C 0010 (USART2)) bit description

Bit	Symbol	Description	Reset Value
0	RXRDYCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
1	-	Reserved. Read value is undefined, only zero should be written.	NA
2	TXRDYCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
3	TXIDLECLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
4	-	Reserved. Read value is undefined, only zero should be written.	NA
5	DELTACTSCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
6	TXDISCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	OVERRUNCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
10:9	-	Reserved. Read value is undefined, only zero should be written.	NA
11	DELTARXBRKCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
12	STARTCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
13	FRAMERRCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
14	PARITYERRCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
15	RXNOISECLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
16	ABERRCLR	Writing 1 clears the corresponding bit in the INTENSET register.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

24.6.6 USART Receiver Data register

The RXDAT register contains the last character received before any overrun.

Remark: Reading this register changes the status flags in the RXDATSTAT register.

Table 352. USART Receiver Data register (RXDAT, address 0x4004 0014 (USART0), 0x4004 4014 (USART1), 0x400C 0014 (USART2)) bit description

Bit	Symbol	Description	Reset Value
8:0	DATA	The USART Receiver Data register contains the next received character. The number of bits that are relevant depends on the USART configuration settings.	0
31:9	-	Reserved, the value read from a reserved bit is not defined.	NA

24.6.7 USART Receiver Data with Status register

The RXDATSTAT register contains the next complete character to be read and its relevant status flags. This allows getting all information related to a received character with one 16-bit read, which may be especially useful when the DMA is used with the USART receiver.

Remark: Reading this register changes the status flags.

Table 353. USART Receiver Data with Status register (RXDATSTAT, address 0x4004 0018 (USART0), 0x4004 4018 (USART1), 0x400C 0018 (USART2)) bit description

Bit	Symbol	Description	Reset Value
8:0	RXDATA	The USART Receiver Data register contains the next received character. The number of bits that are relevant depends on the USART configuration settings.	0
12:9	-	Reserved, the value read from a reserved bit is not defined.	NA
13	FRAMERR	Framing Error status flag. This bit is valid when there is a character to be read in the RXDAT register and reflects the status of that character. This bit will set when the character in RXDAT was received with a missing stop bit at the expected location. This could be an indication of a baud rate or configuration mismatch with the transmitting source.	0
14	PARITYERR	Parity Error status flag. This bit is valid when there is a character to be read in the RXDAT register and reflects the status of that character. This bit will be set when a parity error is detected in a received character.	0
15	RXNOISE	Received Noise flag. See description of the RxNoiseInt bit in Table 349.	0
31:16	-	Reserved, the value read from a reserved bit is not defined.	NA

24.6.8 USART Transmitter Data Register

The TXDAT register is written in order to send data via the USART transmitter. That data will be transferred to the transmit shift register when it is available, and another character may then be written to TXDAT.

Table 354. USART Transmitter Data Register (TXDAT, address 0x4004 001C (USART0), 0x4004 401C (USART1), 0x400C 001C (USART2)) bit description

Bit	Symbol	Description	Reset Value
8:0	TXDATA	Writing to the USART Transmit Data Register causes the data to be transmitted as soon as the transmit shift register is available and any conditions for transmitting data are met: CTS low (if CTSEN bit = 1), TXDIS bit = 0.	0
31:9	-	Reserved. Only zero should be written.	NA

24.6.9 USART Baud Rate Generator register

The Baud Rate Generator is a simple 16-bit integer divider controlled by the BRG register. The BRG register contains the value used to divide the base clock in order to produce the clock used for USART internal operations.

A 16-bit value allows producing standard baud rates from 300 baud and lower at the highest frequency of the device, up to 921,600 baud from a base clock as low as 14.7456 MHz.

Typically, the baud rate clock is 16 times the actual baud rate. This overclocking allows for centering the data sampling time within a bit cell, and for noise reduction and detection by taking three samples of incoming data.

Note that in 32 kHz mode, the baud rate generator is still used and must be set to 0 if 9600 baud is required.

Details on how to select the right values for BRG can be found later in this chapter, see Section 24.7.1.

Remark: If software needs to change the baud rate, the following sequence should be used: 1) Make sure the USART is not currently sending or receiving data. 2) Disable the USART by writing a 0 to the Enable bit (0 may be written to the entire registers). 3) Write the new BRGVAL. 4) Write to the CFG register to set the Enable bit to 1.

Table 355. USART Baud Rate Generator register (BRG, address 0x4004 0020 (USART0), 0x4004 4020 (USART1), 0x400C 0020 (USART2)) bit description

Bit	Symbol	Description	Reset Value
15:0	BRGVAL	This value is used to divide the USART input clock to determine the baud rate, based on the input clock from the FRG. 0 = The FRG clock is used directly by the USART function. 1 = The FRG clock is divided by 2 before use by the USART function. 2 = The FRG clock is divided by 3 before use by the USART function. ... 0xFFFF = The FRG clock is divided by 65,536 before use by the USART function.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

24.6.10 USART Interrupt Status register

The read-only INTSTAT register provides a view of those interrupt flags that are currently enabled. This can simplify software handling of interrupts. See Table 349 for detailed descriptions of the interrupt flags.

Table 356. USART Interrupt Status register (INTSTAT, address 0x4004 0024 (USART0), 0x4004 4024 (USART1), 0x400C 0024 (USART2)) bit description

Bit	Symbol	Description	Reset Value
0	RXRDY	Receiver Ready flag.	0
1	-	Reserved. Read value is undefined, only zero should be written.	NA
2	TXRDY	Transmitter Ready flag.	1
3	TXIDLE	Transmitter Idle status.	0
4	-	Reserved. Read value is undefined, only zero should be written.	NA
5	DELTACTS	This bit is set when a change in the state of the CTS input is detected.	0
6	TXDISINT	Transmitter Disabled Interrupt flag.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	OVERRUNINT	Overrun Error interrupt flag.	0
10:9	-	Reserved. Read value is undefined, only zero should be written.	NA
11	DELTARXBRK	This bit is set when a change in the state of receiver break detection occurs.	0
12	START	This bit is set when a start is detected on the receiver input.	0
13	FRAMERRINT	Framing Error interrupt flag.	0
14	PARITYERRINT	Parity Error interrupt flag.	0
15	RXNOISEINT	Received Noise interrupt flag.	0
16	ABERRINT	Auto baud Error Interrupt flag.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

24.6.11 Oversample selection register

The OSR register allows selection of oversampling in asynchronous modes. The oversample value is the number of BRG clocks used to receive one data bit. The default is industry standard 16x oversampling.

Changing the oversampling can sometimes allow better matching of baud rates in cases where the peripheral clock rate is not a multiple of 16 times the expected maximum baud rate. For all modes where the OSR setting is used, the UART receiver takes three consecutive samples of input data in the approximate middle of the bit time. Smaller values of OSR can make the sampling position within a data bit less accurate and may potentially cause more noise errors or incorrect data.

Table 357. Oversample selection register (OSR, address 0x4004 0028 (USART0), 0x4004 4028 (USART1), 0x400C 0028 (USART2)) bit description

Bit	Symbol	Description	Reset value
3:0	OSRVAL	Oversample Selection Value. 0 to 3 = not supported 0x4 = 5 peripheral clocks are used to transmit and receive each data bit. 0x5 = 6 peripheral clocks are used to transmit and receive each data bit. ... 0xF= 16 peripheral clocks are used to transmit and receive each data bit.	0xF
31:4	-	Reserved, the value read from a reserved bit is not defined.	NA

24.6.12 Address register

The ADDR register holds the address for hardware address matching in address detect mode with automatic address matching enabled.

Table 358. Address register (ADDR, address 0x4004 002C (USART0), 0x4004 402C (USART1), 0x400C 002C (USART2)) bit description

Bit	Symbol	Description	Reset value
7:0	ADDRESS	8-bit address used with automatic address matching. Used when address detection is enabled (ADDRDET in CTL = 1) and automatic address matching is enabled (AUTOADDR in CFG = 1).	0
31:8	-	Reserved, the value read from a reserved bit is not defined.	NA

24.7 Functional description

24.7.1 Clocking and baud rates

In order to use the USART, clocking details must be defined such as setting up the BRG, and typically also setting up the FRG. See Figure 64.

24.7.1.1 Fractional Rate Generator (FRG)

The Fractional Rate Generator can be used to obtain more precise baud rates when the peripheral clock is not a good multiple of standard (or otherwise desirable) baud rates.

The FRG is typically set up to produce an integer multiple of the highest required baud rate, or a very close approximation. The BRG is then used to obtain the actual baud rate needed.

The FRG register controls the USART Fractional Rate Generator, which provides the base clock for the USART. The Fractional Rate Generator creates a lower rate output clock by suppressing selected input clocks. When not needed, the value of 0 can be set for the FRG, which will then not divide the input clock.

The FRG output clock is defined as the inputs clock divided by 1 + (MULT / 256), where MULT is in the range of 1 to 255. This allows producing an output clock that ranges from the input clock divided by 1+1/256 to 1+255/256 (just more than 1 to just less than 2). Any further division can be done specific to each USART block by the integer BRG divider contained in each USART.

The base clock produced by the FRG cannot be perfectly symmetrical, so the FRG distributes the output clocks as evenly as is practical. Since the USART normally uses 16x overclocking, the jitter in the fractional rate clock in these cases tends to disappear in the ultimate USART output.

For setting up the fractional divider use the following registers:

Table 61 “USART fractional baud rate generator register (FRGCTRL, address 0x4007 4128) bit description”

For details see Section 24.3.1 “Configure the USART clock and baud rate”.

24.7.1.2 Baud Rate Generator (BRG)

The Baud Rate Generator (see Section 24.6.9) is used to divide the base clock to produce a rate 16 times the desired baud rate. Typically, standard baud rates can be generated by integer divides of higher baud rates.

24.7.1.3 Baud rate calculations

Base clock rates are 16x for asynchronous mode and 1x for synchronous mode.

24.7.1.4 32 kHz mode

In order to use a 32 kHz clock to operate a USART at any reasonable speed, a number of adaptations need to be made. First, 16x overclocking has to be abandoned. Otherwise, the maximum data rate would be very low. For the same reason, multiple samples of each data bit must be reduced to one. Finally, special clocking has to be used for individual bit times because 32 kHz is not particularly close to an even multiple of any standard baud rate.

When 32 kHz mode is enabled, clocking comes from the RTC oscillator. The FRG is bypassed, and the BRG can be used to divide down the default 9600 baud to lower rates. Other adaptations required to make the UART work for rates up to 9600 baud are done internally. Rate error will be less than one half percent in this mode, provided the RTC oscillator is operating at the intended frequency of 32.768 kHz.

24.7.2 DMA

A DMA request is provided for each USART direction, and can be used in lieu of interrupts for transferring data by configuring the DMA controller appropriately. The DMA controller provides an acknowledgement signal that clears the related request when it completes handling a that request. The transmitter DMA request is asserted when the transmitter can accept more data. The receiver DMA request is asserted when received data is available to be read.

When DMA is used to perform USART data transfers, other mechanisms can be used to generate interrupts when needed. For instance, completion of the configured DMA transfer can generate an interrupt from the DMA controller. Also, interrupts for special conditions, such as a received break, can still generate useful interrupts.

24.7.3 Synchronous mode

Remark: Synchronous mode transmit and receive operate at the incoming clock rate in slave mode and the BRG selected rate (not divided by 16) in master mode.

24.7.4 Flow control

The USART supports both hardware and software flow control.

24.7.4.1 Hardware flow control

The USART supports hardware flow control using RTS and/or CTS signalling. If RTS is configured to appear on a device pin so that it can be sent to an external device, it indicates to an external device the ability of the receiver to receive more data.

If connected to a pin, and if enabled to do so, the CTS input can allow an external device to throttle the USART transmitter.

Figure 66 shows an overview of RTS and CTS within the USART.

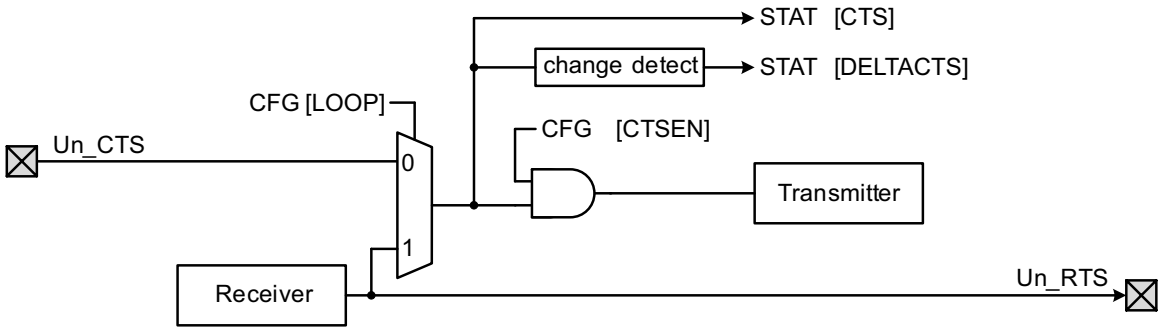


Fig 66. Hardware flow control using RTS and CTS

24.7.4.2 Software flow control

Software flow control could include XON / XOFF flow control, or other mechanisms. these are supported by the ability to check the current state of the CTS input, and/or have an interrupt when CTS changes state (via the CTS and DELTACTS bits, respectively, in the STAT register), and by the ability of software to gracefully turn off the transmitter (via the TXDIS bit in the CTL register).

24.7.5 Autobaud function

The autobaud functions attempts to measure the start bit time of the next received character. For this to work, the measured character must have a 1 in the least significant bit position, so that the start bit is bounded by a falling and rising edge. The measurement is made using the current clocking settings, including the oversampling configuration. The result is that a value is stored in the BRG register that is as close as possible to the correct setting for the sampled character and the current clocking settings. The sampled character is provided in the RXDAT and RXDATSTAT registers, allowing software to double check for the expected character.

Autobaud includes a time-out that is flagged by ABERR if no character is received at the expected time. It is recommended that autobaud only be enabled when the USART receiver is idle. Once enabled, either RXRDY or ABERR will be asserted at some point, at which time software should turn off autobaud.

Autobaud has no meaning, and should not be enabled, if the USART is in synchronous mode.

24.7.6 RS-485 support

RS-485 support requires some form of address recognition and data direction control.

This USART has provisions for hardware address recognition (see the AUTOADDR bit in the CFG register in Section 24.6.1 and the ADDR register in Section 24.6.12), as well as software address recognition (see the ADDRDET bit in the CTL register in Section 24.6.2).

Automatic data direction control with the RTS pin can be set up using the OESEL OEPOL and OETA bits in the CFG register (Section 24.6.1). Data direction control can also be implemented in software using a GPIO pin.

24.7.7 Oversampling

Typical industry standard UARTs use a 16x oversample clock to transmit and receive asynchronous data. This is the number of BRG clocks used for one data bit. The Oversample Select Register (OSR) allows this UART to use a 16x down to a 5x oversample clock. There is no oversampling in synchronous modes.

Reducing the oversampling can sometimes help in getting better baud rate matching when the baud rate is very high, or the peripheral clock is very low. For example, the closest actual rate near 115,200 baud with a 12 MHz peripheral clock and 16x oversampling is 107,143 baud, giving a rate error of 7%. Changing the oversampling to 15x gets the actual rate to 114,286 baud, a rate error of 0.8%. Reducing the oversampling to 13x gets the actual rate to 115,385 baud, a rate error of only 0.16%.

There is a cost for altering the oversampling. In asynchronous modes, the UART takes three samples of incoming data on consecutive oversample clocks, as close to the center of a bit time as can be done. When the oversample rate is reduced, the three samples spread out and occupy a larger proportion of a bit time. For example, with 5x oversampling, there is one oversample clock, then three data samples taken, then one more oversample clock before the end of the bit time. Since the oversample clock is running asynchronously from the input data, skew of the input data relative to the expected timing has little room for error. At 16x oversampling, there are several oversample clocks before actual data sampling is done, making the sampling more robust. Generally speaking, it is recommended to use the highest oversampling where the rate error is acceptable in the system.

25.1 How to read this chapter

SPI0 and SPI1 are available on all parts. SPI0 supports four slave select lines. SPI1 supports two slave select lines.

25.2 Features

- Data transmits of 1 to 16 bits supported directly. Larger frames supported by software.
- Master and slave operation.
- Data can be transmitted to a slave without the need to read incoming data. This can be useful while setting up an SPI memory.
- Control information can optionally be written along with data. This allows very versatile operation, including frames of arbitrary length.
- Up to four Slave Select input/outputs with selectable polarity and flexible usage.
- Supports DMA transfers: SPIn transmit and receive functions can operated with the system DMA controller.

Remark: Texas Instruments SSI and National Microwire modes are not supported.

25.3 Basic configuration

Configure SPI0/1 using the following registers:

- In the SYSAHBCLKCTRL1 register, set bit 9 and 10 (Table 51) to enable the clock to the register interface.
- Clear the SPI0/1 peripheral resets using the PRESETCTRL1 register (Table 36).
- Enable/disable the SPI0/1 interrupts in interrupt slots #25 and 26 in the NVIC.
- Configure the SPI0/1 pin functions through the switch matrix. See Section 25.4.
- The peripheral clock for both SPIs is the system clock (see Figure 3 “Clock generation”).

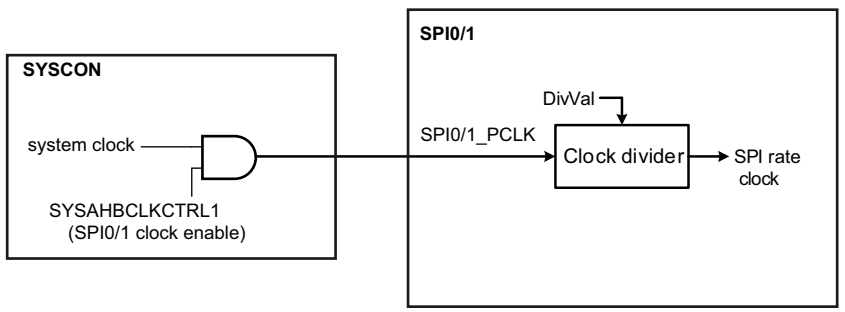


Fig 67. SPI clocking

25.3.1 Configure the SPI for wake-up

In sleep mode, any signal that triggers an SPI interrupt can wake up the part, provided that the interrupt is enabled in the INTENSET register and the NVIC. As long as the SPI clock SPI_PCLK remains active in sleep mode, the SPI can wake up the part independently of whether the SPI block is configured in master or slave mode.

In Deep-sleep or Power-down mode, the SPI clock is turned off as are all peripheral clocks. However, if the SPI is configured in slave mode and an external master on the provides the clock signal, the SPI can create an interrupt asynchronously. This interrupt, if enabled in the NVIC and in the SPI's INTENSET register, can then wake up the core.

25.3.1.1 Wake-up from Sleep mode

- Configure the SPI in either master or slave mode. See Table 361.
- Enable the SPI interrupt in the NVIC.
- Any SPI interrupt wakes up the part from sleep mode. Enable the SPI interrupt in the INTENSET register (Table 364).

25.3.1.2 Wake-up from Deep-sleep or Power-down mode

- Configure the SPI in slave mode. See Table 361. You must connect the SCK function to a pin and connect the pin to the master.
- Enable the SPI interrupt in the STARTERP0 register. See Table 76 "Start logic 0 wake-up enable register 0 (STARTERP0, address 0x4007 4218) bit description".
- In the PDAWAKE register, configure all peripherals that need to be running when the part wakes up.
- Enable the SPI interrupt in the NVIC.
- Enable the interrupt in the INTENSET register which configures the interrupt as wake-up event (Table 364). Examples are the following wake-up events:
  - A change in the state of the SSEL pins.
  - Data available to be received.
  - Receiver overrun.

25.4 Pin description

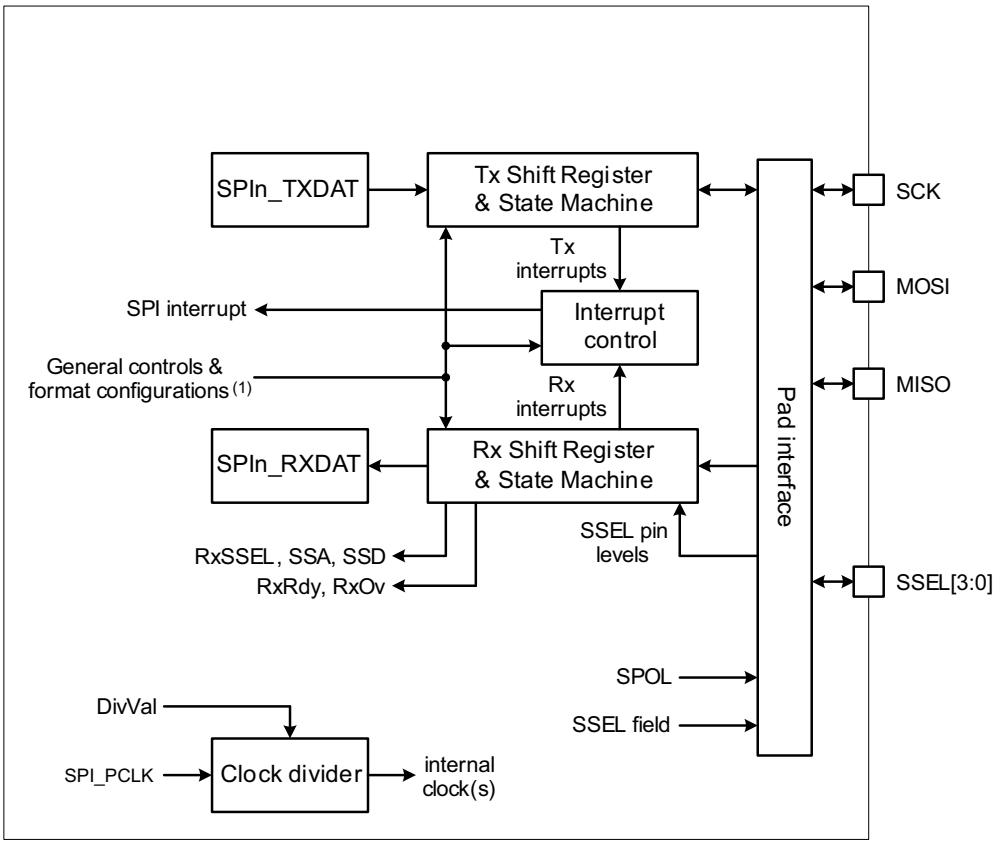
The SPI signals are movable functions and are assigned to external pins through the switch matrix.

See Section 8.3.1 "Connect an internal signal to a package pin" to assign the SPI functions to pins on the LPC15xx package.

Table 359: SPI Pin Description

Function	I/O	Type	Connect to	Use register	Reference	Description
SPI0_SCK	I/O	external to pin	any	PINASSIGN3	Table 110	Serial Clock. SCK is a clock signal used to synchronize the transfer of data. It is driven by the master and received by the slave. When the SPI interface is used, the clock is programmable to be active-high or active-low. SCK only switches during a data transfer. It is driven whenever the Master bit in CFG equals 1, regardless of the state of the Enable bit.
SPI0_MOSI	I/O	external to pin	any	PINASSIGN3	Table 110	Master Out Slave In. The MOSI signal transfers serial data from the master to the slave. When the SPI is a master, it outputs serial data on this signal. When the SPI is a slave, it clocks in serial data from this signal. MOSI is driven whenever the Master bit in SPInCfg equals 1, regardless of the state of the Enable bit.
SPI0_MISO	I/O	external to pin	any	PINASSIGN3	Table 110	Master In Slave Out. The MISO signal transfers serial data from the slave to the master. When the SPI is a master, serial data is input from this signal. When the SPI is a slave, serial data is output to this signal. MISO is driven when the SPI block is enabled, the Master bit in CFG equals 0, and when the slave is selected by one or more SSEL signals.
SPI0_SSEL0	I/O	external to pin	any	PINASSIGN4	Table 111	Slave Select 0. When the SPI interface is a master, it will drive the SSEL signals to an active state before the start of serial data and then release them to an inactive state after the serial data has been sent. By default, this signal is active low but can be selected to operate as active high. When the SPI is a slave, any SSEL in an active state indicates that this slave is being addressed. The SSEL pin is driven whenever the Master bit in the CFG register equals 1, regardless of the state of the Enable bit.
SPI0_SSEL1	I/O	external to pin	any	PINASSIGN4	Table 111	Slave Select 1.
SPI0_SSEL2	I/O	external to pin	any	PINASSIGN4	Table 111	Slave Select 2.
SPI0_SSEL3	I/O	external to pin	any	PINASSIGN4	Table 111	Slave Select 3.
SPI1_SCK	I/O	external to pin	any	PINASSIGN5	Table 112	Serial Clock.
SPI1_MOSI	I/O	external to pin	any	PINASSIGN5	Table 112	Master Out Slave In.
SPI1_MISO	I/O	external to pin	any	PINASSIGN5	Table 112	Master In Slave Out.
SPI1_SSEL0	I/O	external to pin	any	PINASSIGN5	Table 112	Slave Select 0.
SPI1_SSEL1	I/O	external to pin	any	PINASSIGN6	Table 113	Slave Select 1.

25.5 General description



(1) Includes CPOL, CPHA, LSBF, FLEN, master, enable, transfer_delay, frame_delay, pre_delay, post_delay, SOT, EOT, EOF, RXIGNORE, individual interrupt enables.

Fig 68. SPI block diagram

25.6 Register description

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

Table 360. Register overview: SPI (base address 0x4004 8000 (SPI0) and 0x4008 C000 (SPI1))

Name	Access	Offset	Description	Reset value	Reference
CFG	R/W	0x000	SPI Configuration register	0	Table 361
DLY	R/W	0x004	SPI Delay register	0	Table 362
STAT	R/W	0x008	SPI Status. Some status flags can be cleared by writing a 1 to that bit position	0x0102	Table 363

Table 360. Register overview: SPI (base address 0x4004 8000 (SPI0) and 0x4008 C000 (SPI1))
…continued

Name	Access	Offset	Description	Reset value	Reference
INTENSET	R/W	0x00C	SPI Interrupt Enable read and Set. A complete value may be read from this register. Writing a 1 to any implemented bit position causes that bit to be set.	0	Table 364
INTENCLR	W	0x010	SPI Interrupt Enable Clear. Writing a 1 to any implemented bit position causes the corresponding bit in INTENSET to be cleared.	NA	Table 365
RXDAT	R	0x014	SPI Receive Data	NA	Table 366
TXDATCTL	R/W	0x018	SPI Transmit Data with Control	0	Table 367
TXDAT	R/W	0x01C	SPI Transmit Data	0	Table 368
TXCTL	R/W	0x020	SPI Transmit Control	0	Table 369
DIV	R/W	0x024	SPI clock Divider	0	Table 370
INTSTAT	R	0x028	SPI Interrupt Status	0x02	Table 371

25.6.1 SPI Configuration register

The CFG register contains information for the general configuration of the SPI. Typically, this information is not changed during operation. Some configurations, such as CPOL, CPHA, and LSBF should not be made while the SPI is not fully idle. See the description of the master idle status (MSTIDLE in Table 363) for more information.

Remark: If the interface is re-configured from Master mode to Slave mode or the reverse (an unusual case), the SPI should be disabled and re-enabled with the new configuration.

Table 361. SPI Configuration register (CFG, addresses 0x4004 8000 (SPI0), 0x4004 C000 (SPI1)) bit description

Bit	Symbol	Value	Description	Reset value
0	ENABLE		SPI enable.	0
		0	Disabled. The SPI is disabled and the internal state machine and counters are reset.	
		1	Enabled. The SPI is enabled for operation.	
1	-		Reserved. Read value is undefined, only zero should be written.	NA
2	MASTER		Master mode select.	0
		0	Slave mode. The SPI will operate in slave mode. SCK, MOSI, and the SSEL signals are inputs, MISO is an output.	
		1	Master mode. The SPI will operate in master mode. SCK, MOSI, and the SSEL signals are outputs, MISO is an input.	
3	LSBF		LSB First mode enable.	0
		0	Standard. Data is transmitted and received in standard MSB first order.	
		1	Reverse. Data is transmitted and received in reverse order (LSB first).	

Table 361. SPI Configuration register (CFG, addresses 0x4004 8000 (SPI0), 0x4004 C000 (SPI1)) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
4	CPHA		Clock Phase select.	0
		0	Change. The SPI captures serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is changed on the following edge.	
		1	Capture. The SPI changes serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is captured on the following edge.	
5	CPOL		Clock Polarity select.	0
		0	Low. The rest state of the clock (between transfers) is low.	
		1	High. The rest state of the clock (between transfers) is high.	
6	-		Reserved. Read value is undefined, only zero should be written.	NA
7	LOOP		Loopback mode enable. Loopback mode applies only to Master mode, and connects transmit and receive data connected together to allow simple software testing.	0
		0	Disabled.	
		1	Enabled.	
8	SPOL0		SSEL0 Polarity select.	0
		0	Low. The SSEL0 pin is active low. The value in the SSEL0 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL0 is not inverted relative to the pins.	
		1	High. The SSEL0 pin is active high. The value in the SSEL0 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL0 is inverted relative to the pins.	
9	SPOL1		SSEL1 Polarity select.	0
		0	Low. The SSEL1 pin is active low. The value in the SSEL1 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL1 is not inverted relative to the pins.	
		1	High. The SSEL1 pin is active high. The value in the SSEL1 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL1 is inverted relative to the pins.	
10	SPOL2		SSEL2 Polarity select.	0
		0	Low. The SSEL2 pin is active low. The value in the SSEL2 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL2 is not inverted relative to the pins.	
		1	High. The SSEL2 pin is active high. The value in the SSEL2 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL2 is inverted relative to the pins.	
11	SPOL3		SSEL3 Polarity select.	0
		0	Low. The SSEL3 pin is active low. The value in the SSEL3 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL3 is not inverted relative to the pins.	
		1	High. The SSEL3 pin is active high. The value in the SSEL3 fields of the RXDAT, TXDATCTL, and TXCTL registers related to SSEL3 is inverted relative to the pins.	
31:12	-		Reserved. Read value is undefined, only zero should be written.	NA

25.6.2 SPI Delay register

The DLY register controls several programmable delays related to SPI signalling. These delays apply only to master mode, and are all stated in SPI clocks.

Timing details are shown in:

Section 25.7.2.1 “Pre_delay and Post_delay”

Section 25.7.2.2 “Frame_delay”

Section 25.7.2.3 “Transfer_delay”

Table 362. SPI Delay register (DLY, addresses 0x4004 8004 (SPI0), 0x4004 C004 (SPI1)) bit description

Bit	Symbol	Description	Reset value
3:0	PRE_DELAY	Controls the amount of time between SSEL assertion and the beginning of a data transfer. There is always one SPI clock time between SSEL assertion and the first clock edge. This is not considered part of the pre-delay. 0x0 = No additional time is inserted. 0x1 = 1 SPI clock time is inserted. 0x2 = 2 SPI clock times are inserted. ... 0xF = 15 SPI clock times are inserted.	0
7:4	POST_DELAY	Controls the amount of time between the end of a data transfer and SSEL deassertion. 0x0 = No additional time is inserted. 0x1 = 1 SPI clock time is inserted. 0x2 = 2 SPI clock times are inserted. ... 0xF = 15 SPI clock times are inserted.	0
11:8	FRAME_DELAY	If the EOF flag is set, controls the minimum amount of time between the current frame and the next frame (or SSEL deassertion if EOT). 0x0 = No additional time is inserted. 0x1 = 1 SPI clock time is inserted. 0x2 = 2 SPI clock times are inserted. ... 0xF = 15 SPI clock times are inserted.	0
15:12	TRANSFER_DELAY	Controls the minimum amount of time that the SSEL is deasserted between transfers. 0x0 = The minimum time that SSEL is deasserted is 1 SPI clock time. (Zero added time.) 0x1 = The minimum time that SSEL is deasserted is 2 SPI clock times. 0x2 = The minimum time that SSEL is deasserted is 3 SPI clock times. ... 0xF = The minimum time that SSEL is deasserted is 16 SPI clock times.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

25.6.3 SPI Status register

The STAT register provides SPI status flags for software to read, and a control bit for forcing an end of transfer. Flags other than read-only flags may be cleared by writing ones to corresponding bits of STAT.

STAT contains 2 error flags (in slave mode only): RXOV and TXUR. These are receiver overrun and transmit underrun, respectively. If either of these errors occur during operation, the SPI should be disabled, then re-enabled in order to make sure all internal states are cleared before attempting to resume operation.

In this register, the following notation is used: RO = Read-only, W1 = write 1 to clear.

Table 363. SPI Status register (STAT, addresses 0x4004 8008 (SPI0), 0x4004 C008 (SPI1)) bit description

Bit	Symbol	Description	Reset value	Access [1]
0	RXRDY	Receiver Ready flag. When 1, indicates that data is available to be read from the receiver buffer. Cleared after a read of the RXDAT register.	0	RO
1	TXRDY	Transmitter Ready flag. When 1, this bit indicates that data may be written to the transmit buffer. Previous data may still be in the process of being transmitted. Cleared when data is written to TXDAT or TXDATCTL until the data is moved to the transmit shift register.	1	RO
2	RXOV	Receiver Overrun interrupt flag. This flag applies only to slave mode (Master = 0). This flag is set when the beginning of a received character is detected while the receiver buffer is still in use. If this occurs, the receiver buffer contents are preserved, and the incoming data is lost. Data received by the SPI should be considered undefined if RxOv is set.	0	W1
3	TXUR	Transmitter Underrun interrupt flag. This flag applies only to slave mode (Master = 0). In this case, the transmitter must begin sending new data on the next input clock if the transmitter is idle. If that data is not available in the transmitter holding register at that point, there is no data to transmit and the TXUR flag is set. Data transmitted by the SPI should be considered undefined if TXUR is set.	0	W1
4	SSA	Slave Select Assert. This flag is set whenever any slave select transitions from deasserted to asserted, in both master and slave modes. This allows determining when the SPI transmit/receive functions become busy, and allows waking up the device from reduced power modes when a slave mode access begins. This flag is cleared by software.	0	W1
5	SSD	Slave Select Deassert. This flag is set whenever any asserted slave selects transition to deasserted, in both master and slave modes. This allows determining when the SPI transmit/receive functions become idle. This flag is cleared by software.	0	W1
6	STALLED	Stalled status flag. This indicates whether the SPI is currently in a stall condition.	0	RO

Table 363. SPI Status register (STAT, addresses 0x4004 8008 (SPI0), 0x4004 C008 (SPI1)) bit description

Bit	Symbol	Description	Reset value	Access [1]
7	ENDTRANSFER	End Transfer control bit. Software can set this bit to force an end to the current transfer when the transmitter finishes any activity already in progress, as if the EOT flag had been set prior to the last transmission. This capability is included to support cases where it is not known when transmit data is written that it will be the end of a transfer. The bit is cleared when the transmitter becomes idle as the transfer comes to an end. Forcing an end of transfer in this manner causes any specified FRAME_DELAY and TRANSFER_DELAY to be inserted.	0	RO/W1
8	MSTIDLE	Master idle status flag. This bit is 1 whenever the SPI master function is fully idle. This means that the transmit holding register is empty and the transmitter is not in the process of sending data.	1	RO
31:9	-	Reserved. Read value is undefined, only zero should be written.	NA	NA

[1] RO = Read-only, W1 = write 1 to clear.

25.6.4 SPI Interrupt Enable read and Set register

The INTENSET register is used to enable various SPI interrupt sources. Enable bits in INTENSET are mapped in locations that correspond to the flags in the STAT register. The complete set of interrupt enables may be read from this register. Writing ones to implemented bits in this register causes those bits to be set. The INTENCLR register is used to clear bits in this register. See Table 363 for details of the interrupts.

Table 364. SPI Interrupt Enable read and Set register (INTENSET, addresses 0x4004 800C (SPI0), 0x4004 C00C (SPI1)) bit description

Bit	Symbol	Value	Description	Reset value
0	RXRDYEN		Determines whether an interrupt occurs when receiver data is available.	0
		0	No interrupt will be generated when receiver data is available.	
		1	An interrupt will be generated when receiver data is available in the RXDAT register.	
1	TXRDYEN		Determines whether an interrupt occurs when the transmitter holding register is available.	0
		0	No interrupt will be generated when the transmitter holding register is available.	
		1	An interrupt will be generated when data may be written to TXDAT.	
2	RXOVEN		Determines whether an interrupt occurs when a receiver overrun occurs. This happens in slave mode when there is a need for the receiver to move newly received data to the RXDAT register when it is already in use. The interface prevents receiver overrun in Master mode by not allowing a new transmission to begin when a receiver overrun would otherwise occur.	0
		0	No interrupt will be generated when a receiver overrun occurs.	
		1	An interrupt will be generated if a receiver overrun occurs.	
3	TXUREN		Determines whether an interrupt occurs when a transmitter underrun occurs. This happens in slave mode when there is a need to transmit data when none is available.	0
		0	No interrupt will be generated when the transmitter underruns.	
		1	An interrupt will be generated if the transmitter underruns.	

Table 364. SPI Interrupt Enable read and Set register (INTENSET, addresses 0x4004 800C (SPI0), 0x4004 C00C (SPI1)) bit description

Bit	Symbol	Value	Description	Reset value
4	SSAEN		Determines whether an interrupt occurs when the Slave Select is asserted.	0
		0	No interrupt will be generated when any Slave Select transitions from deasserted to asserted.	
		1	An interrupt will be generated when any Slave Select transitions from deasserted to asserted.	
5	SSDEN		Determines whether an interrupt occurs when the Slave Select is deasserted.	0
		0	No interrupt will be generated when all asserted Slave Selects transition to deasserted.	
		1	An interrupt will be generated when all asserted Slave Selects transition to deasserted.	
31:6	-		Reserved. Read value is undefined, only zero should be written.	NA

25.6.5 SPI Interrupt Enable Clear register

The INTENCLR register is used to clear interrupt enable bits in the INTENSET register.

Table 365. SPI Interrupt Enable clear register (INTENCLR, addresses 0x4004 8010 (SPI0), 0x4004 C010 (SPI1)) bit description

Bit	Symbol	Description	Reset value
0	RXRDYEN	Writing 1 clears the corresponding bits in the INTENSET register.	0
1	TXRDYEN	Writing 1 clears the corresponding bits in the INTENSET register.	0
2	RXOVEN	Writing 1 clears the corresponding bits in the INTENSET register.	0
3	TXUREN	Writing 1 clears the corresponding bits in the INTENSET register.	0
4	SSAEN	Writing 1 clears the corresponding bits in the INTENSET register.	0
5	SSDEN	Writing 1 clears the corresponding bits in the INTENSET register.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA

25.6.6 SPI Receiver Data register

The read-only RXDAT register provides the means to read the most recently received data. The value of SSEL can be read along with the data.

For details on the slave select process, see Section 25.7.4.

Table 366. SPI Receiver Data register (RXDAT, addresses 0x4004 8014 (SPI0), 0x4004 C014 (SPI1)) bit description

Bit	Symbol	Description	Reset value
15:0	RXDAT	Receiver Data. This contains the next piece of received data. The number of bits that are used depends on the LEN setting in TXCTL / TXDATCTL.	undefined
16	RXSSEL0_N	Slave Select for receive. This field allows the state of the SSEL0 pin to be saved along with received data. The value will reflect the SSEL0 pin for both master and slave operation. A zero indicates that a slave select is active. The actual polarity of each slave select pin is configured by the related SPOL bit in CFG.	undefined
17	RXSSEL1_N	Slave Select for receive. This field allows the state of the SSEL1 pin to be saved along with received data. The value will reflect the SSEL1 pin for both master and slave operation. A zero indicates that a slave select is active. The actual polarity of each slave select pin is configured by the related SPOL bit in CFG.	undefined
18	RXSSEL2_N	Slave Select for receive. This field allows the state of the SSEL2 pin to be saved along with received data. The value will reflect the SSEL2 pin for both master and slave operation. A zero indicates that a slave select is active. The actual polarity of each slave select pin is configured by the related SPOL bit in CFG.	undefined
19	RXSSEL3_N	Slave Select for receive. This field allows the state of the SSEL3 pin to be saved along with received data. The value will reflect the SSEL3 pin for both master and slave operation. A zero indicates that a slave select is active. The actual polarity of each slave select pin is configured by the related SPOL bit in CFG.	undefined
20	SOT	Start of Transfer flag. This flag will be 1 if this is the first data after the SSELs went from deasserted to asserted (i.e., any previous transfer has ended). This information can be used to identify the first piece of data in cases where the transfer length is greater than 16 bit.	
31:21	-	Reserved, the value read from a reserved bit is not defined.	NA

25.6.7 SPI Transmitter Data and Control register

The TXDATCTL register provides a location where both transmit data and control information can be written simultaneously. This allows detailed control of the SPI without a separate write of control information for each piece of data, which can be especially useful when the SPI is used with DMA.

Remark: The SPI has no receiver control registers. Hence software needs to set the data length in the transmitter control or transmitter data and control register first in order to handle reception with correct data length. The programmed data length becomes active only when data is actually transmitted. Therefore, this must be done before any data can be received.

When control information remains static during transmit, the TXDAT register should be used (see Section 25.6.8) instead of the TXDATCTL register. Control information can then be written separately via the TXCTL register (see Section 25.6.9). The upper part of TXDATCTL (bits 27 to 16) are the same bits contained in the TXCTL register. The two registers simply provide two ways to access them.

For details on the slave select process, see Section 25.7.4.

For details on using multiple consecutive data transmits for transfer lengths larger than 16 bit, see Section 25.7.6 “Data lengths greater than 16 bits”.

Table 367. SPI Transmitter Data and Control register (TXDATCTL, addresses 0x4004 8018 (SPI0), 0x4004 C018 (SPI1)) bit description

Bit	Symbol	Value	Description	Reset value
15:0	TXDAT		Transmit Data. This field provides from 1 to 16 bits of data to be transmitted.	0
16	TXSSEL0_N		Transmit Slave Select. This field asserts SSEL0 in master mode. The output on the pin is active LOW by default. **Remark:** The active state of the SSEL0 pin is configured by bits in the CFG register.	0
		0	SSEL0 asserted.	
		1	SSEL0 not asserted.	
17	TXSSEL1_N		Transmit Slave Select. This field asserts SSEL1 in master mode. The output on the pin is active LOW by default. **Remark:** The active state of the SSEL1 pin is configured by bits in the CFG register.	0
		0	SSEL1 asserted.	
		1	SSEL1 not asserted.	
18	TXSSEL2_N		Transmit Slave Select. This field asserts SSEL2 in master mode. The output on the pin is active LOW by default. **Remark:** The active state of the SSEL2 pin is configured by bits in the CFG register.	0
		0	SSEL2 asserted.	
		1	SSEL2 not asserted.	
19	TXSSEL3_N		Transmit Slave Select. This field asserts SSEL3 in master mode. The output on the pin is active LOW by default. **Remark:** The active state of the SSEL3 pin is configured by bits in the CFG register.	0
		0	SSEL3 asserted.	
		1	SSEL3 not asserted.	

Table 367. SPI Transmitter Data and Control register (TXDATCTL, addresses 0x4004 8018 (SPI0), 0x4004 C018 (SPI1)) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
20	EOT		End of Transfer. The asserted SSEL will be deasserted at the end of a transfer, and remain so for at least the time specified by the Transfer_delay value in the DLY register.	0
		0	SSEL not deasserted. This piece of data is not treated as the end of a transfer. SSEL will not be deasserted at the end of this data.	
		1	SSEL deasserted. This piece of data is treated as the end of a transfer. SSEL will be deasserted at the end of this piece of data.	
21	EOF		End of Frame. Between frames, a delay may be inserted, as defined by the FRAME_DELAY value in the DLY register. The end of a frame may not be particularly meaningful if the FRAME_DELAY value = 0. This control can be used as part of the support for frame lengths greater than 16 bits.	0
		0	Data not EOF. This piece of data transmitted is not treated as the end of a frame.	
		1	Data EOF. This piece of data is treated as the end of a frame, causing the FRAME_DELAY time to be inserted before subsequent data is transmitted.	
22	RXIGNORE		Receive Ignore. This allows data to be transmitted using the SPI without the need to read unneeded data from the receiver.Setting this bit simplifies the transmit process and can be used with the DMA.	0
		0	Read received data. Received data must be read in order to allow transmission to progress. In slave mode, an overrun error will occur if received data is not read before new data is received.	
		1	Ignore received data. Received data is ignored, allowing transmission without reading unneeded received data. No receiver flags are generated.	
23	-		Reserved. Read value is undefined, only zero should be written.	NA
27:24	LEN		Data Length. Specifies the data length from 1 to 16 bits. Note that transfer lengths greater than 16 bits are supported by implementing multiple sequential transmits. 0x0 = Data transfer is 1 bit in length. 0x1 = Data transfer is 2 bits in length. 0x2 = Data transfer is 3 bits in length. ... 0xF = Data transfer is 16 bits in length.	0x0
31:28	-		Reserved. Read value is undefined, only zero should be written.	NA

25.6.8 SPI Transmitter Data Register

The TXDAT register is written in order to send data via the SPI transmitter when control information is not changing during the transfer (see Section 25.6.7). That data will be sent to the transmit shift register when it is available, and another character may then be written to TXDAT.

Table 368. SPI Transmitter Data Register (TXDAT, addresses 0x4004 801C (SPI0), 0x4004 C01C (SPI1)) bit description

Bit	Symbol	Description	Reset value
15:0	DATA	Transmit Data. This field provides from 4 to 16 bits of data to be transmitted.	0
31:16	-	Reserved. Only zero should be written.	NA

25.6.9 SPI Transmitter Control register

The TXCTL register provides a way to separately access control information for the SPI. These bits are another view of the same-named bits in the TXDATCTL register (see Section 25.6.7). Changing bits in TXCTL has no effect unless data is later written to the TXDAT register. Data written to TXDATCTL overwrites the TXCTL register.

When control information needs to be changed during transmission, the TXDATCTL register should be used (see Section 25.6.7) instead of TXDAT. Control information can then be written along with data.

Table 369. SPI Transmitter Control register (TXCTL, addresses 0x4004 8020 (SPI0), 0x4004 C020 (SPI1)) bit description

Bit	Symbol	Description	Reset value
15:0	-	Reserved. Read value is undefined, only zero should be written.	NA
16	TXSSEL0_N	Transmit Slave Select 0.	0x0
17	TXSSEL1_N	Transmit Slave Select 1.	0x0
18	TXSSEL2_N	Transmit Slave Select 2.	0x0
19	TXSSEL3_n	Transmit Slave Select 3.	0x0
20	EOT	End of Transfer.	0
21	EOF	End of Frame.	0
22	RXIGNORE	Receive Ignore.	0
23	-	Reserved. Read value is undefined, only zero should be written.	NA
27:24	LEN	Data transfer Length.	0x0
31:28	-	Reserved. Read value is undefined, only zero should be written.	NA

25.6.10 SPI Divider register

The DIV register determines the clock used by the SPI in master mode.

For details on clocking, see Section 25.7.3 “Clocking and data rates”.

Table 370. SPI Divider register (DIV, addresses 0x4004 8024 (SPI0), 0x4004 C024 (SPI1)) bit description

Bit	Symbol	Description	Reset Value
15:0	DIVVAL	Rate divider value. Specifies how the PCLK for the SPI is divided to produce the SPI clock rate in master mode. DIVVAL is -1 encoded such that the value 0 results in PCLK/1, the value 1 results in PCLK/2, up to the maximum possible divide value of 0xFFFF, which results in PCLK/65536.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

25.6.11 SPI Interrupt Status register

The read-only INTSTAT register provides a view of those interrupt flags that are currently enabled. This can simplify software handling of interrupts. See Table 363 for detailed descriptions of the interrupt flags.

Table 371. SPI Interrupt Status register (INTSTAT, addresses 0x4004 8028 (SPI0), 0x4004 C028 (SPI1)) bit description

Bit	Symbol	Description	Reset value
0	RXRDY	Receiver Ready flag.	0
1	TXRDY	Transmitter Ready flag.	1
2	RXOV	Receiver Overrun interrupt flag.	0
3	TXUR	Transmitter Underrun interrupt flag.	0
4	SSA	Slave Select Assert.	0
5	SSD	Slave Select Deassert.	0
31:6	-	Reserved. Read value is undefined, only zero should be written.	NA

25.7 Functional description

25.7.1 Operating modes: clock and phase selection

SPI interfaces typically allow configuration of clock phase and polarity. These are sometimes referred to as numbered SPI modes, as described in Table 372 and shown in Figure 69. CPOL and CPHA are configured by bits in the CFG register (Section 25.6.1).

Table 372: SPI mode summary

CPOL	CPHA	SPI Mode	Description	SCK rest state	SCK data change edge	SCK data sample edge
0	0	0	The SPI captures serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is changed on the following edge.	low	falling	rising
0	1	1	The SPI changes serial data on the first clock transition of the transfer (when the clock changes away from the rest state). Data is captured on the following edge.	low	rising	falling
1	0	2	Same as mode 0 with SCK inverted.	high	rising	falling
1	1	3	Same as mode 1 with SCK inverted.	high	falling	rising

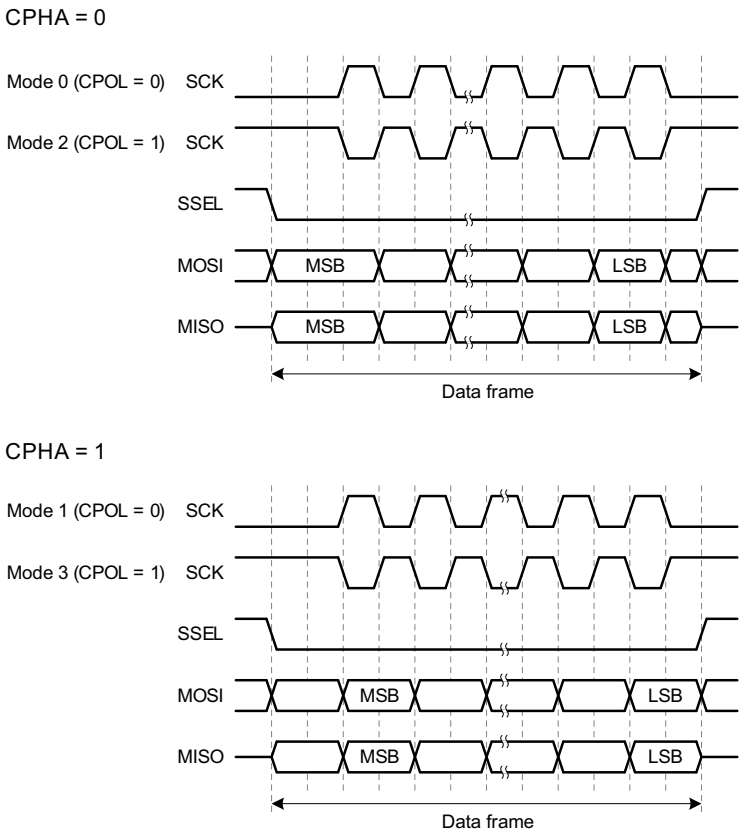


Fig 69. Basic SPI operating modes

25.7.2 Frame delays

Several delays can be specified for SPI frames. These include:

- Pre_delay: delay after SSEL is asserted before data clocking begins
- Post_delay: delay at the end of a data frame before SSEL is deasserted
- Frame_delay: delay between data frames when SSEL is not deasserted
- Transfer_delay: minimum duration of SSEL in the deasserted state between transfers

25.7.2.1 Pre_delay and Post_delay

Pre_delay and Post_delay are illustrated by the examples in Figure 70. The Pre_delay value controls the amount of time between SSEL being asserted and the beginning of the subsequent data frame. The Post_delay value controls the amount of time between the end of a data frame and the deassertion of SSEL.

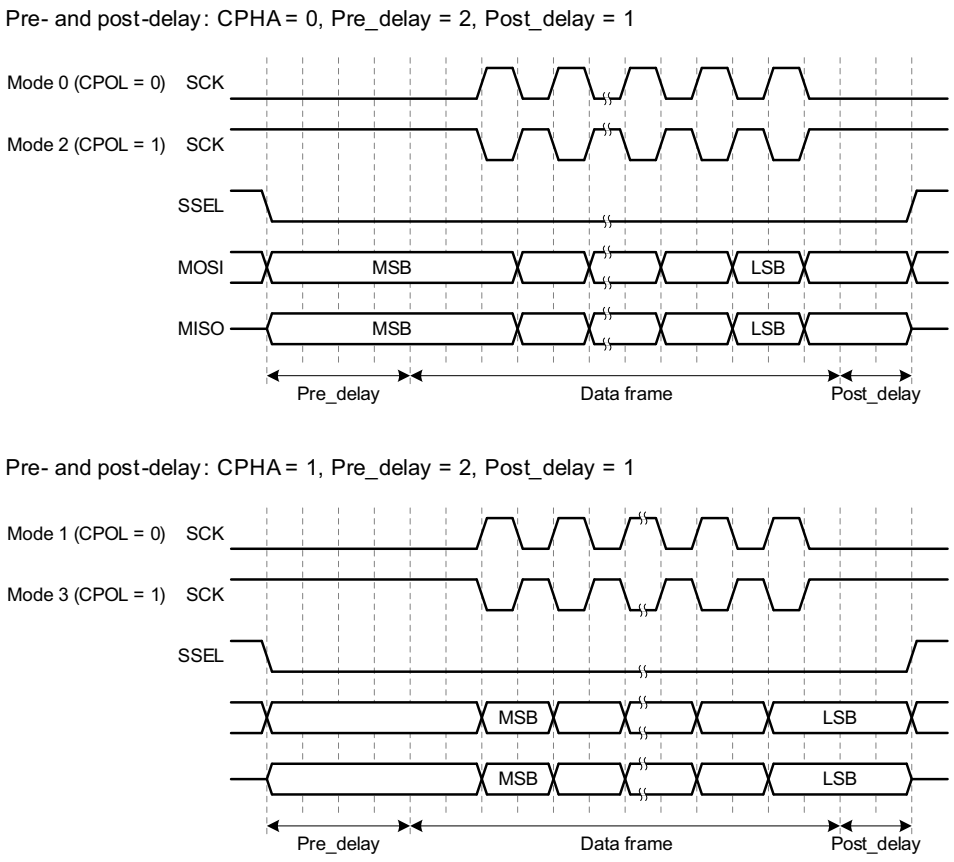


Fig 70. Pre_delay and Post_delay

25.7.2.2 Frame_delay

The Frame_delay value controls the amount of time at the end of each frame. This delay is inserted when the EOF bit = 1. Frame_delay is illustrated by the examples in Figure 71. Note that frame boundaries occur only where specified. This is because frame lengths can be any size, involving multiple data writes. See Section 25.7.6 for more information.

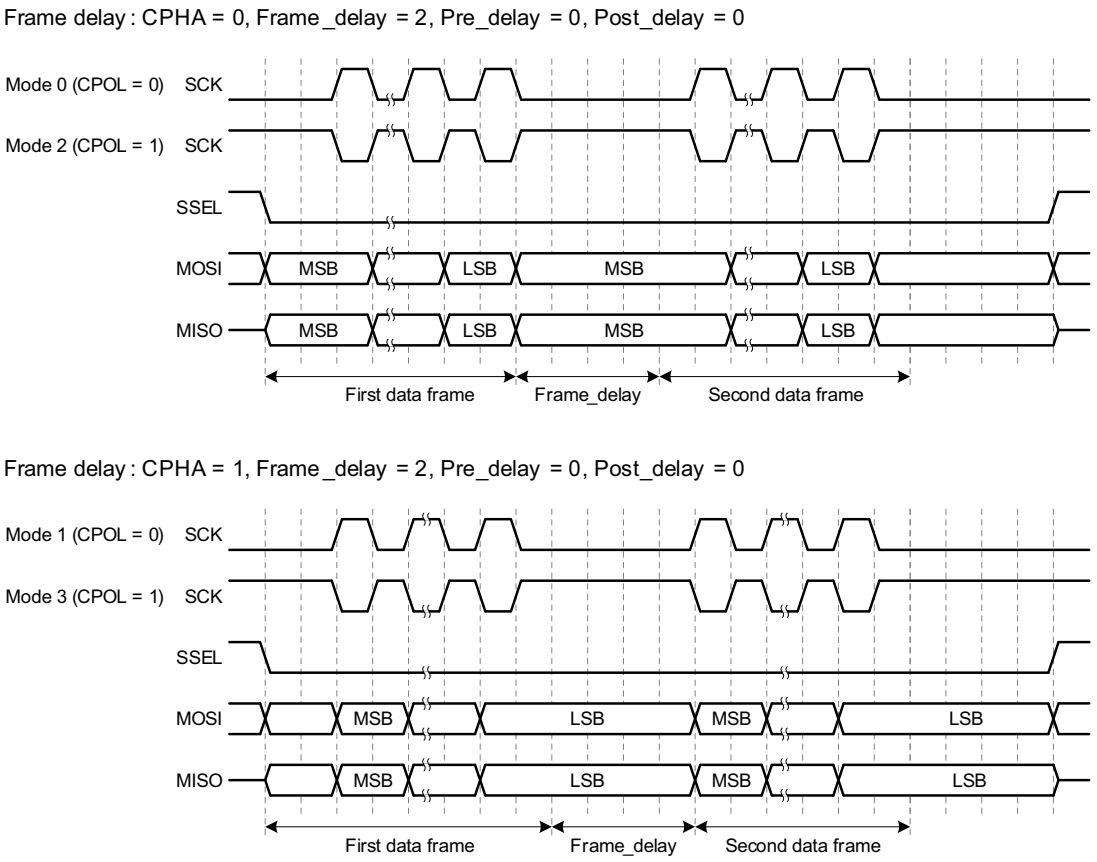


Fig 71. Frame_delay

25.7.2.3 Transfer_delay

The Transfer_delay value controls the minimum amount of time that SSEL is deasserted between transfers, because the EOT bit = 1. When Transfer_delay = 0, SSEL may be deasserted for a minimum of one SPI clock time. Transfer_delay is illustrated by the examples in Figure 72.

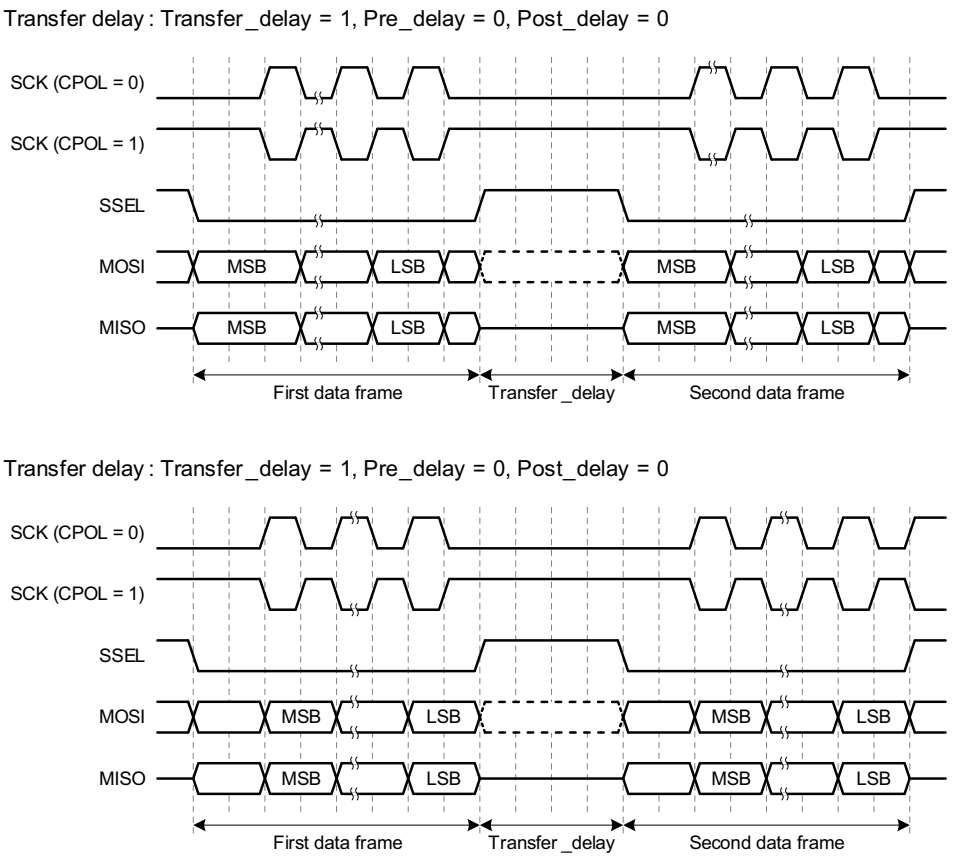


Fig 72. Transfer_delay

25.7.3 Clocking and data rates

In order to use the SPI, clocking details must be defined. This includes configuring the system clock and selection of the clock divider value in DIV. See Figure 67.

25.7.3.1 Data rate calculations

The SPI interface is designed to operate asynchronously from any on-chip clocks, and without the need for overclocking.

In slave mode, this means that the SCK from the external master is used directly to run the transmit and receive shift registers and other logic.

In master mode, the SPI rate clock produced by the SPI clock divider is used directly as the outgoing SCK.

The SPI clock divider is an integer divider. The SPI in master mode can be set to run at the same speed as the selected PCLK, or at lower integer divide rates. The SPI rate will be = PCLK_SPIn / DIVVAL.

In slave mode, the clock is taken from the SCK input and the SPI clock divider is not used.

25.7.4 Slave select

The SPI block provides for four Slave Select inputs in slave mode or outputs in master mode. Each SSEL can be set for normal polarity (active low), or can be inverted (active high). Representation of the 4 SSELs in a register is always active low. If an SSEL is inverted, this is done as the signal leaves/enters the SPI block.

In slave mode, **any** asserted SSEL that is connected to a pin will activate the SPI. In master mode, all SSELs that are connected to a pin will be output as defined in the SPI registers. In the latter case, the SSELs could potentially be decoded externally in order to address more than four slave devices. Note that at least one SSEL is asserted when data is transferred in master mode.

In master mode, Slave Selects come from the SSELN field, which appears in both the CTL and DATCTL registers. In slave mode, the state of all four SSELs is saved along with received data in the RXSSEL_N field of the RXDAT register.

25.7.5 DMA operation

A DMA request is provided for each SPI direction, and can be used in lieu of interrupts for transferring data by configuring the DMA controller appropriately, and enabling the Rx and/or Tx DMA via the CFG register. The DMA controller provides an acknowledgement signal that clears the related request when it completes handling that request.

The transmitter DMA request is asserted when Tx DMA is enabled and the transmitter can accept more data.

The receiver DMA request is asserted when Rx DMA is enabled and received data is available to be read.

25.7.6 Data lengths greater than 16 bits

The SPI interface handles data frame sizes from 1 to 16 bits directly. Larger sizes can be handled by splitting data up into groups of 16 bits or less. For example, 24 bits can be supported as 2 groups of 16 bits and 8 bits or 2 groups of 12 bits, among others. Frames of any size, including greater than 32 bits, can supported in the same way.

Details of how to handle larger data widths depend somewhat on other SPI configuration options. For instance, if it is intended for Slave Selects to be deasserted between frames, then this must be suppressed when a larger frame is split into more than one part. Sending 2 groups of 12 bits with SSEL deasserted between 24-bit increments, for instance, would require changing the value of the EOF bit on alternate 12-bit frames.

25.7.7 Data stalls

A stall for Master transmit data can happen in modes 0 and 2 when SCK cannot be returned to the rest state until the MSB of the next data frame can be driven on MOSI. In this case, the stall happens just before the final clock edge of data if the next piece of data is not yet available.

A stall for Master receive can happen when a receiver overrun would otherwise occur if the transmitter was not stalled. In modes 0 and 2, this occurs if the previously received data is not read before the end of the next piece of is received. This stall happens one clock edge earlier than the transmitter stall.

In modes 1 and 3, the same kind of receiver stall can occur, but just before the final clock edge of the received data. Also, a transmitter stall will not happen in modes 1 and 3 because the transmitted data is complete at the point where a stall would otherwise occur, so it is not needed.

Stalls are reflected in the STAT register by the Stalled status flag, which indicates the current SPI status.

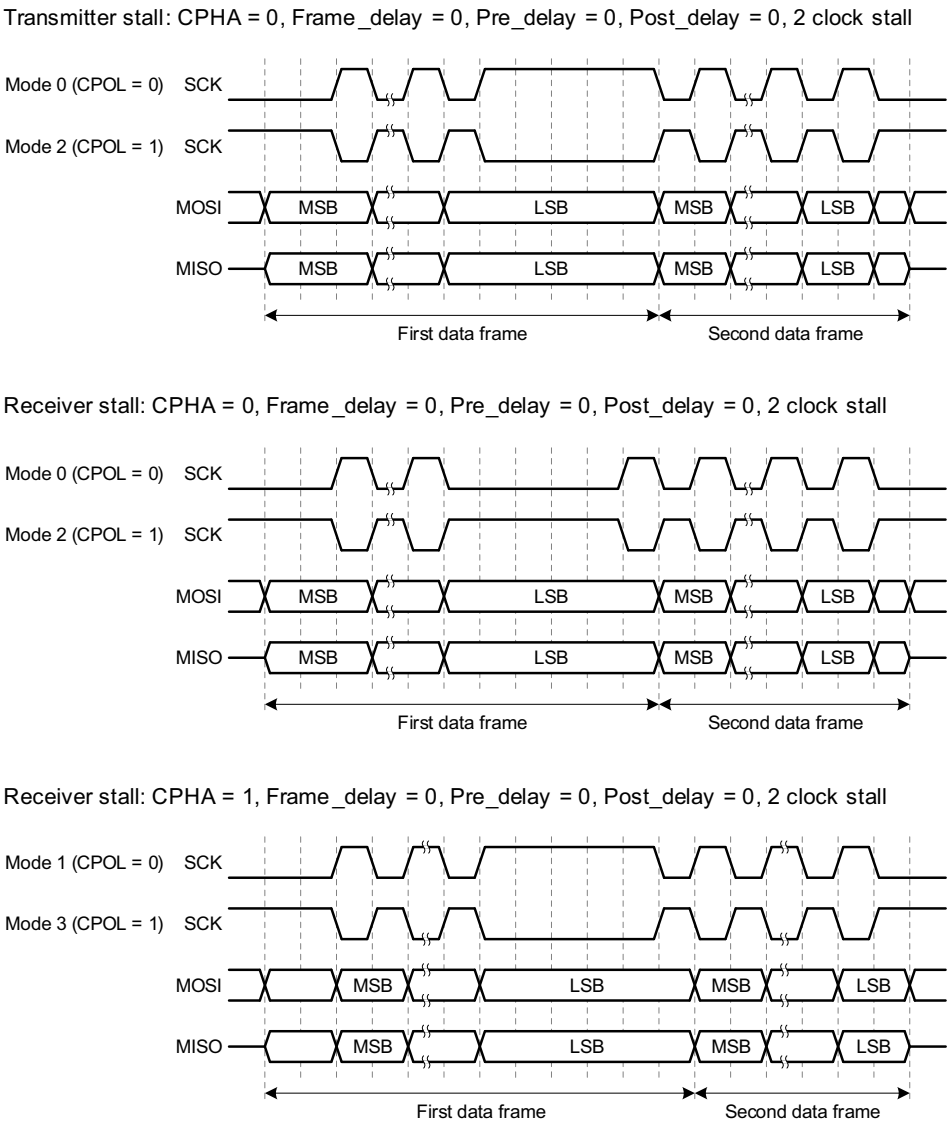


Fig 73. Examples of data stalls

26.1 How to read this chapter

The I2C-bus interface is available on all parts.

Read this chapter if you want to understand the I2C operation and the software interface and want to learn how to use the I2C for wake-up from reduced power modes.

The LPC15xx provides an on-chip ROM-based I2C API to configure and operate the I2C. See Chapter 38.

26.2 Features

- Independent Master, Slave, and Monitor functions.
- Supports both Multi-master and Multi-master with Slave functions.
- Multiple I^2C slave addresses supported in hardware.
- One slave address can be selectively qualified with a bit mask or an address range in order to respond to multiple I^2C bus addresses.
- 10-bit addressing supported with software assist.
- Supports SMBus.
- Supports the I^2C-bus specification up to Fast-mode Plus (up to 1 MHz).

26.3 Pin description

The I2C pins are fixed-pin functions and enabled through the switch matrix.

If the I2C-bus interface is used in Fast-mode Plus mode, configure the I2C-pins for this mode in the IOCON block: Table 101 “Digital open-drain pin control registers (PIO0_[22:23], address 0x400F 805C (PIO0_22) to 0x400F 8060 (PIO0_23)) bit description”.

Table 373. I2C-bus pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
I2C0_SCL	I/O	external to pin	P0_22	PINENABLE1	Table 124	I2C0 serial clock.
I2C0_SDA	I/O	external to pin	P0_23	PINENABLE1	Table 124	I2C0 serial data.

26.4 Basic configuration

Configure I2C using the following registers:

- In the SYSAHBCLKCTRL1 register, set bit 13 (Table 51) to enable the clock to the register interface.
- Clear the I2C peripheral reset using the PRESETCTRL1 register (Table 36).

- Enable/disable the I2C interrupt in interrupt slots #24 in the NVIC.
- Configure the I2C pin functions through the switch matrix. See Table 124.
- The peripheral clock for the I2C is the system clock (see Figure 74).

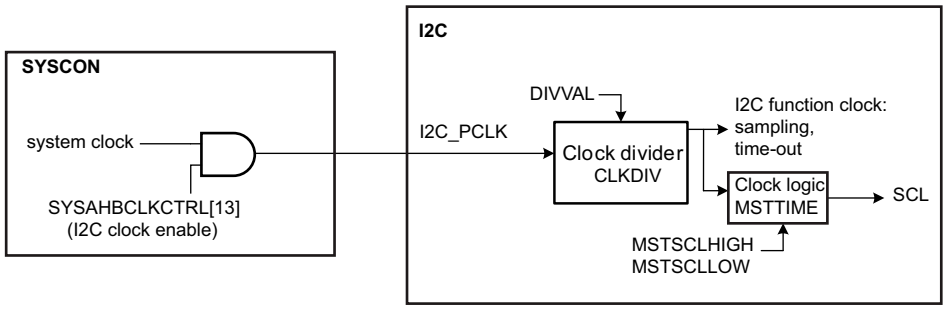


Fig 74. I2C clocking

26.4.1 I2C transmit/receive in master mode

Remark: See Section 44.2 “Code examples I2C” for code examples.

In this example, the I2C is configured as the master. The master sends 8 bits to the slave and then receives 8 bits from the slave. The system clock is set to 30 MHz and the bit rate is approximately 400 KHz. You must enable the I2C0_SCL and I2C0_SDA functions on pins PIO0_22 and PIO0_23 through the switch matrix. See Table 124.

For a 400 KHz bit rate, the pins can be configured in standard mode in the IOCON block. See Table 101 “Digital open-drain pin control registers (PIO0_[22:23], address 0x400F 805C (PIO0_22) to 0x400F 8060 (PIO0_23)) bit description”.

The transmission of the address and data bits is controlled by the state of the MSTPENDING status bit. Whenever the status is Master pending, the master can read or write to the MSTDAT register and go to the next step of the transmission protocol by writing to the MSTCTL register.

Configure the I2C bit rate:

- Divide the system clock (I2C_PCLK) by a factor of 2. See Table 382 “I^2C Clock Divider register (CLKDIV, address 0x4005 0014) bit description”.
- Set the SCL high and low times to 2 clock cycles each. This is the default. See Table 385 “Master Time register (MSTTIME, address 0x4005 0024) bit description”. The result is an SCL clock of 375 kHz.

26.4.1.1 Master write to slave

Configure the I2C as master: Set the MSTEN bit to 1 in the CFG register. See Table 375.

Write data to the slave:

1. Write the slave address with the $\overline{RW}$ bit set to 0 to the Master data register MSTDAT. See Table 386.
2. Start the transmission by setting the MSTSTART bit to 1 in the Master control register. See Table 384. The following happens:
   - The pending status is cleared and the I2C-bus is busy.
   - The I2C master sends the start bit and address with the $\overline{RW}$ bit to the slave.
3. Wait for the pending status to be set (MSTPENDING = 1) by polling the STAT register.
4. Write 8 bits of data to the MSTDAT register.
5. Continue with the transmission of data by setting the MSTCONT bit to 1 in the Master control register. See Table 384. The following happens:
   - The pending status is cleared and the I2C-bus is busy.
   - The I2C master sends the data bits to the slave address.
6. Wait for the pending status to be set (MSTPENDING = 1) by polling the STAT register.
7. Stop the transmission by setting the MSTSTOP bit to 1 in the Master control register. See Table 384.

26.4.1.2 Master read from slave

Configure the I2C as master: Set the MSTEN bit to 1 in the CFG register. See Table 375.

Read data from the slave:

1. Write the slave address with the $\overline{RW}$ bit set to 1 to the Master data register MSTDAT. See Table 386.
2. Start the transmission by setting the MSTSTART bit to 1 in the Master control register. See Table 384. The following happens:
   - The pending status is cleared and the I2C-bus is busy.
   - The I2C master sends the start bit and address with the $\overline{RW}$ bit to the slave.
   - The slave sends 8 bit of data.
3. Wait for the pending status to be set (MSTPENDING = 1) by polling the STAT register.
4. Read 8 bits of data from the MSTDAT register.
5. Stop the transmission by setting the MSTSTOP bit to 1 in the Master control register. See Table 384.

26.4.2 I2C receive/transmit in slave mode

In this example, the I2C is configured as the slave. The slave receives 8 bits from the master and sends 8 bits to the slave. The system clock is set to 30 MHz and the bit rate is approximately 400 KHz. You must enable the I2C0_SCL and I2C0_SDA functions on pins PIO0_22 and PIO0_23 through the switch matrix. See Table 124.

For a 400 KHz bit rate, the pins can be configured in standard mode in the IOCON block. See Table 101 “Digital open-drain pin control registers (PIO0_[22:23], address 0x400F 805C (PIO0_22) to 0x400F 8060 (PIO0_23)) bit description”.

The transmission of the address and data bits is controlled by the state of the SLVPENDING status bit. Whenever the status is Slave pending, the slave can acknowledge (“ack”) or send or receive an address and data. The received data or the data to be sent to the master are available in the SLVDAT register. After sending and receiving data, continue to the next step of the transmission protocol by writing to the SLVCTL register.

26.4.2.1 Slave read from master

Configure the I2C as slave with address x:

- Set the SLVEN bit to 1 in the CFG register. See Table 375.
- Write the slave address x to the address 0 match register. See Table 389.

Read data from the master:

1. Wait for the pending status to be set (SLVPENDING = 1) by polling the STAT register.
2. Acknowledge (“ack”) the address by setting SLVCONTINUE = 1 in the slave control register. See Table 387.
3. Wait for the pending status to be set (SLVPENDING = 1) by polling the STAT register.
4. Read 8 bits of data from the SLVDAT register. See Table 388.
5. Acknowledge (“ack”) the data by setting SLVCONTINUE = 1 in the slave control register. See Table 387.

26.4.2.2 Slave write to master

- Set the SLVEN bit to 1 in the CFG register. See Table 375.
- Write the slave address x to the address 0 match register. See Table 389.

Write data to the master:

1. Wait for the pending status to be set (SLVPENDING = 1) by polling the STAT register.
2. ACK the address by setting SLVCONTINUE = 1 in the slave control register. See Table 387.
3. Wait for the pending status to be set (SLVPENDING = 1) by polling the STAT register.
4. Write 8 bits of data to SLVDAT register. See Table 388.
5. Continue the transaction by setting SLVCONTINUE = 1 in the slave control register. See Table 387.

26.4.3 Configure the I2C for wake-up

In sleep mode, any activity on the I2C-bus that triggers an I2C interrupt can wake up the part, provided that the interrupt is enabled in the INTENSET register and the NVIC. As long as the I2C clock I2C_PCLK remains active in sleep mode, the I2C can wake up the part independently of whether the I2C block is configured in master or slave mode.

In Deep-sleep or Power-down mode, the I2C clock is turned off as are all peripheral clocks. However, if the I2C is configured in slave mode and an external master on the I2C-bus provides the clock signal, the I2C block can create an interrupt asynchronously. This interrupt, if enabled in the NVIC and in the I2C block's INTENCLR register, can then wake up the core.

26.4.3.1 Wake-up from Sleep mode

- Enable the I2C interrupt in the NVIC.
- Enable the I2C wake-up event in the I2C INTENSET register. Wake-up on any enabled interrupts is supported (see the INTENSET register). Examples are the following events:
  - Master pending
  - Change to idle state
  - Start/stop error
  - Slave pending
  - Address match (in slave mode)
  - Data available/ready

26.4.3.2 Wake-up from Deep-sleep and Power-down modes

- Enable the I2C interrupt in the NVIC.
- Enable the I2C interrupt in the STARTERP1 register in the SYSCON block to create the interrupt signal asynchronously while the core and the peripheral are not clocked. See Table 77 "Start logic 1 wake-up enable register (STARTERP1, address 0x4007 421C) bit description".

- In the PDAWAKE register, configure all peripherals that need to be running when the part wakes up.
- Configure the I2C in slave mode.
- Enable the I2C the interrupt in the I2C INTENCLR register which configures the interrupt as wake-up event. Examples are the following events:
  - Slave deselect
  - Slave pending (wait for read, write, or ACK)
  - Address match
  - Data available/ready for the monitor

26.5 General description

The architecture of the I2C-bus interface is shown in Figure 75.

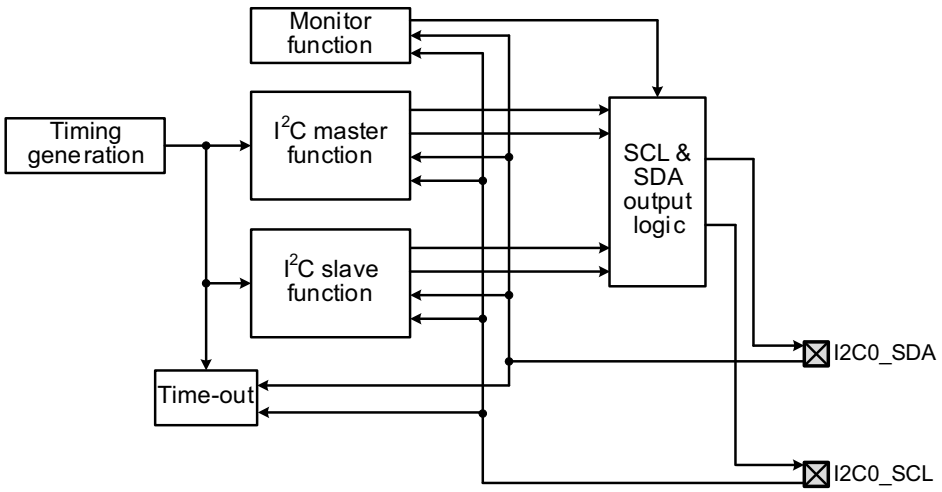


Fig 75. I2C block diagram

26.6 Register description

The register functions can be grouped as follows:

- Common registers:
  - Table 375 “I2C Configuration register (CFG, address 0x4005 0000) bit description”
  - Table 376 “I2C Status register (STAT, address 0x4005 0004) bit description”
  - Table 383 “I2C Interrupt Status register (INTSTAT, address 0x4005 0018) bit description”
  - Table 379 “Interrupt Enable Set and read register (INTENSET, address 0x4005 0008) bit description”
  - Table 380 “Interrupt Enable Clear register (INTENCLR, address 0x4005 000C) bit description”
  - Table 381 “Time-out value register (TIMEOUT, address 0x4005 0010) bit description”

- Table 382 “I2C Clock Divider register (CLKDIV, address 0x4005 0014) bit description”

- Master function registers:
  - Table 384 “Master Control register (MSTCTL, address 0x4005 0020) bit description”
  - Table 385 “Master Time register (MSTTIME, address 0x4005 0024) bit description”
  - Table 386 “Master Data register (MSTDAT, address 0x4005 0028) bit description”
- Slave function registers:
  - Table 387 “Slave Control register (SLVCTL, address 0x4005 0040) bit description”
  - Table 387 “Slave Control register (SLVCTL, address 0x4005 0040) bit description”
  - Table 389 “Slave Address registers (SLVADR[0:3], address 0x4005 0048 (SLVADR0) to 0x4005 0054 (SLVADR3)) bit description”
  - Table 390 “Slave address Qualifier 0 register (SLVQUAL0, address 0x4005 0058) bit description”
- Monitor function register: Table 391 “Monitor data register (MONRXDAT, address 0x4005 0080) bit description”

Table 374: Register overview: I2C (base address 0x4005 0000)

Name	Access	Offset	Description	Reset value	Reference
CFG	R/W	0x00	Configuration for shared functions.	0	Table 375
STAT	R/W	0x04	Status register for Master, Slave, and Monitor functions.	0x00080 1	Table 376
INTENSET	R/W	0x08	Interrupt Enable Set and read register.	0	Table 379
INTENCLR	W	0x0C	Interrupt Enable Clear register.	NA	Table 380
TIMEOUT	R/W	0x10	Time-out value register.	0xFFFF	Table 381
CLKDIV	R/W	0x14	Clock pre-divider for the entire I^2C block. This determines what time increments are used for the MSTTIME and SLVTIME registers.	0	Table 382
INTSTAT	R	0x18	Interrupt Status register for Master, Slave, and Monitor functions.	0	Table 383
MSTCTL	R/W	0x20	Master control register.	0	Table 384
MSTTIME	R/W	0x24	Master timing configuration.	0x77	Table 385
MSTDAT	R/W	0x28	Combined Master receiver and transmitter data register.	NA	Table 386
SLVCTL	R/W	0x40	Slave control register.	0	Table 387
SLVDAT	R/W	0x44	Combined Slave receiver and transmitter data register.	NA	Table 388
SLVADR0	R/W	0x48	Slave address 0.	0x01	Table 389
SLVADR1	R/W	0x4C	Slave address 1.	0x01	Table 389
SLVADR2	R/W	0x50	Slave address 2.	0x01	Table 389
SLVADR3	R/W	0x54	Slave address 3.	0x01	Table 389
SLVQUAL0	R/W	0x58	Slave Qualification for address 0.	0	Table 390
MONRXDAT	RO	0x80	Monitor receiver data register.	0	Table 391

26.6.1 I2C Configuration register

The CFG register contains mode settings that apply to Master, Slave, and Monitor functions.

Table 375. I2C Configuration register (CFG, address 0x4005 0000) bit description

Bit	Symbol	Value	Description	Reset Value
0	MSTEN		Master Enable. When disabled, configurations settings for the Master function are not changed, but the Master function is internally reset.	0
		0	Disabled. The I^2C Master function is disabled.	
		1	Enabled. The I^2C Master function is enabled.	
1	SLVEN		Slave Enable. When disabled, configurations settings for the Slave function are not changed, but the Slave function is internally reset.	0
		0	Disabled. The I^2C slave function is disabled.	
		1	Enabled. The I^2C slave function is enabled.	

Table 375. I2C Configuration register (CFG, address 0x4005 0000) bit description

Bit	Symbol	Value	Description	Reset Value
2	MONEN		Monitor Enable. When disabled, configurations settings for the Monitor function are not changed, but the Monitor function is internally reset.	0
		0	Disabled. The I^2C monitor function is disabled.	
		1	Enabled. The I^2C monitor function is enabled.	
3	TIMEOUTEN		I^2C bus Time-out Enable. When disabled, the time-out function is internally reset.	0
		0	Disabled. Time-out function is disabled.	
		1	Enabled. Time-out function is enabled. Both types of time-out flags will be generated and will cause interrupts if they are enabled. Typically, only one time-out will be used in a system.	
4	MONCLKSTR		Monitor function Clock Stretching.	0
		0	Disabled. The monitor function will not perform clock stretching. Software or DMA may not always be able to read data provided by the monitor function before it is overwritten. This mode may be used when non-invasive monitoring is critical.	
		1	Enabled. The monitor function will perform clock stretching in order to ensure that software or DMA can read all incoming data supplied by the monitor function.	
31:5	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.2 I2C Status register

The STAT register provides status flags and state information about all of the functions of the I^2C block. Some information in this register is read-only and some flags can be cleared by writing a 1 to them.

Access to bits in this register varies. RO = Read-only, W1 = write 1 to clear.

Details on the master and slave states described in the MSTSTATE and SLVSTATE bits in this register are listed in Table 377 and Table 378.

Table 376. I^2C Status register (STAT, address 0x4005 0004) bit description

Bit	Symbol	Value	Description	Reset value	Access
0	MSTPENDING		Master Pending. Indicates that the Master is waiting to continue communication on the I2C-bus (pending) or is idle. When the master is pending, the MSTSTATE bits indicate what type of software service if any the master expects. This flag will cause an interrupt when set if, enabled via the INTENSET register. The MSTPENDING flag is not set when the DMA is handling an event (if the MSTDMA bit in the MSTCTL register is set). If the master is in the idle state, and no communication is needed, mask this interrupt.	1	RO
		0	In progress. Communication is in progress and the Master function is busy and cannot currently accept a command.		
		1	Pending. The Master function needs software service or is in the idle state. If the master is not in the idle state, it is waiting to receive or transmit data or the NACK bit.		
3:1	MSTSTATE		Master State code. The master state code reflects the master state when the MSTPENDING bit is set, that is the master is pending or in the idle state. Each value of this field indicates a specific required service for the Master function. All other values are reserved.	0	RO
		0x0	Idle. The Master function is available to be used for a new transaction.		
		0x1	Receive ready. Received data available (Master Receiver mode). Address plus Read was previously sent and Acknowledged by slave.		
		0x2	Transmit ready. Data can be transmitted (Master Transmitter mode). Address plus Write was previously sent and Acknowledged by slave.		
		0x3	NACK Address. Slave NACKed address.		
		0x4	NACK Data. Slave NACKed transmitted data.		
4	MSTARBLOSS		Master Arbitration Loss flag. This flag can be cleared by software writing a 1 to this bit. It is also cleared automatically a 1 is written to MSTCONTINUE.	0	W1
		0	No loss. No Arbitration Loss has occurred.		
		1	Arbitration loss. The Master function has experienced an Arbitration Loss. At this point, the Master function has already stopped driving the bus and gone to an idle state. Software can respond by doing nothing, or by sending a Start in order to attempt to gain control of the bus when it next becomes idle.		
5	-		Reserved. Read value is undefined, only zero should be written.	NA	NA

Table 376. I²C Status register (STAT, address 0x4005 0004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value	Access
6	MSTSTSTPERR		Master Start/Stop Error flag. This flag can be cleared by software writing a 1 to this bit. It is also cleared automatically a 1 is written to MSTCONTINUE.	0	W1
		0	No Start/Stop Error has occurred.		
		1	Start/stop error has occurred. The Master function has experienced a Start/Stop Error. A Start or Stop was detected at a time when it is not allowed by the I²C specification. The Master interface has stopped driving the bus and gone to an idle state, no action is required. A request for a Start could be made, or software could attempt to insure that the bus has not stalled.		
7	-		Reserved. Read value is undefined, only zero should be written.	NA	NA
8	SLVPENDING		Slave Pending. Indicates that the Slave function is waiting to continue communication on the I2C-bus and needs software service. This flag will cause an interrupt when set if enabled via INTENSET. The SLVPENDING flag is not set when the DMA is handling an event (if the SLVDMA bit in the SLVCTL register is set). The SLVPENDING flag is read-only and is automatically cleared when a 1 is written to the SLVCONTINUE bit in the MSTCTL register.	0	RO
		0	In progress. The Slave function does not currently need service.		
		1	Pending. The Slave function needs service. Information on what is needed can be found in the adjacent SLVSTATE field.		
10:9	SLVSTATE		Slave State code. Each value of this field indicates a specific required service for the Slave function. All other values are reserved.	0	RO
		0x0	Slave address. Address plus R/W received. At least one of the four slave addresses has been matched by hardware.		
		0x1	Slave receive. Received data is available (Slave Receiver mode).		
		0x2	Slave transmit. Data can be transmitted (Slave Transmitter mode).		
		0x3	Reserved.		
11	SLVNOTSTR		Slave Not Stretching. Indicates when the slave function is stretching the I²C clock. This is needed in order to gracefully invoke Deep Sleep or Power-down modes during slave operation. This read-only flag reflects the slave function status in real time.	1	RO
		0	Stretching. The slave function is currently stretching the I²C bus clock. Deep-Sleep or Power-down mode cannot be entered at this time.		
		1	Not stretching. The slave function is not currently stretching the I²C bus clock. Deep-sleep or Power-down mode could be entered at this time.		

Table 376. I²C Status register (STAT, address 0x4005 0004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value	Access
13:12	SLVIDX		Slave address match Index. This field is valid when the I²C slave function has been selected by receiving an address that matches one of the slave addresses defined by any enabled slave address registers, and provides an identification of the address that was matched. It is possible that more than one address could be matched, but only one match can be reported here.	0	RO
		0x0	Slave address 0 was matched.		
		0x1	Slave address 1 was matched.		
		0x2	Slave address 2 was matched.		
		0x3	Slave address 3 was matched.		
14	SLVSEL		Slave selected flag. SLVSEL is set after an address match when software tells the Slave function to acknowledge the address. It is cleared when another address cycle presents an address that does not match an enabled address on the Slave function, when slave software decides to NACK a matched address, or when there is a Stop detected on the bus. SLVSEL is not cleared if software Nacks data.	0	RO
		0	Not selected. The Slave function is not currently selected.		
		1	Selected. The Slave function is currently selected.		
15	SLVDESEL		Slave Deselected flag. This flag will cause an interrupt when set if enabled via INTENSET. This flag can be cleared by writing a 1 to this bit.	0	W1
		0	Not deselected. The Slave function has not become deselected. This does not mean that it is currently selected. That information can be found in the SLVSEL flag.		
		1	Deselected. The Slave function has become deselected. This is specifically caused by the SLVSEL flag changing from 1 to 0. See the description of SLVSEL for details on when that event occurs.		
16	MONRDY		Monitor Ready. This flag is cleared when the MONRXDAT register is read.	0	RO
		0	No data. The Monitor function does not currently have data available.		
		1	Data waiting. The Monitor function has data waiting to be read.		
17	MONOV		Monitor Overflow flag.	0	W1
		0	No overrun. Monitor data has not overrun.		
		1	Overrun. A Monitor data overrun has occurred. This can only happen when Monitor clock stretching not enabled via the MONCLKSTR bit in the CFG register. Writing 1 to this bit clears the flag.		
18	MONACTIVE		Monitor Active flag. This flag indicates when the Monitor function considers the I²C bus to be active. Active is defined here as when some Master is on the bus: a bus Start has occurred more recently than a bus Stop.	0	RO
		0	Inactive. The Monitor function considers the I²C bus to be inactive.		
		1	Active. The Monitor function considers the I²C bus to be active.		

Table 376. I²C Status register (STAT, address 0x4005 0004) bit description *…continued*

Bit	Symbol	Value	Description	Reset value	Access
19	MONIDLE		Monitor Idle flag. This flag is set when the Monitor function sees the I²C bus change from active to inactive. This can be used by software to decide when to process data accumulated by the Monitor function. This flag will cause an interrupt when set if enabled via the INTENSET register . The flag can be cleared by writing a 1 to this bit.	0	W1
		0	Not idle. The I²C bus is not idle, or this flag has been cleared by software.		
		1	Idle. The I²C bus has gone idle at least once since the last time this flag was cleared by software.		
23:20	-		Reserved. Read value is undefined, only zero should be written.	NA	NA
24	EVENTTIMEOUT		Event Time-out Interrupt flag. Indicates when the time between events has been longer than the time specified by the TIMEOUT register. Events include Start, Stop, and clock edges. The flag is cleared by writing a 1 to this bit. No time-out is created when the I2C-bus is idle.	0	W1
		0	No time-out. I²C bus events have not caused a time-out.		
		1	Event time-out. The time between I²C bus events has been longer than the time specified by the I2C TIMEOUT register.		
25	SCLTIMEOUT		SCL Time-out Interrupt flag. Indicates when SCL has remained low longer than the time specific by the TIMEOUT register. The flag is cleared by writing a 1 to this bit.	0	W1
		0	No time-out. SCL low time has not caused a time-out.		
		1	Time-out. SCL low time has caused a time-out.		
31:26	-		Reserved. Read value is undefined, only zero should be written.	NA	NA

Table 377. Master function state codes (MSTSTATE)

MSTSTATE	Description	Actions	DMA access allowed
0x0	**Idle.** The Master function is available to be used for a new transaction.	Send a Start or disable MSTPENDING interrupt if the Master function is not needed currently.	No
0x1	**Received data is available (Master Receiver mode).** Address plus Read was previously sent and Acknowledged by slave.	Read data and either continue, send a Stop, or send a Repeated Start.	Yes
0x2	**Data can be transmitted (Master Transmitter mode).** Address plus Write was previously sent and Acknowledged by slave.	Send data and continue, or send a Stop or Repeated Start.	Yes
0x3	**Slave NACKed address.**	Send a Stop or Repeated Start.	No
0x4	**Slave NACKed transmitted data.**	Send a Stop or Repeated Start.	No

Table 378. Slave function state codes (SLVSTATE)

SLVSTATE		Description	Actions	DMA access allowed
0	SLVST_ADDR	**Address plus R/W received.** At least one of the 4 slave addresses has been matched by hardware.	Software can further check the address if needed, for instance if a subset of addresses qualified by SLVQUAL0 is to be used. Software can ACK or NACK the address by writing 1 to either SLVCONTINUE or SLVNACK. Also see Section 26.7.3 regarding 10-bit addressing.	No
1	SLVST_RX	**Received data is available (Slave Receiver mode).**	Read data reply with an ACK or a NACK.	Yes
2	SLVST_TX	**Data can be transmitted (Slave Transmitter mode).**	Send data.	Yes
3	-	Reserved.	-	-

26.6.3 Interrupt Enable Set and read register

The INTENSET register controls which I^2C status flags generate interrupts. Writing a 1 to a bit position in this register enables an interrupt in the corresponding position in the STAT register, if an interrupt is supported there. Reading INTENSET indicates which interrupts are currently enabled.

Table 379. Interrupt Enable Set and read register (INTENSET, address 0x4005 0008) bit description

Bit	Symbol	Value	Description	Reset value
0	MSTPENDINGEN		Master Pending interrupt Enable.	0
		0	The MstPending interrupt is disabled.	
		1	The MstPending interrupt is enabled.	
3:1	-		Reserved. Read value is undefined, only zero should be written.	NA
4	MSTARBLOSSEN		Master Arbitration Loss interrupt Enable.	0
		0	The MstArbLoss interrupt is disabled.	
		1	The MstArbLoss interrupt is enabled.	
5	-		Reserved. Read value is undefined, only zero should be written.	NA
6	MSTSTSTPERREN		Master Start/Stop Error interrupt Enable.	0
		0	The MstStStpErr interrupt is disabled.	
		1	The MstStStpErr interrupt is enabled.	
7	-		Reserved. Read value is undefined, only zero should be written.	NA
8	SLVPENDINGEN		Slave Pending interrupt Enable.	0
		0	The SlvPending interrupt is disabled.	
		1	The SlvPending interrupt is enabled.	
10:9	-		Reserved. Read value is undefined, only zero should be written.	NA

Table 379. Interrupt Enable Set and read register (INTENSET, address 0x4005 0008) bit description

Bit	Symbol	Value	Description	Reset value
11	SLVNOTSTREN		Slave Not Stretching interrupt Enable.	0
		0	The SlvNotStr interrupt is disabled.	
		1	The SlvNotStr interrupt is enabled.	
14:12	-		Reserved. Read value is undefined, only zero should be written.	NA
15	SLVDESELEN		Slave Deselect interrupt Enable.	0
		0	The SlvDeSel interrupt is disabled.	
		1	The SlvDeSel interrupt is enabled.	
16	MONRDYEN		Monitor data Ready interrupt Enable.	0
		0	The MonRdy interrupt is disabled.	
		1	The MonRdy interrupt is enabled.	
17	MONOVEN		Monitor Overrun interrupt Enable.	0
		0	The MonOv interrupt is disabled.	
		1	The MonOv interrupt is enabled.	
18	-		Reserved. Read value is undefined, only zero should be written.	NA
19	MONIDLEEN		Monitor Idle interrupt Enable.	0
		0	The MonIdle interrupt is disabled.	
		1	The MonIdle interrupt is enabled.	
23:20	-		Reserved. Read value is undefined, only zero should be written.	NA
24	EVENTTIMEOUTEN		Event time-out interrupt Enable.	0
		0	The Event time-out interrupt is disabled.	
		1	The Event time-out interrupt is enabled.	
25	SCLTIMEOUTEN		SCL time-out interrupt Enable.	0
		0	The SCL time-out interrupt is disabled.	
		1	The SCL time-out interrupt is enabled.	
31:26	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.4 Interrupt Enable Clear register

Writing a 1 to a bit position in INTENCLR clears the corresponding position in the INTENSET register, disabling that interrupt. INTENCLR is a write-only register.

Bits that do not correspond to defined bits in INTENSET are reserved and only zeroes should be written to them.

Table 380. Interrupt Enable Clear register (INTENCLR, address 0x4005 000C) bit description

Bit	Symbol	Description	Reset value
0	MSTPENDINGCLR	Master Pending interrupt clear. Writing 1 to this bit clears the corresponding bit in the INTENSET register if implemented.	0
3:1	-	Reserved. Read value is undefined, only zero should be written.	NA
4	MSTARBLOSSCLR	Master Arbitration Loss interrupt clear.	0
5	-	Reserved. Read value is undefined, only zero should be written.	NA
6	MSTSTSTPERRCLR	Master Start/Stop Error interrupt clear.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	SLVPENDINGCLR	Slave Pending interrupt clear.	0
10:9	-	Reserved. Read value is undefined, only zero should be written.	NA
11	SLVNOTSTRCLR	Slave Not Stretching interrupt clear.	0
14:12	-	Reserved. Read value is undefined, only zero should be written.	NA
15	SLVDESELCLR	Slave Deselect interrupt clear.	0
16	MONRDYCLR	Monitor data Ready interrupt clear.	0
17	MONOVCLR	Monitor Overrun interrupt clear.	0
18	-	Reserved. Read value is undefined, only zero should be written.	NA
19	MONIDLECLR	Monitor Idle interrupt clear.	0
23:20	-	Reserved. Read value is undefined, only zero should be written.	NA
24	EVENTTIMEOUTCLR	Event time-out interrupt clear.	0
25	SCLTIMEOUTCLR	SCL time-out interrupt clear.	0
31:26	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.5 Time-out value register

The TIMEOUT register allows setting an upper limit to certain I^2C bus times, informing by status flag and/or interrupt when those times are exceeded.

Two time-outs are generated, and software can elect to use either of them.

1. EVENTTIMEOUT checks the time between bus events while the bus is not idle: Start, SCL rising, SCL falling, and Stop. The EVENTTIMEOUT status flag in the STAT register is set if the time between any two events becomes longer than the time configured in the TIMEOUT register. The EVENTTIMEOUT status flag can cause an interrupt if enabled to do so by the EVENTTIMEOUTEN bit in the INTENSET register.

2. SCLTIMEOUT checks only the time that the SCL signal remains low while the bus is not idle. The SCLTIMEOUT status flag in the STAT register is set if SCL remains low longer than the time configured in the TIMEOUT register. The SCLTIMEOUT status flag can cause an interrupt if enabled to do so by the SCLTIMEOUTEN bit in the INTENSET register. The SCLTIMEOUT can be used with the SMBus.

Also see Section 26.7.2 “Time-out”.

Table 381. Time-out value register (TIMEOUT, address 0x4005 0010) bit description

Bit	Symbol	Description	Reset value
3:0	TOMIN	Time-out time value, bottom four bits. These are hard-wired to 0xF. This gives a minimum time-out of 16 I^2C function clocks and also a time-out resolution of 16 I^2C function clocks.	0xF
15:4	TO	Time-out time value. Specifies the time-out interval value in increments of 16 I^2C function clocks, as defined by the CLKDIV register. To change this value while I^2C is in operation, disable all time-outs, write a new value to TIMEOUT, then re-enable time-outs. 0x000 = A time-out will occur after 16 counts of the I^2C function clock. 0x001 = A time-out will occur after 32 counts of the I^2C function clock. ... 0xFFF = A time-out will occur after 65,536 counts of the I^2C function clock.	0xFFF
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.6 Clock Divider register

The CLKDIV register divides down the Peripheral Clock (PCLK) to produce the I^2C function clock that is used to time various aspects of the I^2C interface. The I^2C function clock is used for some internal operations in the I^2C block and to generate the timing required by the I^2C bus specification, some of which are user configured in the MSTTIME register for Master operation and the SLVTIME register for Slave operation.

See Section 26.7.1.1 “Rate calculations” for details on bus rate setup.

Table 382. I^2C Clock Divider register (CLKDIV, address 0x4005 0014) bit description

Bit	Symbol	Description	Reset value
15:0	DIVVAL	This field controls how the clock (PCLK) is used by the I^2C functions that need an internal clock in order to operate. 0x0000 = PCLK is used directly by the I^2C function. 0x0001 = PCLK is divided by 2 before use by the I^2C function. 0x0002 = PCLK is divided by 3 before use by the I^2C function. ... 0xFFFF = PCLK is divided by 65,536 before use by the I^2C function.	0
31:16	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.7 Interrupt Status register

The INTSTAT register provides register provides a view of those interrupt flags that are currently enabled. This can simplify software handling of interrupts. See Table 376 for detailed descriptions of the interrupt flags.

Table 383. I²C Interrupt Status register (INTSTAT, address 0x4005 0018) bit description

Bit	Symbol	Description	Reset value
0	MSTPENDING	Master Pending.	1
3:1	-	Reserved.	
4	MSTARBLOSS	Master Arbitration Loss flag.	0
5	-	Reserved. Read value is undefined, only zero should be written.	NA
6	MSTSTSTPERR	Master Start/Stop Error flag.	0
7	-	Reserved. Read value is undefined, only zero should be written.	NA
8	SLVPENDING	Slave Pending.	0
10:9	-	Reserved. Read value is undefined, only zero should be written.	NA
11	SLVNOTSTR	Slave Not Stretching status.	1
14:12	-	Reserved. Read value is undefined, only zero should be written.	NA
15	SLVDESEL	Slave Deselected flag.	0
16	MONRDY	Monitor Ready.	0
17	MONOV	Monitor Overflow flag.	0
18	-	Reserved. Read value is undefined, only zero should be written.	NA
19	MONIDLE	Monitor Idle flag.	0
23:20	-	Reserved. Read value is undefined, only zero should be written.	NA
24	EVENTTIMEOUT	Event time-out Interrupt flag.	0
25	SCLTIMEOUT	SCL time-out Interrupt flag.	0
31:26	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.8 Master Control register

The MSTCTL register contains bits that control various functions of the I²C Master interface. Only write to this register when the master is pending (MSTPENDING = 1 in the STAT register, Table 376).

Software should always write a complete value to MSTCTL, and not OR new control bits into the register as is possible in other registers such as CFG. This is due to the fact that MSTSTART and MSTSTOP are not self-clearing flags. ORing in new data following a Start or Stop may cause undesirable side effects.

After an initial I2C Start, MSTCTL should generally only be written when the MSTPENDING flag in the STAT register is set, after the last bus operation has completed. An exception is when DMA is being used and a transfer completes. In this case there is no

MSTPENDING flag, and the MSTDMA control bit would be cleared by software potentially at the same time as setting either the MSTSTOP or MSTSTART control bit.

Remark: When in the idle or slave NACKed states (see Table 377), set the MSTDMA bit either with or after the MSTCONTINUE bit. MSTDMA can be cleared at any time.

Table 384. Master Control register (MSTCTL, address 0x4005 0020) bit description

Bit	Symbol	Value	Description	Reset value
0	MSTCONTINUE		Master Continue. This bit is write-only.	0
		0	No effect.	
		1	Continue. Informs the Master function to continue to the next operation. This must done after writing transmit data, reading received data, or any other housekeeping related to the next bus operation.	
1	MSTSTART		Master Start control. This bit is write-only.	0
		0	No effect.	
		1	Start. A Start will be generated on the I^2C bus at the next allowed time.	
2	MSTSTOP		Master Stop control. This bit is write-only.	0
		0	No effect.	
		1	Stop. A Stop will be generated on the I^2C bus at the next allowed time, preceded by a NACK to the slave if the master is receiving data from the slave (Master Receiver mode).	
3	MSTDMA		Master DMA enable. Data operations of the I^2C can be performed with DMA. Protocol type operations such as Start, address, Stop, and address match must always be done with software, typically via an interrupt. When a DMA data transfer is complete, MSTDMA must be cleared prior to beginning the next operation, typically a Start or Stop.This bit is read/write.	0
		0	Disable. No DMA requests are generated for master operation.	
		1	Enable. A DMA request is generated for I^2C master data operations. When this I^2C master is generating Acknowledge bits in Master Receiver mode, the acknowledge is generated automatically.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.9 Master Time

The MSTTIME register allows programming of certain times that may be controlled by the Master function. These include the clock (SCL) high and low times, repeated Start setup time, and transmitted data setup time.

The I2C clock pre-divider is described in Table 382.

Table 385. Master Time register (MSTTIME, address 0x4005 0024) bit description

Bit	Symbol	Value	Description	Reset value
2:0	MSTSCLLOW		Master SCL Low time. Specifies the minimum low time that will be asserted by this master on SCL. Other devices on the bus (masters or slaves) could lengthen this time. This corresponds to the parameter t_{LOW} in the I^2C bus specification. I^2C bus specification parameters t_{BUF} and $t_{SU;STA}$ have the same values and are also controlled by MSTSCLLOW.	0
		0x0	2 clocks. Minimum SCL low time is 2 clocks of the I^2C clock pre-divider.	
		0x1	3 clocks. Minimum SCL low time is 3 clocks of the I^2C clock pre-divider.	
		0x2	4 clocks. Minimum SCL low time is 4 clocks of the I^2C clock pre-divider.	
		0x3	5 clocks. Minimum SCL low time is 5 clocks of the I^2C clock pre-divider.	
		0x4	6 clocks. Minimum SCL low time is 6 clocks of the I^2C clock pre-divider.	
		0x5	7 clocks. Minimum SCL low time is 7 clocks of the I^2C clock pre-divider.	
		0x6	8 clocks. Minimum SCL low time is 8 clocks of the I^2C clock pre-divider.	
		0x7	9 clocks. Minimum SCL low time is 9 clocks of the I^2C clock pre-divider.	
3	-		Reserved.	0

Table 385. Master Time register (MSTTIME, address 0x4005 0024) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
6:4	MSTSCLHIGH		Master SCL High time. Specifies the minimum high time that will be asserted by this master on SCL. Other masters in a multi-master system could shorten this time. This corresponds to the parameter t_{HIGH} in the I^2C bus specification. I^2C bus specification parameters $t_{SU;STO}$ and $t_{HD;STA}$ have the same values and are also controlled by MSTSCLHIGH.	0
		0x0	2 clocks. Minimum SCL high time is 2 clock of the I^2C clock pre-divider.	
		0x1	3 clocks. Minimum SCL high time is 3 clocks of the I^2C clock pre-divider .	
		0x2	4 clocks. Minimum SCL high time is 4 clock of the I^2C clock pre-divider.	
		0x3	5 clocks. Minimum SCL high time is 5 clock of the I^2C clock pre-divider.	
		0x4	6 clocks. Minimum SCL high time is 6 clock of the I^2C clock pre-divider.	
		0x5	7 clocks. Minimum SCL high time is 7 clock of the I^2C clock pre-divider.	
		0x6	8 clocks. Minimum SCL high time is 8 clock of the I^2C clock pre-divider.	
		0x7	9 clocks. Minimum SCL high time is 9 clocks of the I^2C clock pre-divider.	
31:7	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.10 Master Data register

The MSTDAT register provides the means to read the most recently received data for the Master function, and to transmit data using the Master function.

Table 386. Master Data register (MSTDAT, address 0x4005 0028) bit description

Bit	Symbol	Description	Reset value
7:0	DATA	Master function data register. Read: read the most recently received data for the Master function. Write: transmit data using the Master function.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.11 Slave Control register

The SLVCTL register contains bits that control various functions of the I^2C Slave interface. Only write to this register when the slave is pending (SLVPENDING = 1 in the STAT register, Table 376).

Remark: When in the slave address state (slave state 0, see Table 378), set the SLVDMA bit either with or after the SLVCONTINUE bit. SLVDMA can be cleared at any time.

Table 387. Slave Control register (SLVCTL, address 0x4005 0040) bit description

Bit	Symbol	Value	Description	Reset Value
0	SLVCONTINUE		Slave Continue.	0
		0	No effect.	
		1	Continue. Informs the Slave function to continue to the next operation. This must done after writing transmit data, reading received data, or any other housekeeping related to the next bus operation.	
1	SLVNACK		Slave NACK.	0
		0	No effect.	
		1	NACK. Causes the Slave function to NACK the master when the slave is receiving data from the master (Slave Receiver mode).	
3	SLVDMA		Slave DMA enable.	0
		0	Disabled. No DMA requests are issued for Slave mode operation.	
		1	Enabled. DMA requests are issued for I^2C slave data transmission and reception.	
31:4	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.12 Slave Data register

The SLVDAT register provides the means to read the most recently received data for the Slave function and to transmit data using the Slave function.

Table 388. Slave Data register (SLVDAT, address 0x4005 0044) bit description

Bit	Symbol	Description	Reset Value
7:0	DATA	Slave function data register. Read: read the most recently received data for the Slave function. Write: transmit data using the Slave function.	0
31:8	-	Reserved. Read value is undefined, only zero should be written.	NA

26.6.13 Slave Address registers

The SLVADR[0:3] registers allow enabling and defining one of the addresses that can be automatically recognized by the I^2C slave hardware. The value in the SLVADR0 register is qualified by the setting of the SLVQUAL0 register.

When the slave address is compared to the receive address, the compare can be affected by the setting of the SLVQUAL0 register (see Section 26.6.14).

The I^2C slave function has 4 address comparators. The additional 3 address comparators do not include the address qualifier feature. For handling of the general call address, one of the 4 address registers can be programmed to respond to address 0.

Table 389. Slave Address registers (SLVADR[0:3], address 0x4005 0048 (SLVADR0) to 0x4005 0054 (SLVADR3)) bit description

Bit	Symbol	Value	Description	Reset value
0	SADISABLE		Slave Address n Disable.	1
		0	Enabled. Slave Address n is enabled and will be recognized with any changes specified by the SLVQUAL0 register.	
		1	Ignored Slave Address n is ignored.	
7:1	SLVADR		Seven bit slave address that is compared to received addresses if enabled.	0
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.14 Slave address Qualifier 0 register

The SLVQUAL0 register can alter how Slave Address 0 is interpreted.

Table 390. Slave address Qualifier 0 register (SLVQUAL0, address 0x4005 0058) bit description

Bit	Symbol	Value	Description	Reset Value
0	QUALMODE0		Reserved. Read value is undefined, only zero should be written.	0
		0	The SLVQUAL0 field is used as a logical mask for matching address 0.	
		1	The SLVQUAL0 field is used to extend address 0 matching in a range of addresses.	
7:1	SLVQUAL0		Slave address Qualifier for address 0. A value of 0 causes the address in SLVADR0 to be used as-is, assuming that it is enabled. If QUALMODE0 = 0, any bit in this field which is set to 1 will cause an automatic match of the corresponding bit of the received address when it is compared to the SLVADR0 register. If QUALMODE0 = 1, an address range is matched for address 0. This range extends from the value defined by SLVADR0 to the address defined by SLVQUAL0 (address matches when SLVADR0[7:1] <= received address <= SLVQUAL0[7:1]).	0
31:8	-		Reserved. Read value is undefined, only zero should be written.	NA

26.6.15 Monitor data register

The read-only MONRXDAT register provides information about events on the I^2C bus, primarily to facilitate debugging of the I^2C during application development. All data addresses and data passing on the bus and whether these were acknowledged, as well as Start and Stop events, are reported.

The Monitor function must be enabled by the MONEN bit in the CFG register. Monitor mode can be configured to stretch the I^2C clock if data is not read from the MONRXDAT register in time to prevent it, via the MONCLKSTR bit in the CFG register**.** This can help ensure that nothing is missed but can cause the monitor function to be somewhat intrusive (by potentially adding clock delays, depending on software or DMA response time). In order to improve the chance of collecting all Monitor information if clock stretching is not enabled, Monitor data is buffered such that it is available until the end of the next piece of information from the I^2C bus.

Table 391. Monitor data register (MONRXDAT, address 0x4005 0080) bit description

Bit	Symbol	Value	Description	Reset value
7:0	MONRXDAT		Monitor function Receiver Data. This reflects every data byte that passes on the I^2C pins, and adds indication of Start, Repeated Start, and data NACK.	0
8	MONSTART		Monitor Received Start.	0
		0	No detect. The monitor function has not detected a Start event on the I^2C bus.	
		1	Start detect. The monitor function has detected a Start event on the I^2C bus.	
9	MONRESTART		Monitor Received Repeated Start.	0
		0	No start detect. The monitor function has not detected a Repeated Start event on the I^2C bus.	
		1	Repeated start detect. The monitor function has detected a Repeated Start event on the I^2C bus.	
10	MONNACK		Monitor Received NACK.	0
		0	Acknowledged. The data currently being provided by the monitor function was acknowledged by at least one master or slave receiver.	
		1	Not acknowledged. The data currently being provided by the monitor function was not acknowledged by any receiver.	
31:11	-		Reserved. Read value is undefined, only zero should be written.	NA

26.7 Functional description

26.7.1 Bus rates and timing considerations

Due to the nature of the I^2C bus, it is generally not possible to guarantee a specific clock rate on the SCL pin. On the I2C-bus, the The clock can be stretched by any slave device, extended by software overhead time, etc. In a multi-master system, the master that provides the shortest SCL high time will cause that time to appear on SCL as long as that master is participating in I2C traffic (i.e. when it is the only master on the bus, or during arbitration between masters).

Rate calculations give a base frequency that represents the fastest that the I^2C bus could operate if nothing slows it down.

26.7.1.1 Rate calculations

SCL high time (in I^2C function clocks) = (CLKDIV + 1) * (MSTSCLHIGH + 2)

SCL low time (in I^2C function clocks) = (CLKDIV + 1) * (MSTSCLLOW + 2)

Nominal SCL rate = I^2C function clock rate / (SCL high time + SCL low time)

26.7.2 Time-out

A time-out feature on an I^2C interface can be used to detect a “stuck” bus and potentially do something to alleviate the condition. Two different types of time-out are supported. Both types apply whenever the I^2C block and the time-out function are both enabled, Master, Slave, or Monitor functions do not need to be enabled.

In the first type of time-out, reflected by the EVENTTIMEOUT flag in the STAT register, the time between bus events governs the time-out check. These events include Start, Stop, and all changes on the I^2C clock (SCL). This time-out is asserted when the time between any of these events is longer than the time configured in the TIMEOUT register. This time-out could be useful in monitoring an I^2C bus within a system as part of a method to keep the bus running of problems occur.

The second type of I^2C time-out is reflected by the SCLTIMEOUT flag in the STAT register. This time-out is asserted when the SCL signal remains low longer than the time configured in the TIMEOUT register. This corresponds to SMBus time-out parameter $T_{TIMEOUT}$. In this situation, a slave could reset its own I^2C interface in case it is the offending device. If all listening slaves (including masters that can be addressed as slaves) do this, then the bus will be released unless it is a current master causing the problem. Refer to the SMBus specification for more details.

Both types of time-out are generated when the I^2C bus is considered busy.

26.7.3 Ten-bit addressing

Ten-bit addressing is accomplished by the I^2C master sending a second address byte to extend a particular range of standard 7-bit addresses. In the case of the master writing to the slave, the I^2C frame simply continues with data after the 2 address bytes. For the master to read from a slave, it needs to reverse the data direction after the second address byte. This is done by sending a Repeated Start, followed by a repeat of the same standard 7-bit address, with a Read bit. The slave must remember that it had been addressed by the previous write operation and stay selected for the subsequent read with the correct partial I^2C address.

For the Master function, the I2C is simply instructed to perform the 2-byte addressing as a normal write operation, followed either by more write data, or by a Repeated Start with a repeat of the first part of the 10-bit slave address and then reading in the normal fashion.

For the Slave function, the first part of the address is automatically matched in the same fashion as 7-bit addressing. The Slave address qualifier feature (see Section 26.6.14) can be used to intercept all potential 10-bit addresses (first address byte values F0 through F6), or just one. In the case of Slave Receiver mode, data is received in the normal fashion after software matches the first data byte to the remaining portion of the 10-bit address. The Slave function should record the fact that it has been addressed, in case there is a follow-up read operation.

For Slave Transmitter mode, the slave function responds to the initial address in the same fashion as for Slave Receiver mode, and checks that it has previously been addressed with a full 10-bit address. If the address matched is address 0, and address qualification is enabled, software must check that the first part of the 10-bit address is a complete match to the previous address before acknowledging the address.

26.7.4 Clocking and power considerations

The Master function of the I^2C always requires a peripheral clock to be running in order to operate. The Slave function can operate without any internal clocking when the slave is not currently addressed. This means that reduced power modes up to Power-down mode can be entered, and the device will wake up when the I^2C Slave function recognizes an address. Monitor mode can similarly wake up the device from a reduced power mode when information becomes available.

26.7.5 Interrupt handling

The I2C provides a single interrupt output that handles all interrupts for Master, Slave, and Monitor functions.

26.7.6 DMA

Generally, data transfers can be handled by DMA for Master mode after an address is sent and acknowledged by a slave, and for Slave mode after software has acknowledged an address. In either mode, software is always involved in the address portion of a message. In master and slave modes, data receive and transmit data can be transferred by the DMA. The DMA supports three DMA requests: data transfer in master mode, slave mode, and monitor mode.

27.1 How to read this chapter

The C_CAN controller is available on all parts.

27.2 Features

- Conforms to protocol version 2.0 parts A and B.
- Supports bit rate of up to 1 Mbit/s.
- Supports 32 Message Objects.
- Each Message Object has its own identifier mask.
- Provides programmable FIFO mode (concatenation of Message Objects).
- Provides maskable interrupts.
- Supports Disabled Automatic Retransmission (DAR) mode for time-triggered CAN applications.
- Provides programmable loop-back mode for self-test operation.

27.3 Basic configuration

The C_CAN0 is configured using the following registers:

1. Power: In the SYSAHBCLKCTRL1 register, set bit 7 (Table 51).
2. Clocking: For an accurate peripheral clock to the C_CAN0 block, select the system oscillator as clock source for the system clock. Do not select the IRC if C_CAN0 baud rates above 100 kbit/s are required.
3. Use the PRESETCTRL1 register (Table 36) reset the C_CAN0 block.
4. Enable/disable the C_CAN0 interrupt in interrupt slot #27 in the NVIC.

The peripheral clock PCLK to the C_CAN0 (the C_CAN system clock) and to the programmable C_CAN0 clock divider (see Table 423) is provided by the system clock (see Table 49). This clock can be disabled through bit 7 in the SYSAHBCLKCTRL1 register for power savings.

Remark: If C_CAN0 baudrates above 100 kbit/s are required, the system oscillator must be selected as the clock source for the system clock. For lower baudrates, the IRC may also be used as clock source.

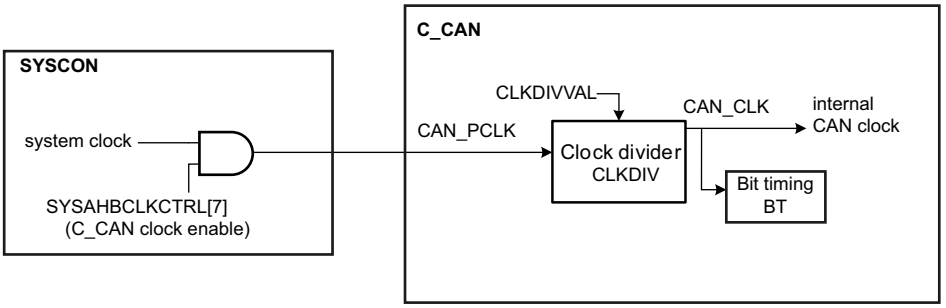


Fig 76. C_CAN clocking

27.4 General description

Controller Area Network (CAN) is the definition of a high performance communication protocol for serial data communication. The C_CAN controller is designed to provide a full implementation of the CAN protocol according to the CAN Specification Version 2.0B. The C_CAN controller allows to build powerful local networks with low-cost multiplex wiring by supporting distributed real-time control with a very high level of security.

The CAN controller consists of a CAN core, message RAM, a message handler, control registers, and the APB interface.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the CAN controller can be accessed directly by an external CPU via the APB bus. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

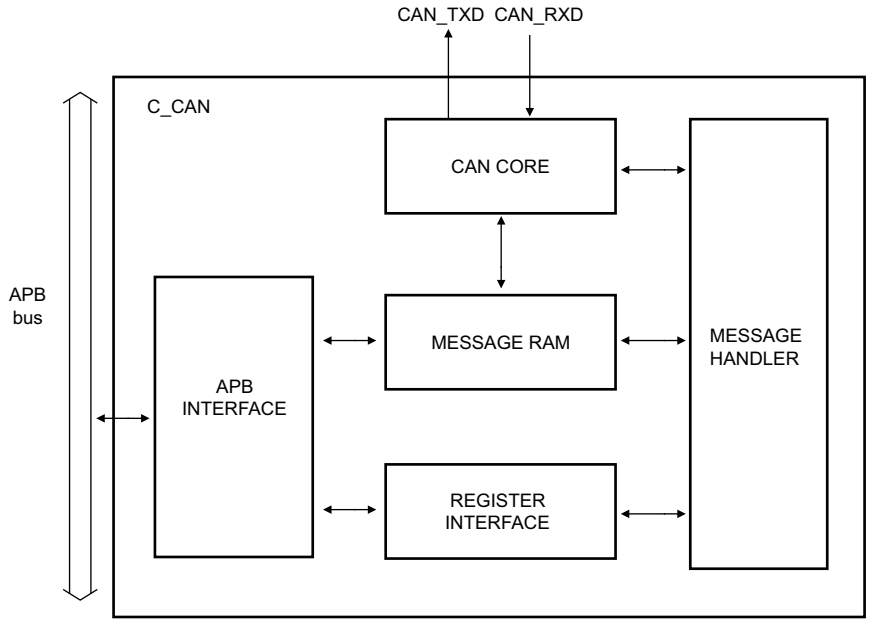


Fig 77. C_CAN block diagram

27.5 Pin description

The C_CAN pins are movable functions and are assigned to pins through the switch matrix.

See Section 8.3.1 “Connect an internal signal to a package pin” to assign the C_CAN pins to any pin on the package.

Table 392. C_CAN pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
CAN_TD1_O	O	external to pin	any	PINASSIGN6	Table 113	Transmit
CAN_RD1_I	I	external to pin	any	PINASSIGN6	Table 113	Receive

27.6 Register description

The C_CAN registers are organized as 32-bit wide registers.

The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

Table 393. Register overview: CCAN (base address 0x400F 0000)

Name	Access	Address offset	Description	Reset value	Reference
CNTL	R/W	0x000	CAN control	0x0001	Table 394
STAT	R/W	0x004	Status register	0x0000	Table 395
EC	RO	0x008	Error counter	0x0000	Table 396
BT	R/W	0x00C	Bit timing register	0x2301	Table 397
INT	RO	0x010	Interrupt register	0x0000	Table 398
TEST	R/W	0x014	Test register	-	Table 399
BRPE	R/W	0x018	Baud rate prescaler extension register	0x0000	Table 400
-	-	0x01C	Reserved	-	
IF1_CMDREQ	R/W	0x020	Message interface 1 command request	0x0001	Table 403
IF1_CMDMSK_W	R/W	0x024	Message interface 1 command mask (write direction)	0x0000	Table 404
IF1_CMDMSK_R	R/W	0x024	Message interface 1 command mask (read direction)	0x0000	Table 405
IF1_MSK1	R/W	0x028	Message interface 1 mask 1	0xFFFF	Table 406
IF1_MSK2	R/W	0x02C	Message interface 1 mask 2	0xFFFF	Table 407
IF1_ARB1	R/W	0x030	Message interface 1 arbitration 1	0x0000	Table 408
IF1_ARB2	R/W	0x034	Message interface 1 arbitration 2	0x0000	Table 409
IF1_MCTRL	R/W	0x038	Message interface 1 message control	0x0000	Table 410
IF1_DA1	R/W	0x03C	Message interface 1 data A1	0x0000	Table 411
IF1_DA2	R/W	0x040	Message interface 1 data A2	0x0000	Table 412
IF1_DB1	R/W	0x044	Message interface 1 data B1	0x0000	Table 413
IF1_DB2	R/W	0x048	Message interface 1 data B2	0x0000	Table 414
-	-	0x04C - 0x07C	Reserved	-	
IF2_CMDREQ	R/W	0x080	Message interface 2 command request	0x0001	Table 403
IF2_CMDMSK_W	R/W	0x084	Message interface 2 command mask (write direction)	0x0000	Table 404
IF2_CMDMSK_R	R/W	0x084	Message interface 2 command mask (read direction)	0x0000	Table 405
IF2_MSK1	R/W	0x088	Message interface 2 mask 1	0xFFFF	Table 406
IF2_MSK2	R/W	0x08C	Message interface 2 mask 2	0xFFFF	Table 407
IF2_ARB1	R/W	0x090	Message interface 2 arbitration 1	0x0000	Table 408
IF2_ARB2	R/W	0x094	Message interface 2 arbitration 2	0x0000	Table 409
IF2_MCTRL	R/W	0x098	Message interface 2 message control	0x0000	Table 410
IF2_DA1	R/W	0x09C	Message interface 2 data A1	0x0000	Table 411
IF2_DA2	R/W	0x0A0	Message interface 2 data A2	0x0000	Table 412
IF2_DB1	R/W	0x0A4	Message interface 2 data B1	0x0000	Table 413
IF2_DB2	R/W	0x0A8	Message interface 2 data B2	0x0000	Table 414
-	-	0x0AC - 0x0FC	Reserved	-	
TXREQ1	RO	0x100	Transmission request 1	0x0000	Table 415
TXREQ2	RO	0x104	Transmission request 2	0x0000	Table 416

Table 393. Register overview: CCAN (base address 0x400F 0000)

Name	Access	Address offset	Description	Reset value	Reference
-	-	0x108 - 0x11C	Reserved	-	
ND1	RO	0x120	New data 1	0x0000	Table 417
ND2	RO	0x124	New data 2	0x0000	Table 418
-	-	0x128 - 0x13C	Reserved	-	
IR1	RO	0x140	Interrupt pending 1	0x0000	Table 419
IR2	RO	0x144	Interrupt pending 2	0x0000	Table 420
-	-	0x148 - 0x15C	Reserved	-	
MSGV1	RO	0x160	Message valid 1	0x0000	Table 421
MSGV2	RO	0x164	Message valid 2	0x0000	Table 422
-	-	0x168 - 0x17C	Reserved	-	
CLKDIV	R/W	0x180	Can clock divider register	0x0001	Table 423

27.6.1 CAN protocol registers

27.6.1.1 CAN control register

The reset value 0x0001 of the CANCTL register enables initialization by software (INIT = 1). The C_CAN does not influence the CAN bus until the CPU resets the INIT bit to 0.

Table 394. CAN control registers (CNTL, address 0x400F 0000) bit description

Bit	Symbol	Value	Description	Reset value	Access
0	INIT		Initialization	1	R/W
		0	Normal operation.		
		1	Started. Initialization is started. On reset, software needs to initialize the CAN controller.		
1	IE		Module interrupt enable	0	R/W
		0	Disable CAN interrupts. The interrupt line is always HIGH.		
		1	Enable CAN interrupts. The interrupt line is set to LOW and remains LOW until all pending interrupts are cleared.		
2	SIE		Status change interrupt enable	0	R/W
		0	Disable status change interrupts. No status change interrupt will be generated.		
		1	Enable status change interrupts. A status change interrupt will be generated when a message transfer is successfully completed or a CAN bus error is detected.		

Table 394. CAN control registers (CNTL, address 0x400F 0000) bit description
…continued

Bit	Symbol	Value	Description	Reset value	Access
3	EIE		Error interrupt enable	0	R/W
		0	Disable error interrupt. No error status interrupt will be generated.		
		1	Enable error interrupt. A change in the bits BOFF or EWARN in the CANSTAT registers will generate an interrupt.		
4	-	-	reserved	0	-
5	DAR		Disable automatic retransmission	0	R/W
		0	Enabled. Automatic retransmission of disturbed messages enabled.		
		1	Disabled. Automatic retransmission disabled.		
6	CCE		Configuration change enable	0	R/W
		0	No write access. The CPU has no write access to the bit timing register.		
		1	Write access. The CPU has write access to the CANBT register while the INIT bit is one.		
7	TEST		Test mode enable	0	R/W
		0	Normal operation.		
		1	Test mode.		
31:8	-		reserved	-	-

Remark: The busoff recovery sequence (see *CAN Specification Rev. 2.0*) cannot be shortened by setting or resetting the INIT bit. If the device goes into busoff state, it will set INIT, stopping all bus activities. Once INIT has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 × 11 consecutive HIGH/recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

During the waiting time after the resetting of INIT, each time a sequence of 11 HIGH/recessive bits has been monitored, a Bit0Error code is written to the Status Register CANSTAT, enabling the CPU to monitor the proceeding of the busoff recovery sequence and to determine whether the CAN bus is stuck at LOW/dominant or continuously disturbed.

27.6.1.2 CAN status register

A status interrupt is generated by bits BOFF, EWARN, RXOK, TXOK, or LEC. BOFF and EWARN generate an error interrupt, and RXOK, TXOK, and LEC generate a status change interrupt if EIE and SIE respectively are set to enabled in the CANCTRL register.

A change of bit EPASS and a write to RXOK, TXOK, or LEC will never create a status interrupt.

Reading the STAT register will clear the Status Interrupt value (0x8000) in the INT register.

Table 395. CAN status register (STAT, address 0x400F 0004) bit description

Bit	Symbol	Value	Description	Reset value	Access
2:0	LEC		Last error code Type of the last error to occur on the CAN bus.The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The unused code '111' may be written by the CPU to check for updates.	000	R/W
		0x0	**No error**.		
		0x1	**Stuff error**. More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.		
		0x2	**Form error**. A fixed format part of a received frame has the wrong format.		
		0x3	**AckError**. The message this CAN core transmitted was not acknowledged.		
		0x4	**Bit1Error**. During the transmission of a message (with the exception of the arbitration field), the device wanted to send a HIGH/recessive level (bit of logical value '1'), but the monitored bus value was LOW/dominant.		
		0x5	**Bit0Error**. During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a LOW/dominant level (data or identifier bit logical value '0'), but the monitored Bus value was HIGH/recessive. During busoff recovery this status is set each time a sequence of 11 HIGH/recessive bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at LOW/dominant or continuously disturbed).		
		0x6	**CRCError**. The CRC checksum was incorrect in the message received.		
		0x7	**Unused.** No CAN bus event was detected (written by the CPU).		
3	TXOK		Transmitted a message successfully This bit must be reset by the CPU. It is never reset by the CAN controller.	0	R/W
		0	No transmit. Since this bit was last reset by the CPU, no message has been successfully transmitted.		
		1	Successful transmit. Since this bit was last reset by the CPU, a message has been successfully transmitted (error free and acknowledged by at least one other node).		
4	RXOK		Received a message successfully This bit must be reset by the CPU. It is never reset by the CAN controller.	0	R/W
		0	No receive. Since this bit was last reset by the CPU, no message has been successfully received.		
		1	Successful receive.Since this bit was last set to zero by the CPU, a message has been successfully received independent of the result of acceptance filtering.		

Table 395. CAN status register (STAT, address 0x400F 0004) bit description

…continued

Bit	Symbol	Value	Description	Reset value	Access
5	EPASS		Error passive	0	RO
		0	Active. The CAN controller is in the error active state.		
		1	Passive. The CAN controller is in the error passive state as defined in the *CAN 2.0 specification*.		
6	EWARN		Warning status	0	RO
		0	Below limit. Both error counters are below the error warning limit of 96.		
		1	At limit. At least one of the error counters in the EC has reached the error warning limit of 96.		
7	BOFF		Busoff status	0	RO
		0	The CAN module is not in busoff.		
		1	The CAN controller is in busoff state.		
31:8	-	-	reserved		

27.6.1.3 CAN error counter

Table 396. CAN error counter (EC, address 0x400F 0008) bit description

Bit	Symbol	Value	Description	Reset value	Access
7:0	TEC7_0		Transmit error counter Current value of the transmit error counter (maximum value 255)	0	RO
14:8	REC6_0		Receive error counter Current value of the receive error counter (maximum value 127).	-	RO
15	RP		Receive error passive	-	RO
		0	Below error level. The receive counter is below the error passive level.		
		1	At error level. The receive counter has reached the error passive level as defined in the *CAN2.0 specification*.		
31:16	-	-	Reserved	-	-

27.6.1.4 CAN bit timing register

Hardware interprets the value programmed into the bits in this register as the bit value + 1.

Table 397. CAN bit timing register (BT, address 0x400F 000C) bit description

Bit	Symbol	Description	Reset value	Access
5:0	BRP	Baud rate prescaler The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are 0 to 63.	000001	R/W
7:6	SJW	(Re)synchronization jump width Valid programmed values are 0 to 3.	00	R/W
11:8	TSEG1	Time segment before the sample point Valid values are 1 to 15.	0011	R/W
14:12	TSEG2	Time segment after the sample point Valid values are 0 to 7.	010	R/W
31:15	-	Reserved	-	-

For example, with a system clock set to of 8 MHz, the reset value of 0x2301 configures the C_CAN for a bit rate of 500 kBit/s.

The registers are only writable if a configuration change is enabled in CANCTRL and the controller is initialized by software (bits CCE and INIT in the CAN Control Register are set).

For details on bit timing, see Section 27.7.5 and the *Bosch C_CAN user's manual, revision 1.2*.

Baud rate prescaler

The bit time quanta t_q are determined by the BRP value:

$t_q = BRP / f_{sys}$

(f_{sys} is the system clock to the C_CAN block).

Time segments 1 and 2

Time segments TSEG1 and TSEG2 determine the number of time quanta per bit time and the location of the sample point:

$t_{TSEG1/2} = t_q \times (TSEG1/2 + 1)$

Synchronization jump width

To compensate for phase shifts between clock oscillators of different bus controllers, any bus controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width t_{SJW} defines the maximum number of clock cycles a certain bit period may be shortened or lengthened by one re-synchronization:

$t_{SJW} = t_q \times (SJW + 1)$

27.6.1.5 CAN interrupt register

Table 398. CAN interrupt register (INT, address 0x400F 0010) bit description

Bit	Symbol	Description	Reset value	Access
15:0	INTID	0x0000 = No interrupt is pending. 0x0001 - 0x0020 = Number of message object which caused the interrupt. 0x0021 - 0x7FFF = Unused 0x8000 = Status interrupt 0x8001 - 0xFFFF = Unused	0	R
31:16	-	Reserved	-	-

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If INTID is different from 0x0000 and IE is set, the interrupt line to the CPU is active. The interrupt line remains active until INTID is back to value 0x0000 (the cause of the interrupt is reset) or until IE is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object' s interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's INTPND bit. The StatusInterrupt is cleared by reading the Status Register.

27.6.1.6 CAN test register

Write access to the Test Register is enabled by setting bit Test in the CAN Control Register.

The different test functions may be combined, but when TX[1:0] ≠ "00" is selected, the message transfer is disturbed.

Table 399. CAN test register (TEST, address 0x400F 0014) bit description

Bit	Symbol	Value	Description	Reset value	Access
1:0	-	-	Reserved		-
2	BASIC		Basic mode	0	R/W
		0	Disabled. Basic mode disabled.		
		1	Enabled. IF1 registers used as TX buffer, IF2 registers used as RX buffer.		
3	SILENT		Silent mode	0	R/W
		0	Normal operation.		
		1	Silent mode. The module is in silent mode.		
4	LBACK		Loop back mode	0	R/W
		0	Disabled. Loop back mode is disabled.		
		1	Enabled. Loop back mode is enabled.		

Table 399. CAN test register (TEST, address 0x400F 0014) bit description

Bit	Symbol	Value	Description	Reset value	Access
6:5	TX		Control of CAN_TXD pins	00	R/W
		0x0	Controller. Level at the CAN_TXD pin is controlled by the CAN controller. This is the value at reset.		
		0x1	Sample point. The sample point can be monitored at the CAN_TXD pin.		
		0x2	Low. CAN_TXD pin is driven LOW/dominant.		
		0x3	High. CAN_TXD pin is driven HIGH/recessive.		
7	RX		Monitors the actual value of the CAN_RXD pin.	0	R
		0	Recessive. The CAN bus is recessive (CAN_RXD = 1).		
		1	Dominant. The CAN bus is dominant (CAN_RXD = 0).		
31:8	-		R/W		-

27.6.1.7 CAN baud rate prescaler extension register

Table 400. CAN baud rate prescaler extension register (BRPE, address 0x400F 0018) bit description

Bit	Symbol	Description	Reset value	Access
3:0	BRPE	Baud rate prescaler extension By programming BRPE the Baud Rate Prescaler can be extended to values up to 1023. Hardware interprets the value as the value of BRPE (MSBs) and BRP (LSBs) plus one. Allowed values are 0 to 15.	0x0000	R/W
31:4	-	Reserved	-	-

27.6.2 Message interface registers

There are two sets of interface registers which are used to control the CPU access to the Message RAM. The interface registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see Section 27.6.2.1) or parts of the Message Object may be transferred between the Message RAM and the IFx Message Buffer registers in one single transfer.

The function of the two interface register sets is identical (except for test mode Basic). One set of registers may be used for data transfer to the Message RAM while the other set of registers may be used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other.

Each set of interface registers consists of message buffer registers controlled by their own command registers. The command mask register specifies the direction of the data transfer and which parts of a message object will be transferred. The command request register is used to select a message object in the message RAM as target or source for the transfer and to start the action specified in the command mask register.

Table 401. Message interface registers

IF1 register names	IF1 register set	IF2 register names	IF2 register set
IF1_CMDREQ	IF1 command request	IF2_CMDREQ	IF2 command request
IF1_CMDMASK	IF1 command mask	IF2_CMDMASK	IF2 command mask
IF1_MASK1	IF1 mask 1	IF2_MSK1	IF2 mask 1
IF1_MASK2	IF1 mask 2	IF2_MSK2	IF2 mask 2
IF1_ARB1	IF1 arbitration 1	IF2_ARB1	IF2 arbitration 1
IF1_ARB2	IF1 arbitration 2	IF2_ARB2	IF2 arbitration 2
IF1_MCTRL	IF1 message control	IF2_MCTRL	IF2 message control
IF1_DA1	IF1 data A1	IF2_DA1	IF2 data A1
IF1_DA2	IF1 data A2	IF2_DA2	IF2 data A2
IF1_DB1	IF1 data B1	IF2_DB1	IF2 data B1
IF1_DB2	IF1 data B2	IF2_DB2	IF2 data B2

There are 32 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects. The message objects are accessed through the IFx Interface Registers.

For details of message handling, see Section 27.7.3.

27.6.2.1 Message objects

A message object contains the information from the various bits in the message interface registers. Table 402 below shows a schematic representation of the structure of the message object. The bits of a message object and the respective interface register where this bit is set or cleared are shown. For bit functions see the corresponding interface register.

Table 402. Structure of a message object in the message RAM

UMASK	MSK[28:0]	MXTD	MDIR	EOB	NEWDAT	MSGLST	RXIE	TXIE	INTPND
IF1/2_MCTRL	IF1/2_MSK1/2			IF1/2_MCTRL					
RMTEN	**TXRQST**	**MSGVAL**	**ID[28:0]**	**XTD**	**DIR**	**DLC3**	**DLC2**	**DLC1**	**DLC0**
IF1/2_MCTRL		IF1/2_ARB1/2				IF1/2_MCTRL			
DATA0	**DATA1**	**DATA2**	**DATA3**	**DATA4**	**DATA5**	**DATA6**	**DATA7**		
IF1/2_DA1		IF1/2_DA2		IF1/2_DB1		IF1/2_DB2			

27.6.2.2 CAN message interface command request registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the BUSY bit is automatically set to ‘1’ and the signal CAN_WAIT_B is pulled LOW to notify the CPU that a transfer is in progress. After a wait time of 3 to 6 CAN_CLK periods, the transfer between the Interface Register and the Message RAM has completed. The BUSY bit is set back to zero and the signal CAN_WAIT_B is set back.

Table 403. CAN message interface command request registers (IF1_CMDREQ, address 0x400F 0020 and IF2_CMDREQ, address 0x400F 0080) bit description

Bit	Symbol	Value	Description	Reset Value	Access
5:0	MN		Message number 0x01 - 0x20 = Valid message numbers. The message object in the message RAM is selected for data transfer. 0x00 = Not a valid message number. This value is interpreted as 0x20.[1] 0x21 - 0x3F = Not a valid message number. This value is interpreted as 0x01 - 0x1F.[1]	0x00	R/W
14:6	-		reserved	-	-
15	BUSY		BUSY flag	0	RO
		0	Done. Set to zero by hardware when read/write action to this Command request register has finished.		
		1	Busy. Set to one by hardware when writing to this Command request register.		
31:16	-	-	Reserved	-	-

[1] When a message number that is not valid is written into the Command request registers, the message number will be transformed into a valid value and that message object will be transferred.

27.6.2.3 CAN message interface command mask registers

The control bits of the IFx Command Mask Register specify the transfer direction and select which of the IFx Message Buffer Registers are source or target of the data transfer.The functions of the register bits depend on the transfer direction (read or write) which is selected in the WR_RD bit (bit 7) of this Command mask register.

Select the WR_RD to

one for the Write transfer direction (write to message RAM)

zero for the Read transfer direction (read from message RAM)

Table 404. CAN message interface command mask registers (IF1_CMDMSK_W, address 0x400F 0024 and IF2_CMDMSK_W, address 0x400F 0084) bit description for write direction

Bit	Symbol	Value	Description	Reset value	Access
0	DATA_B		Access data bytes 4-7	0	R/W
		0	Unchanged. Data bytes 4-7 unchanged.		
		1	Transfer. Transfer data bytes 4-7 to message object.		
1	DATA_A		Access data bytes 0-3	0	R/W
		0	Unchanged. Data bytes 0-3 unchanged.		
		1	Transfer. Transfer data bytes 0-3 to message object.		

Table 404. CAN message interface command mask registers (IF1_CMDMSK_W, address 0x400F 0024 and IF2_CMDMSK_W, address 0x400F 0084) bit description for write direction *...continued*

Bit	Symbol	Value	Description	Reset value	Access
2	TXRQST		Access transmission request bit	0	R/W
		0	No transmission request. TXRQST bit unchanged in IF1/2_MCTRL. **Remark:** If a transmission is requested by programming this bit, the TXRQST bit in the CANIFn_MCTRL register is ignored.		
		1	Request a transmission. Set the TXRQST bit IF1/2_MCTRL.		
3	CLRINTPND	-	This bit is ignored in the write direction.	0	R/W
4	CTRL		Access control bits	0	R/W
		0	Unchanged. Control bits unchanged.		
		1	Transfer. Transfer control bits to message object		
5	ARB		Access arbitration bits	0	R/W
		0	Unchanged. Arbitration bits unchanged.		
		1	Transfer. Transfer Identifier, DIR, XTD, and MSGVAL bits to message object.		
6	MASK		Access mask bits	0	R/W
		0	Unchanged. Mask bits unchanged.		
		1	Transfer. Transfer Identifier MSK + MDIR + MXTD to message object.		
7	WR_RD	1	Write transfer Transfer data from the selected message buffer registers to the message object addressed by the command request register CANIFn_CMDREQ.	0	R/W
31:8	-	-	reserved	0	-

Table 405. CAN message interface command mask registers (IF1_CMDMSK_R, address 0x400F 0024 and IF2_CMDMSK_R, address 0x400F 0084) bit description for read direction

Bit	Symbol	Value	Description	Reset value	Access
0	DATA_B		Access data bytes 4-7	0	R/W
		0	Unchanged. Data bytes 4-7 unchanged.		
		1	Transfer. Transfer data bytes 4-7 to IFx message buffer register.		
1	DATA_A		Access data bytes 0-3	0	R/W
		0	Unchanged. Data bytes 0-3 unchanged.		
		1	Transfer. Transfer data bytes 0-3 to IFx message buffer.		

Table 405. CAN message interface command mask registers (IF1_CMDMSK_R, address 0x400F 0024 and IF2_CMDMSK_R, address 0x400F 0084) bit description for read direction *…continued*

Bit	Symbol	Value	Description	Reset value	Access
2	NEWDAT		Access new data bit	0	R/W
		0	Unchanged. NEWDAT bit remains unchanged. **Remark:** A read access to a message object can be combined with the reset of the control bits INTPND and NEWDAT in IF1/2_MCTRL. The values of these bits transferred to the IFx Message Control Register always reflect the status before resetting these bits.		
		1	Clear. Clear NEWDAT bit in the message object.		
3	CLRINTPND		Clear interrupt pending bit.	0	R/W
		0	Unchanged. INTPND bit remains unchanged.		
		1	Clear. Clear INTPND bit in the message object.		
4	CTRL		Access control bits	0	R/W
		0	Unchanged. Control bits unchanged.		
		1	Transfer. Transfer control bits to IFx message buffer.		
5	ARB		Access arbitration bits	0	R/W
		0	Unchanged. Arbitration bits unchanged.		
		1	Transfer. Transfer Identifier, DIR, XTD, and MSGVAL bits to IFx message buffer register.		
6	MASK		Access mask bits	0	R/W
		0	Unchanged. Mask bits unchanged.		
		1	Transfer. Transfer Identifier MSK + MDIR + MXTD to IFx message buffer register.		
7	WR_RD	0	Read transfer Transfer data from the message object addressed by the command request register to the selected message buffer registers CANIFn_CMDREQ.	0	R/W
31:8	-	-	reserved	0	-

27.6.2.4 IF1 and IF2 message buffer registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM.

27.6.2.4.1 CAN message interface mask 1 registers

Table 406. CAN message interface mask 1 registers (IF1_MSK1, address 0x400F 0028 and IF2_MASK1, address 0x400F 0088) bit description

Bit	Symbol	Value	Description	Reset value	Access
15:0	MSK15_0		Identifier mask [15:0]	0xFFFF	R/W
		0	Match. The corresponding bit in the identifier of the message cannot inhibit the match in the acceptance filtering.		
		1	Mask. The corresponding identifier bit is used for acceptance filtering.		
31:16	-	-	Reserved	0	-

27.6.2.4.2 CAN message interface mask 2 registers

Table 407. CAN message interface mask 2 registers (IF1_MSK2, address 0x400F 002C and IF2_MASK2, address 0x400F 008C) bit description

Bit	Symbol	Value	Description	Reset value	Access
12:0	MSK28_16		Identifier mask [28:16]	0xFFF	R/W
		0	Match. The corresponding bit in the identifier of the message cannot inhibit the match in the acceptance filtering.		
		1	Mask. The corresponding identifier bit is used for acceptance filtering.		
13	-		Reserved	1	-
14	MDIR		Mask message direction	1	R/W
		0	Without DIR bit. The message direction bit (DIR) has no effect on acceptance filtering.		
		1	With DIR bit. The message direction bit (DIR) is used for acceptance filtering.		
15	MXTD		Mask extend identifier	1	R/W
		0	Without XTD. The extended identifier bit (XTD) has no effect on acceptance filtering.		
		1	With XTD. The extended identifier bit (XTD) is used for acceptance filtering.		
31:16	-	-	Reserved	0	-

27.6.2.4.3 CAN message interface arbitration 1 registers

Table 408. CAN message interface arbitration 1 registers (IF1_ARB1, address 0x400F 0030 and IF2_ARB1, address 0x400F 0090) bit description

Bit	Symbol	Description	Reset value	Access
15:0	ID15_0	Message identifier [15:0] 29-bit identifier (extended frame) 11-bit identifier (standard frame). These bits are not used for 11-bit identifiers.	0x00	R/W
31:16	-	Reserved	0	-

[1]

27.6.2.4.4 CAN message interface arbitration 2 registers

Table 409. CAN message interface arbitration 2 registers (IF1_ARB2, address 0x400F 0034 and IF2_ARB2, address 0x400F 0094) bit description

Bit	Symbol	Value	Description	Reset value	Access
12:0	ID28_16 ID28_18		Message identifier 29-bit identifier (extended frame) 11-bit identifier (standard frame). ID[17:16] are not used for 11-bit identifiers.	0x00	R/W
13	DIR		Message direction	0x00	R/W
		0	Receive. On TXRQST, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.		
		1	Transmit. On TXRQST, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TXRQST bit of this Message Object is set (if RMTEN = one).		
14	XTD		Extend identifier	0x00	R/W
		0	Standard. The 11-bit standard identifier will be used for this message object.		
		1	Extended. The 29-bit extended identifier will be used for this message object.		
15	MSGVAL		Message valid **Remark:** The CPU must reset the MSGVAL bit of all unused Messages Objects during the initialization before it resets bit INIT in the CAN Control Register. This bit must also be reset before the identifier ID28:0, the control bits XTD, DIR, or the Data Length Code DLC3:0 are modified, or if the Messages Object is no longer required.	0	R/W
		0	Invalid. The message object is ignored by the message handler.		
		1	Valid. The message object is configured and should be considered by the message handler.		
31:16	-	-	Reserved	0	-

[1]

27.6.2.4.5 CAN message interface message control registers

Table 410. CAN message interface message control registers (IF1_MCTRL, address 0x400F 0038 and IF2_MCTRL, address 0x400F 0098) bit description

Bit	Symbol	Value	Description	Reset value	Access
3:0	DLC3_0		Data length code 3:0 **Remark:** The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message. 0000 - 1000 = Data frame has 0 - 8 data bytes. 1001 - 1111 = Data frame has 8 data bytes.	0000	R/W
6:4	-		Reserved	-	-
7	EOB		End of buffer	0	R/W
		0	Not end of buffer. Message object belongs to a FIFO buffer and is not the last message object of that FIFO buffer.		
		1	End of buffer. Single message object or last message object of a FIFO buffer.		
8	TXRQST		Transmit request	0	R/W
		0	Not waiting. This message object is not waiting for transmission.		
		1	Waiting. The transmission of this message object is requested and is not yet done		
9	RMTEN		Remote enable	0	R/W
		0	TXRQST unchanged. At the reception of a remote frame, TXRQST is left unchanged.		
		1	TXRQST set. At the reception of a remote frame, TXRQST is set.		
10	RXIE		Receive interrupt enable	0	R/W
		0	INTPND unchanged. INTPND will be left unchanged after successful reception of a frame.		
		1	INTPND set. INTPND will be set after successful reception of a frame.		
11	TXIE		Transmit interrupt enable	0	R/W
		0	INTPND unchanged. The INTPND bit will be left unchanged after a successful transmission of a frame.		
		1	INTPND set. INTPND will be set after a successful transmission of a frame.		
12	UMASK		Use acceptance mask **Remark:** If UMASK is set to 1, the message object's mask bits have to be programmed during initialization of the message object before MAGVAL is set to 1.	0	R/W
		0	Ignore. Mask ignored.		
		1	Use. Use mask (MSK[28:0], MXTD, and MDIR) for acceptance filtering.		
13	INTPND		Interrupt pending	0	R/W
		0	Not pending. This message object is not the source of an interrupt.		
		1	Pending. This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.		

Table 410. CAN message interface message control registers (IF1_MCTRL, address 0x400F 0038 and IF2_MCTRL, address 0x400F 0098) bit description *…continued*

Bit	Symbol	Value	Description	Reset value	Access
14	MSGLST		Message lost (only valid for message objects in the direction receive).	0	R/W
		0	Not lost. No message lost since this bit was reset last by the CPU.		
		1	Lost. The Message Handler stored a new message into this object when NEWDAT was still set, the CPU has lost a message.		
15	NEWDAT		New data	0	R/W
		0	No new data. No new data has been written into the data portion of this message object by the message handler since this flag was cleared last by the CPU.		
		1	New data. The message handler or the CPU has written new data into the data portion of this message object.		
31:16	-	-	Reserved	0	-

27.6.2.4.6 CAN message interface data A1 registers

In a CAN Data Frame, DATA0 is the first, DATA7 (in CAN_IF1B2 AND CAN_IF2B2) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

Remark: Byte DATA0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte DATA7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

Table 411. CAN message interface data A1 registers (IF1_DA1, address 0x400F 003C and IF2_DA1, address 0x400F 009C) bit description

Bit	Symbol	Description	Reset value	Access
7:0	DATA0	Data byte 0	0x00	R/W
15:8	DATA1	Data byte 1	0x00	R/W
31:16	-	Reserved	-	-

27.6.2.4.7 CAN message interface data A2 registers

Table 412. CAN message interface data A2 registers (IF1_DA2, address 0x400F 0040 and IF2_DA2, address 0x400F 00A0) bit description

Bit	Symbol	Description	Reset value	Access
7:0	DATA2	Data byte 2	0x00	R/W
15:8	DATA3	Data byte 3	0x00	R/W
31:16	-	Reserved	-	-

27.6.2.4.8 CAN message interface data B1 registers

Table 413. CAN message interface data B1 registers (IF1_DB1, address 0x400F 0044 and IF2_DB1, address 0x400F 00A4) bit description

Bit	Symbol	Description	Reset value	Access
7:0	DATA4	Data byte 4	0x00	R/W
15:8	DATA5	Data byte 5	0x00	R/W
31:16	-	Reserved	-	-

27.6.2.4.9 CAN message interface data B2 registers

Table 414. CAN message interface data B2 registers (IF1_DB2, address 0x400F 0048 and IF2_DB2, address 0x400F 00A8) bit description

Bit	Symbol	Description	Reset value	Access
7:0	DATA6	Data byte 6	0x00	R/W
15:8	DATA7	Data byte 7	0x00	R/W
31:16	-	Reserved	-	-

27.6.3 Message handler registers

All Message Handler registers are read-only. Their contents (TXRQST, NEWDAT, INTPND, and MSGVAL bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM.

27.6.3.1 CAN transmission request 1 register

This register contains the TXRQST bits of message objects 1 to 16. By reading out the TXRQST bits, the CPU can check for which Message Object a Transmission Request is pending. The TXRQST bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

Table 415. CAN transmission request 1 register (TXREQ1, address 0x400F 0100) bit description

Bit	Symbol	Description	Reset value	Access
15:0	TXRQST16_1	Transmission request bit of message objects 16 to 1. 0 = This message object is not waiting for transmission. 1 = The transmission of this message object is requested and not yet done.	0x00	R
31:16	-	Reserved	-	-

27.6.3.2 CAN transmission request 2 register

This register contains the TXRQST bits of message objects 32 to 17. By reading out the TXRQST bits, the CPU can check for which Message Object a Transmission Request is pending. The TXRQST bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

Table 416. CAN transmission request 2 register (TXREQ2, address 0x400F 0104) bit description

Bit	Symbol	Description	Reset value	Access
15:0	TXRQST32_17	Transmission request bit of message objects 32 to 17. 0 = This message object is not waiting for transmission. 1 = The transmission of this message object is requested and not yet done.	0x00	R
31:16	-	Reserved	-	-

27.6.3.3 CAN new data 1 register

This register contains the NEWDAT bits of message objects 16 to 1. By reading out the NEWDAT bits, the CPU can check for which Message Object the data portion was updated. The NEWDAT bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Table 417. CAN new data 1 register (ND1, address 0x400F 0120) bit description

Bit	Symbol	Description	Reset value	Access
15:0	NEWDAT16_1	New data bits of message objects 16 to 1. 0 = No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. 1 = The Message Handler or the CPU has written new data into the data portion of this Message Object.	0x00	R
31:16	-	Reserved	-	-

27.6.3.4 CAN new data 2 register

This register contains the NEWDAT bits of message objects 32 to 17. By reading out the NEWDAT bits, the CPU can check for which Message Object the data portion was updated. The NEWDAT bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Table 418. CAN new data 2 register (ND2, address 0x400F 0124) bit description

Bit	Symbol	Description	Reset value	Access
15:0	NEWDAT32_17	New data bits of message objects 32 to 17. 0 = No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. 1 = The Message Handler or the CPU has written new data into the data portion of this Message Object.	0x00	R
31:16	-	Reserved	-	-

27.6.3.5 CAN interrupt pending 1 register

This register contains the INTPND bits of message objects 16 to 1. By reading out the INTPND bits, the CPU can check for which Message Object an interrupt is pending. The INTPND bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of INTPND in the Interrupt Register.

Table 419. CAN interrupt pending 1 register (IR1, address 0x400F 0140) bit description

Bit	Symbol	Description	Reset value	Access
15:0	INTPND16_1	Interrupt pending bits of message objects 16 to 1. 0 = This message object is ignored by the message handler. 1 = This message object is the source of an interrupt.	0x00	R
31:16	-	Reserved	-	-

27.6.3.6 CAN interrupt pending 2 register

This register contains the INTPND bits of message objects 32 to 17. By reading out the INTPND bits, the CPU can check for which Message Object an interrupt is pending. The INTPND bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of INTPND in the Interrupt Register.

Table 420. CAN interrupt pending 2 register (IR2, addresses 0x400F 0144) bit description

Bit	Symbol	Description	Reset value	Access
15:0	INTPND32_17	Interrupt pending bits of message objects 32 to 17. 0 = This message object is ignored by the message handler. 1 = This message object is the source of an interrupt.	0x00	R
31:16	-	Reserved	-	-

27.6.3.7 CAN message valid 1 register

This register contains the MSGVAL bits of message objects 16 to 1. By reading out the MSGVAL bits, the CPU can check which Message Object is valid. The MSGVAL bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers.

Table 421. CAN message valid 1 register (MSGV1, addresses 0x400F 0160) bit description

Bit	Symbol	Description	Reset value	Access
15:0	MSGVAL16_1	Message valid bits of message objects 16 to 1. 0 = This message object is ignored by the message handler. 1 = This message object is configured and should be considered by the message handler.	0x00	R
31:16	-	Reserved	-	-

27.6.3.8 CAN message valid 2 register

This register contains the MSGVAL bits of message objects 32 to 17. By reading out the MSGVAL bits, the CPU can check which Message Object is valid. The MSGVAL bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers.

Table 422. CAN message valid 2 register (MSGV2, address 0x400F 0164) bit description

Bit	Symbol	Description	Access	Reset value
15:0	MSGVAL32_17	Message valid bits of message objects 32 to 17. 0 = This message object is ignored by the message handler. 1 = This message object is configured and should be considered by the message handler.	R	0x00
31:16	-	Reserved	-	-

27.6.4 CAN timing register

27.6.4.1 CAN clock divider register

This register determines the CAN clock signal. The CAN_CLK is derived from the peripheral clock PCLK divided by the values in this register. Also see Section 27.3 for details on how the C_CAN clock is connected.

Table 423. CAN clock divider register (CLKDIV, address 0x400F 0180) bit description

Bit	Symbol	Description	Reset value	Access
3:0	CLKDIVVAL	Clock divider value. CAN_CLK = PCLK/(CLKDIVVAL +1) 0000: CAN_CLK = PCLK divided by 1. 0001: CAN_CLK = PCLK divided by 2. 0010: CAN_CLK = PCLK divided by 3 0011: CAN_CLK = PCLK divided by 4. ... 1111: CAN_CLK = PCLK divided by 16.	1	R/W
31:4	-	reserved	-	-

27.7 Functional description

27.7.1 C_CAN controller state after reset

After a hardware reset, the registers hold the values described in Table 393. Additionally, the busoff state is reset and the output CAN_TXD is set to recessive (HIGH). The value 0x0001 (INIT = ‘1’) in the CAN Control Register enables the software initialization. The CAN controller does not communicate with the CAN bus until the CPU resets INIT to ‘0’.

The data stored in the message RAM is not affected by a hardware reset. After power-on, the contents of the message RAM is undefined.

27.7.2 C_CAN operating modes

27.7.2.1 Software initialization

The software initialization is started by setting the bit INIT in the CAN Control Register, either by software or by a hardware reset, or by entering the busoff state.

During software initialization (INIT bit is set), the following conditions are present:

- All message transfer from and to the CAN bus is stopped.
- The status of the CAN output CAN_TXD is recessive (HIGH).
- The EC counters are unchanged.
- The configuration registers are unchanged.
- Access to the bit timing register and the BRP extension register is enabled if the CCE bit in the CAN control register is also set.

To initialize the CAN controller, software has to set up the bit timing register and each message object. If a message object is not needed, it is sufficient to set its MSGVAL bit to not valid. Otherwise, the whole message object has to be initialized.

Resetting the INIT bit finishes the software initialization. Afterwards the Bit Stream Processor BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle) before it can take part in bus activities and starts the message transfer.

Remark: The initialization of the Message Objects is independent of INIT and also can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid during software initialization before the BSP starts the message transfer. To change the configuration of a Message Object during normal operation, the CPU has to start by setting the MSGVAL bit to not valid. When the configuration is completed, MSAGVALis set to valid again.

27.7.2.2 CAN message transfer

Once the CAN controller is initialized and INIT is reset to zero, the CAN core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler’s acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to “don’t care” may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers. The Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then TXRQUT bit with NEWDAT bit are set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, and they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

27.7.2.3 Disabled Automatic Retransmission (DAR)

According to the *CAN Specification (ISO11898, 6.3.3 Recovery Management)*, the CAN controller provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, the automatic retransmission on lost arbitration or error is enabled. It can be disabled to enable the CAN controller to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment.

The Disable Automatic Retransmission mode is enabled by programming bit DAR in the CAN Control Register to one. In this operation mode the programmer has to consider the different behavior of bits TXRQST and NEWDAT in the Control Registers of the Message Buffers:

- When a transmission starts, bit TXRQST of the respective Message Buffer is reset while bit NEWDAT remains set.
- When the transmission completed successfully, bit NEWDAT is reset.
- When a transmission failed (lost arbitration or error), bit NEWDAT remains set. To restart the transmission, the CPU has to set TXRQST back to one.

27.7.2.4 Test modes

The Test mode is entered by setting bit TEST in the CAN Control Register to one. In Test mode the bits TX1, TX0, LBACK, SILENT, and BASIC in the Test Register are writable. Bit RX monitors the state of pins RD0,1 and therefore is only readable. All Test register functions are disabled when bit TEST is reset to zero.

27.7.2.4.1 Silent mode

The CAN core can be set in Silent mode by programming the Test register bit SILENT to one.

In Silent Mode, the CAN controller is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus, and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).

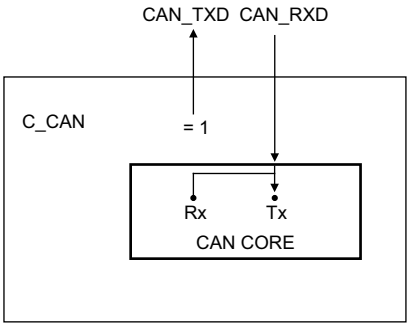


Fig 78. CAN core in Silent mode

27.7.2.4.2 Loop-back mode

The CAN Core can be set in Loop-back mode by programming the Test Register bit LBACK to one. In Loop-back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer.

This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop-back mode. In this mode the CAN core performs an internal feedback from its CAN_TXD output to its CAN_RXD input. The actual value of the CAN_RXD input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the CAN_TXD pin.

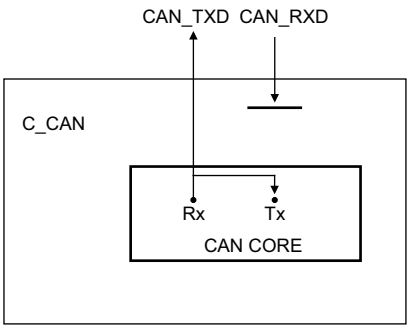


Fig 79. CAN core in Loop-back mode

27.7.2.4.3 Loop-back mode combined with Silent mode

It is also possible to combine Loop-back mode and Silent mode by programming bits LBACK and SILENT to one at the same time. This mode can be used for a “Hot Selftest”, meaning the C_CAN can be tested without affecting a running CAN system connected to the pins CAN_TXD and CAN_RXD. In this mode the CAN_RXD pin is disconnected from the CAN Core and the CAN_TXD pin is held recessive.

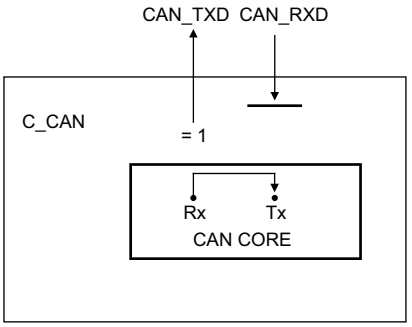


Fig 80. CAN core in Loop-back mode combined with Silent mode

27.7.2.4.4 Basic mode

The CAN Core can be set in Basic mode by programming the Test Register bit BASIC to one. In this mode the CAN controller runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the BUSY bit of the IF1 Command Request Register to '1'. The IF1 Registers are locked while the BUSY bit is set. The BUSY bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the BUSY bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the BUSY bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the BUSY bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the BUSY bit of the IF2 Command Request Register to '1', the contents of the shift register is stored into the IF2 Registers.

In Basic mode the evaluation of all Message Object related control and status bits and of the control bits of the IFx Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The NEWDAT and MSGLST bits of the IF2 Message Control Register retain their function, DLC3-0 will show the received DLC, the other control bits will be read as '0'.

In Basic mode the ready output CAN_WAIT_B is disabled (always '1')

27.7.2.4.5 Software control of pin CAN_TXD

Four output functions are available for the CAN transmit pin CAN_TXD:

1. serial data output (default).

2. drives CAN sample point signal to monitor the CAN controller's timing.
3. drives recessive constant value.
4. drives dominant constant value.

The last two functions, combined with the readable CAN receive pin CAN_RXD, can be used to check the CAN bus' physical layer.

The output mode of pin CAN_TXD is selected by programming the Test Register bits TX1 and TX0 as described Section 27.6.1.6.

Remark: The three test functions for pin CAN_TXD interfere with all CAN protocol functions. The CAN_TXD pin must be left in its default function when CAN message transfer or any of the test modes Loo-back mode, Silent mode, or Basic mode are selected.

27.7.3 CAN message handler

The Message handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFx Registers, see Figure 81.

The message handler controls the following functions:

- Data Transfer between IFx Registers and the Message RAM
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of TXRQST flags
- Handling of interrupts

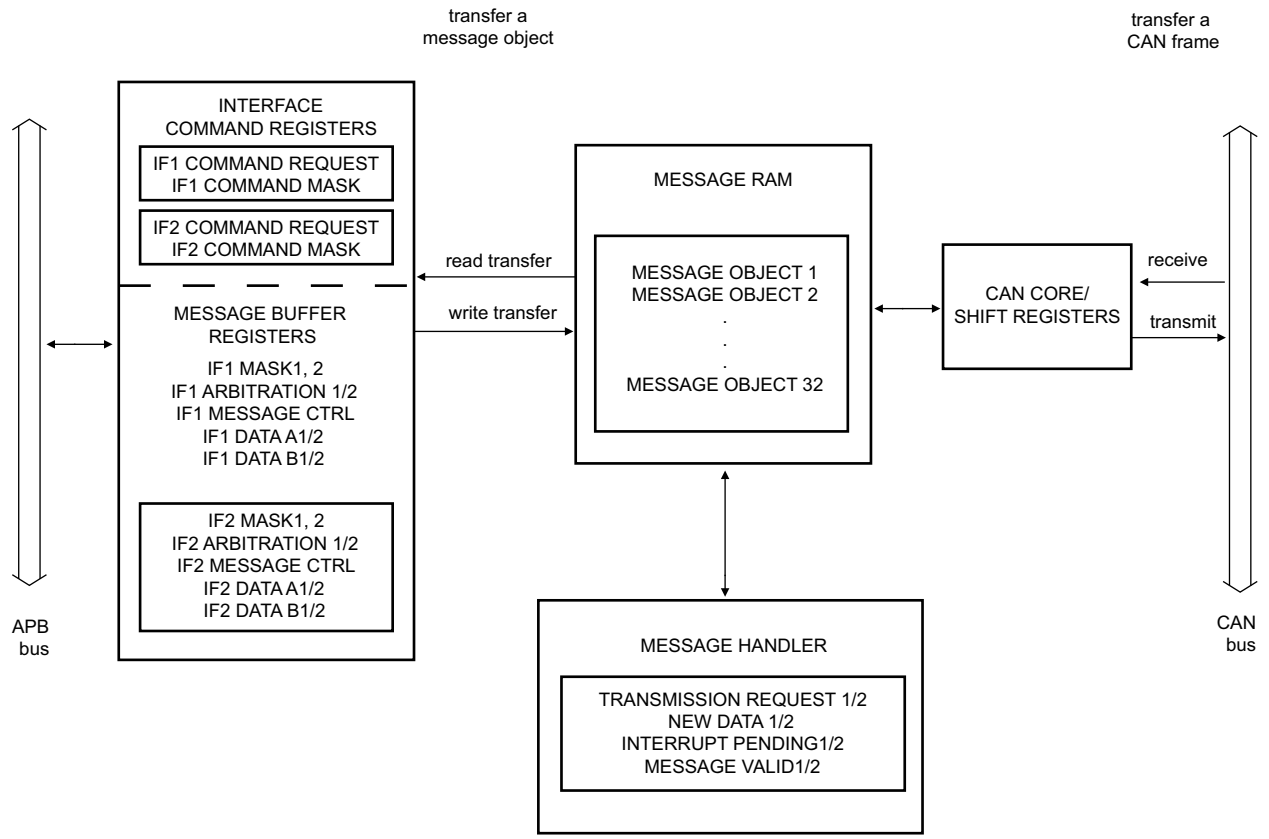


Fig 81. Block diagram of a message object transfer

27.7.3.1 Management of message objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits MSGVAL, NEWDAT, INTPND, and TXRQST) is not be affected by resetting the chip. All the Message Objects must be initialized by the CPU or they must be set to not valid (MSGVAL = '0').The bit timing must be configured before the CPU clears the INIT bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFx Command Request Register, the IFx Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the INIT bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN core and the Message Handler State Machine control the CAN controller's internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, and messages with pending transmission request are loaded into the CAN core's shift register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFx Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

27.7.3.2 Data Transfer between IFx Registers and the Message RAM

When the CPU initiates a data transfer between the IFx Registers and Message RAM, the Message Handler sets the BUSY bit in the respective Command Register to ‘1’. After the transfer has completed, the BUSY bit is set back to ‘0’.

The Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object. Software must always write a complete Message Object into the Message RAM. Therefore the data transfer from the IFx Registers to the Message RAM requires a read-modify-write cycle:

1. Read the parts of the message object that are not to be changed from the message RAM using the command mask register.
   - After the partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.
2. Write the complete contents of the message buffer registers into the message object.
   - After the partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will set to the actual contents of the selected Message Object.

27.7.3.3 Transmission of messages between the shift registers in the CAN core and the Message buffer

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFx Registers and Message RAM, the MSGVAL bits in the Message Valid Register TXRQST bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object’s NEWDAT bit is reset.

After a successful transmission and if no new data was written to the Message Object (NEWDAT = ‘0’) since the start of the transmission, the TXRQST bit will be reset. If TXIE is set, INTPND will be set after a successful transmission. If the CAN controller has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

27.7.3.4 Acceptance filtering of received messages

When the arbitration and control field (Identifier + IDE + RTR + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler state machine starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including MSGVAL, UMASK, NEWDAT, and EOB) of Message Object 1 are loaded into the Acceptance Filtering unit and compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler state machine proceeds depending on the type of frame (Data Frame or Remote Frame) received.

27.7.3.4.1 Reception of a data frame

The Message Handler state machine stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. The data bytes, all arbitration bits, and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NEWDAT bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU/software should reset NEWDAT when it reads the Message Object. If at the time of the reception the NEWDAT bit was already set, MSGLST is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RxIE bit is set, the INTPND bit is also set, causing the Interrupt Register to point to this Message Object.

The TXRQST bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

27.7.3.4.2 Reception of a remote frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1. DIR = ‘1’ (direction = transmit), RMTEN = ‘1’, UMASK = ‘1’ or’0’

   On the reception of a matching Remote Frame, the TXRQST bit of this Message Object is set. The rest of the Message Object remains unchanged.

2. DIR = ‘1’ (direction = transmit), RMTEN = ‘0’, UMASK = ’0’

   On the reception of a matching Remote Frame, the TXRQST bit of this Message Object remains unchanged; the Remote Frame is ignored.

3. DIR = ‘1’ (direction = transmit), RMTEN = ‘0’, UMASK = ’1’

   On the reception of a matching Remote Frame, the TXRQST bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM, and the NEWDAT bit of this Message Object is set. The data field of the Message Object remains unchanged; the Remote Frame is treated similar to a received Data Frame.

27.7.3.5 Receive/transmit priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced due to the priority of the corresponding Message Object.

27.7.3.6 Configuration of a transmit object

Table 424 shows how a transmit object should be initialized by software (see also Table 402):

Table 424. Initialization of a transmit object

MSGVAL	Arbitration bits	Data bits	Mask bits	EOB	DIR	NEWDAT
1	application dependent	application dependent	application dependent	1	1	0
MSGLST	**RXIE**	**TXIE**	**INTPND**	**RMTEN**	**TXRQST**	
0	0	application dependent	0	application dependent	0	

The Arbitration Registers (ID28:0 and XTD bit) are given by the application. They define the identifier and the type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28. In this case ID18, ID17 to ID0 can be disregarded.

If the TXIE bit is set, the INTPND bit will be set after a successful transmission of the Message Object.

If the RMTEN bit is set, a matching received Remote Frame will cause the TXRQST bit to be set, and the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (DLC3:0, Data0:7) are given by the application. TXRQST and RMTEN may not be set before the data is valid.

The Mask Registers (Msk28-0, UMASK, MXTD, and MDIR bits) may be used (UMASK=’1’) to allow groups of Remote Frames with similar identifiers to set the TXRQST bit. For details see Section 27.7.3.4.2. The DIR bit should not be masked.

27.7.3.7 Updating a transmit object

The CPU may update the data bytes of a Transmit Object any time via the IFx Interface registers. Neither MSGVAL nor TXRQST have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFx Data A Register or IFx Data B Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFx Data Register or the Message Object is transferred to the IFx Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 is written to the Command Mask Register. Then the number of the Message Object is written to the Command Request Register, concurrently updating the data bytes and setting TXRQST.

To prevent the reset of TXRQST at the end of a transmission that may already be in progress while the data is updated, NEWDAT has to be set together with TXRQST. For details see Section 27.7.3.3.

When NEWDAT is set together with TXRQST, NEWDAT will be reset as soon as the new transmission has started.

27.7.3.8 Configuration of a receive object

Table 425 shows how a receive object should be initialized by software (see also Table 402)

Table 425. Initialization of a receive object

MSGVAL	Arbitration bits	Data bits	Mask bits	EOB	DIR	NEWDAT
1	application dependent	application dependent	application dependent	1	0	0
MSGLST	**RXIE**	**TXIE**	**INTPND**	**RMTEN**	**TXRQST**	
0	application dependent	0	0	0	0	

The Arbitration Registers (ID28-0 and XTD bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 to ID18. ID17 to ID0 can then be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 to ID0 will be set to ‘0’.

If the RxIE bit is set, the INTPND bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLC[3:0] is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The Mask Registers (Msk[28:0], UMASK, MXTD, and MDIR bits) may be used (UMASK=’1’) to allow groups of Data Frames with similar identifiers to be accepted. For details see section Section 27.7.3.4.1. The DIR bit should not be masked in typical applications.

27.7.3.9 Handling of received messages

The CPU may read a received message any time via the IFx Interface registers. The data consistency is guaranteed by the Message Handler state machine.

To transfer the entire received message from message RAM into the message buffer, software must write first 0x007F to the Command Mask Register and then the number of the Message Object to the Command Request Register. Additionally, the bits NEWDAT and INTPND are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of NEWDAT shows whether a new message has been received since last time this Message Object was read. The actual value of MSGLST shows whether more than one message has been received since last time this Message Object was read. MSGLST will not be automatically reset.

Using a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TXRQST bit of a receive object will cause the transmission of a Remote Frame with the receive object’s identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TXRQST bit is automatically reset.

27.7.3.10 Configuration of a FIFO buffer

With the exception of the EOB bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see section Section 27.7.3.8.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EOB bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EOB bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

27.7.3.10.1 Reception of messages with FIFO buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the NEWDAT bit of this Message Object is set. By setting NEWDAT while EOB is zero the Message Object is locked for further write accesses by the Message Handler until the CPU has written the NEWDAT bit back to zero.

Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing NEWDAT to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

27.7.3.10.2 Reading from a FIFO buffer

When the CPU transfers the contents of Message Object to the IFx Message Buffer registers by writing its number to the IFx Command Request Register, bits NEWDAT and INTPND in the corresponding Command Mask Register should be reset to zero (TXRQST/NEWDAT = ‘1’ and ClrINTPND = ‘1’). The values of these bits in the Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number.

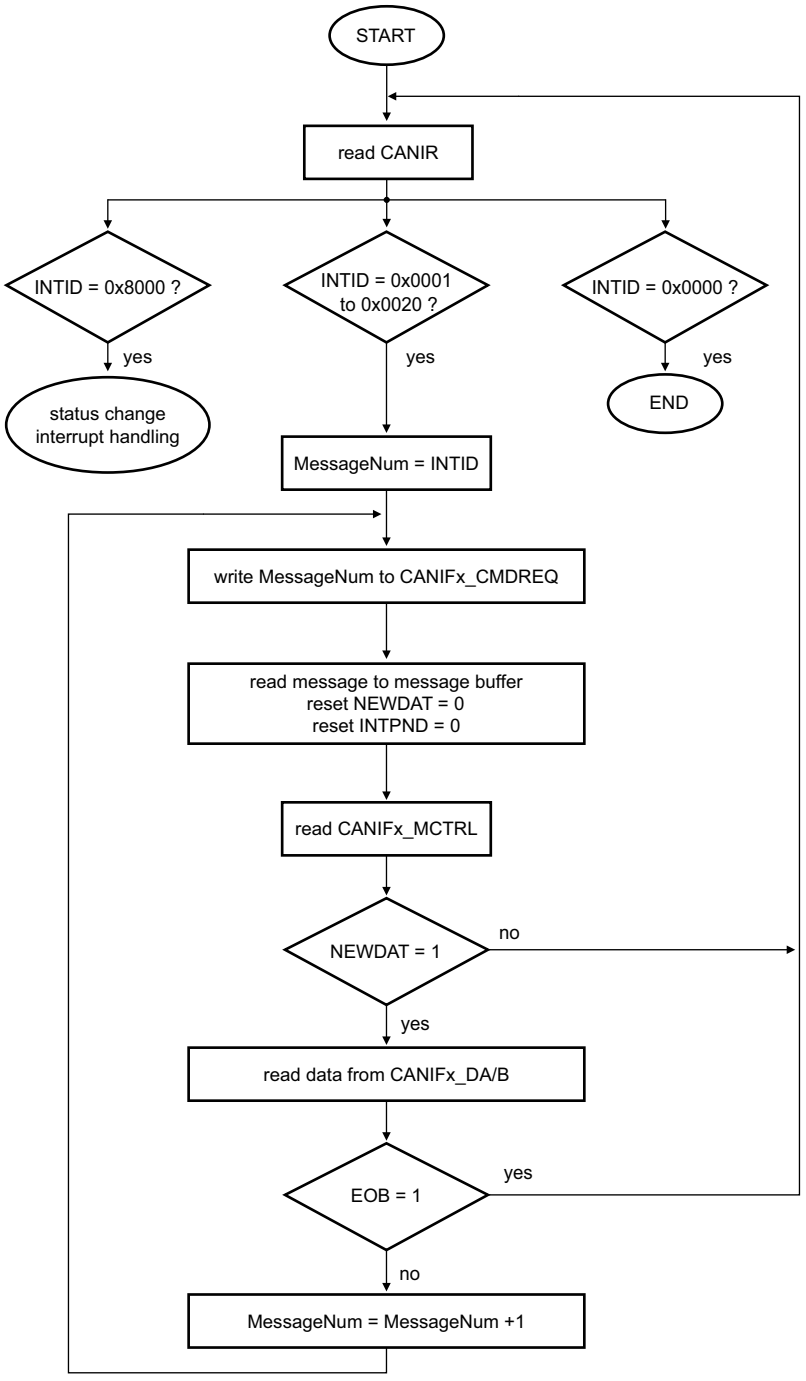


Fig 82. Reading a message from the FIFO buffer to the message buffer

27.7.4 Interrupt handling

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's INTPND bit. The Status Interrupt is cleared by reading the Status Register.

The interrupt identifier INTID in the Interrupt Register indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value zero. If the value of the Interrupt Register is different from zero, then there is an interrupt pending and, if IE is set, the interrupt line to the CPU, IRQ_B, is active. The interrupt line remains active until the Interrupt Register is back to value zero (the cause of the interrupt is reset) or until IE is reset.

The value 0x8000 indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the status bits RXOK, TXOK and LEC, but a write access of the CPU to the Status Register can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects where INTID points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Register may cause an interrupt (bits EIE and SIE in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit IE in the CAN Control Register). The Interrupt Register will be updated even when IE is reset.

The CPU has two possibilities to follow the source of a message interrupt:

- Software can follow the INTID in the Interrupt Register.
- Software can poll the interrupt pending register, see Section 27.6.3.5.

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's INTPND at the same time (bit ClrINTPND in the Command Mask Register). When INTPND is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

27.7.5 Bit timing

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly. In many cases, the CAN bit synchronization will amend a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated. In the case of arbitration however, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive.

The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and of the CAN nodes' interaction on the CAN bus.

27.7.5.1 Bit time and bit rate

CAN supports bit rates in the range of lower than 1 kBit/s up to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (f_{osc}) may be different.

The frequencies of these oscillators are not absolutely stable, as small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range (df), the CAN nodes are able to compensate for the different bit rates by re-synchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (Figure 83). The Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1, and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see Table 426). The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f and the Baud Rate Prescaler (BRP): t_q = BRP / f_{sys}. The C_CAN's system clock f_{sys} is the frequency of the system clock (see Section 27.3).

The Synchronization Segment Sync_Seg is the part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-)Synchronization Jump Width (SJW) defines how far a re-synchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Table 426 describes the minimum programmable ranges required by the CAN protocol. Bit time parameters are programmed through the CANBT register, Table 397. For details on bit timing and examples, see the *C_CAN user's manual, revision 1.2*.

Table 426. Parameters of the C_CAN bit time

Parameter	Range	Function
BRP	(1...32)	Defines the length of the time quantum t_q.
SYNC_SEG	$1t_q$	Synchronization segment. Fixed length. Synchronization of bus input to system clock.
PROP_SEG	(1...8) × t_q	Propagation time segment. Compensates for physical delay times. This parameter is determined by the system delay times in the C_CAN network.
TSEG1	(1...8) × t_q	Phase buffer segment 1. May be lengthened temporarily by synchronization.
TSEG2	(1...8) × t_q	Phase buffer segment 2. May be shortened temporarily by synchronization.
SJW	(1...4) × t_q	(Re-) synchronization jump width. May not be longer than either phase buffer segment.

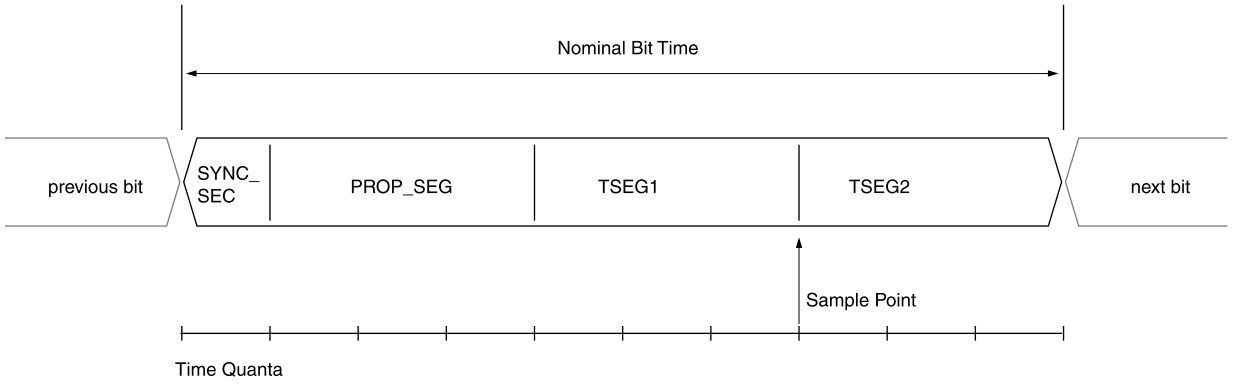


Fig 83. Bit timing

28.1 How to read this chapter

The ADC controller is available on all parts. For API support, see Chapter 39. The number of ADC channels available is dependent on the package size.

Table 427. ADC available analog inputs

Package	ADC0	ADC1	Channels ADC0/ADC1
LQFP48	ADC0_0 to ADC0_7; ADC0_10	ADC1_1 to ADC1_3; ADC1_6 to ADC1_9	9/7
LQFP64	ADC0_0 to ADC0_11	ADC1_0 to ADC1_11	12/12
LQFP100	ADC0_0 to ADC0_11	ADC1_0 to ADC1_11	12/12

28.2 Features

- 12-bit successive approximation analog to digital converter.
- Input multiplexing among 12 pins and 4 internal sources.
- Two configurable conversion sequences with independent triggers.
- Optional automatic high/low threshold comparison and “zero crossing” detection.
- Power-down mode and low-power operating mode.
- Measurement range V_{REFN} to V_{REFP} (typically 3 V; not to exceed V_{DDA} voltage level).
- 12-bit conversion rate of 2 MHz.
- Burst conversion mode for single or multiple inputs.
- Synchronous or asynchronous operation. Asynchronous operation maximizes flexibility in choosing the ADC clock frequency, Synchronous mode minimizes trigger latency and can eliminate uncertainty and jitter in response to a trigger.

28.3 Basic configuration

Configure the ADC as follows:

- Use the SYSAHBCLKCTRL0 register (Table 50) to enable the clock to the ADC0/1 register interface and the ADC0/1 clock.
- Clear the ADC0/1 peripheral reset using the PRESETCTRL0 register (Table 35).
- Each ADC block creates four interrupts which are connected to slot #31/35 (ADC0/1_SEQA), slot #32/36 (ADC0/1_SEQB), slot #33/37 (ADC0/1_THCMP), and slot #34/38 (ADC0/1_OVR) in the NVIC. The interrupts can also be configured as DMA triggers through the INPUT MUX (see Table 132) for each DMA channel and as inputs to the SCTs (Figure 31 “SCT to ADC connections”).
- Use the switch matrix to enable the ADC analog inputs.
- The power to the ADC0/1 blocks is controlled by the PDRUNCFG register in the SYSCON block. See Table 75.

- Calibration is required after every power-up or wake-up from Deep power-down mode. See Section 28.3.4 “Hardware self-calibration”.
- See Figure 31 “SCT to ADC connections” for connections between the ADC and the large SCTs.
- You can select from two options in the ADC CTRL register to clock ADC conversions:
  - Use the system clock to clock the ADC in synchronous mode. This option allows exact timing of triggers but requires a system clock of 50 MHz or higher to obtain the full ADC conversion speed.
  - Use the asynchronous ADC clock derived from the output of any of the three PLLs or the IRC. This option provides a clock which is independent of the system clock for ADC conversions. Some extra time might be needed to synchronize the ADC trigger inputs.
- Configure the ADC for the appropriate analog supply voltage using the TRM register (Table 447).

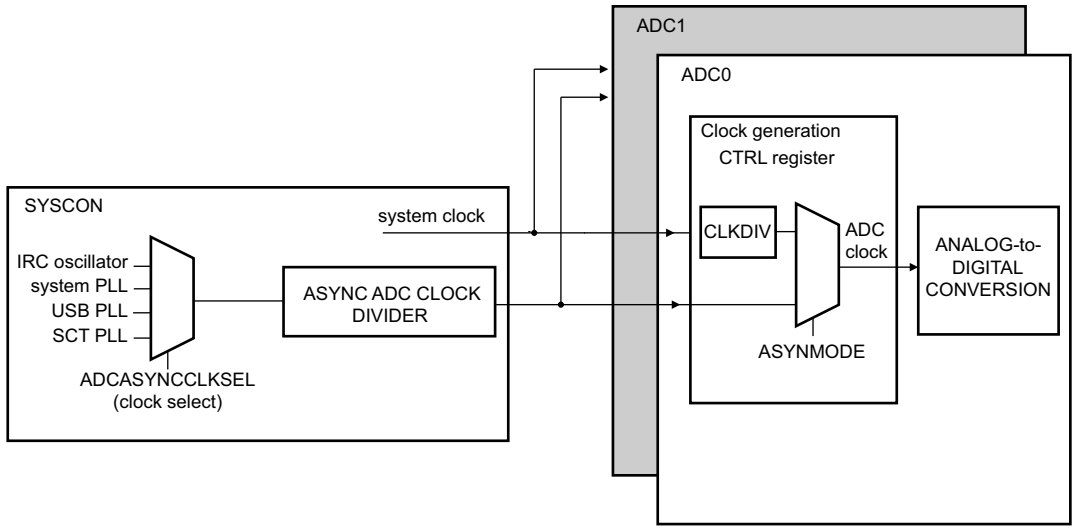


Fig 84. ADC clocking

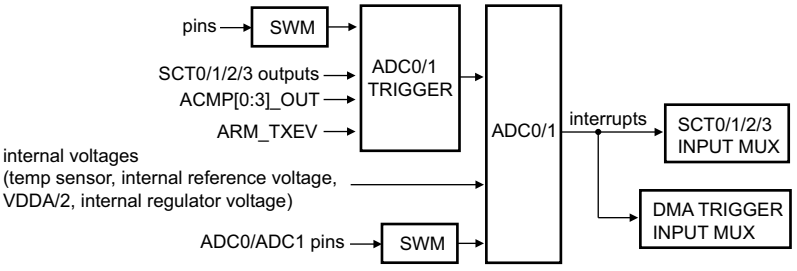


Fig 85. ADC connections

28.3.1 Perform a single ADC conversion triggered by software

Remark: When A/D conversions are triggered by software only and hardware triggers are not used in the conversion sequence, follow these steps to avoid spurious conversions:

1. Before changing the trigger set-up, disable the conversion sequence by setting the SEQ_ENA bit to 0 in the SEQA_CTRL register.
2. Set the trigger source to 0x0 using the TRIGGER bits in the SEQA_CTRL register. This is the default.
3. Disconnect the ADC0_PINTRIG0 from any pin by writing 0xFF to the PINASSIGN10 register bits ADC0_PIN_TRIG0_I. (For ADC1, write 0xFF to PINASSIGN11 register bits (ADC1_PIN_TRIG0_I). This is the default.
4. Set the TRIGPOL bit to 1 in the in the SEQA_CTRL register.

Once the sequence is enabled again, the ADC converts a sample whenever the START bit is written to. The TRIGPOL bit can be set in the same write that sets the SEQ_ENA and the START bits. Be careful not to modify the TRIGGER, TRIGPOL, and SEQ_ENA bits on subsequent writes to the START bit. See also Section 28.7.2.1 “Avoiding spurious hardware triggers”.

The ADC converts an analog input signal VIN on the ADC0_[11:0] and ADC1_[11:0] pins. The VREFP and VREFN pins provide a positive and negative reference voltage input. The result of the conversion is (4095 x VIN)/(VREFP - VREFN). The result of an input voltage below VREFN is 0, and the result of an input voltage above VREFP is 4095 (0xFFF).

To perform a single ADC conversion for ADC0 channel 1 using the analog signal on pin ADC0_1, follow these steps:

1. Enable the analog function on pin ADC0_1 through the switch matrix register PINENABLE0 on PIO0_7. See Table 123.
2. Configure the system clock to be 50 MHz and select a CLKDIV value of 0 for a sampling rate of 2 Msamples/s using the ADC CTRL register.
3. Select the synchronous mode in the CTRL register.
4. Select ADC channel 1 to perform the conversion by setting the CHANNELS bits to 0x2 in the SEQA_CTL register.
5. Set the TRIGPOL bit to 1 and the SEQA_ENA bit to 1 in the SEQA_CTRL register.
6. Set the START bit to 1 in the SEQA_CTRL register.
7. Read the RESULT bits in the DAT1 register for the conversion result.

28.3.2 Perform a sequence of conversions triggered by an external pin

The ADC can perform conversions on a sequence of selected channels. Each individual conversion of the sequence (single-step) or the entire sequence can be triggered by hardware. Hardware triggers are either a signal from an external pin or an internal signal. See Section 28.3.3.

To perform a single-step conversion on the first four channels of ADC0 triggered by rising edges on pin PIO1_0, follow these steps:

1. Enable the analog function on pin ADC0_0 to ADC0_3 through the switch matrix register PINENABLE0 on PIO0_8, PIO0_7, PIO0_6, and PIO0_5. See Table 123.

2. Connect the ADC0_PINTRIG1 function to pin PIO1_0 through the PINASSIGN11 register in the switch matrix. See Table 118.
3. Configure the system clock to be 50 MHz and select a CLKDIV value of 0 for a sampling rate of 2 Msamples/s using the ADC CTRL register.
4. Select the synchronous mode in the CTRL register.
5. Select ADC channels 0 to 3 to perform the conversion by setting the CHANNELS bits to 0xF in the SEQA_CTRL register.
6. Select ADC0_PINTRG1 by writing 0x1 the TRIGGER bits in the SEQA_CTRL register.
7. To generate one interrupt at the end of the entire sequence, set the MODE bit to 1 in the SEQA_CTRL register.
8. Select single-step mode by setting the SINGLESTEP bit in the SEQA_CTRL register to 1.
9. Enable the Sequence A by setting the SEQA_ENA bit.

   A conversion on ADC0 channel 0 will be triggered whenever the pin PIO1_0 goes from LOW to HIGH. The conversion on the next channel (channel 1) is triggered on the next rising edge of pin PIO1_0. The ADC0 interrupt is generated when the sequence has finished after four rising edges on pin PIO1_0.
10. Read the RESULT bits in the DAT0 to DAT3 registers for the conversion result.

28.3.3 ADC hardware trigger inputs

An analog-to-digital conversion can be initiated by a hardware trigger. You can select the trigger independently for each of the two conversion sequences in the ADC SEQA_CTRL or SEQB_CTRL registers by programming the hardware trigger input # into the TRIGGER bits.

Related registers:

- Table 435 “A/D Conversion Sequence A Control Register (SEQA_CTRL, addresses 0x4000 0008 (ADC0) and 0x4008 0008 (ADC1)) bit description”
- Table 436 “A/D Conversion Sequence B Control Register (SEQB_CTRL, 0x4000 000C (ADC0) and 0x4008 000C (ADC1)) bit description”

Table 428. ADC0 hardware trigger inputs

Input #	Source	Description
0	ADC0_PINTRIG0	Use the switch matrix to select a pin.
1	ADC0_PINTRIG1	Use the switch matrix to select a pin.
2	SCT0_OUT7	Internal; SCT output
3	SCT0_OUT9	Internal; SCT output
4	SCT1_OUT7	Internal; SCT output
5	SCT1_OUT9	Internal; SCT output
6	SCT2_OUT3	Internal; SCT output
7	SCT2_OUT4	Internal; SCT output
8	SCT3_OUT3	Internal; SCT output
9	SCT3_OUT4	Internal; SCT output
10	ACMP0_O	Internal; comparator output

Table 428. ADC0 hardware trigger inputs

Input #	Source	Description
11	ACMP1_O	Internal; comparator output
12	ACMP2_O	Internal; comparator output
13	ACMP3_O	Internal; comparator output
14	Reserved	-
15	ARM_TXEV	-

Table 429. ADC1 hardware trigger inputs

Input #	Source	Description
0	ADC1_PINTRIG0	Use the switch matrix to select a pin.
1	ADC1_PINTRIG1	Use the switch matrix to select a pin.
2	SCT0_OUT6	Internal; SCT output
3	SCT0_OUT9	Internal; SCT output
4	SCT1_OUT8	Internal; SCT output
5	SCT1_OUT9	Internal; SCT output
6	SCT2_OUT2	Internal; SCT output
7	SCT2_OUT5	Internal; SCT output
8	SCT3_OUT2	Internal; SCT output
9	SCT3_OUT5	Internal; SCT output
10	ACMP0_O	Internal; comparator output
11	ACMP1_O	Internal; comparator output
12	ACMP2_O	Internal; comparator output
13	ACMP3_O	Internal; comparator output
14	Reserved	-
15	ARM_TXEV	-

28.3.4 Hardware self-calibration

The A/D converter includes a built-in, hardware self-calibration mode. In order to achieve the specified ADC accuracy, the A/D converter must be recalibrated, at a minimum, following every chip reset before initiating normal ADC operation.

The calibration voltage level is VREFP - VREFN.

To calibrate the ADC use the ADC calibration API (see Table 563 “ADC API calls”) or follow these steps:

1. Save the current contents of the ADC CTRL register if different from default.
2. In a single write to the ADC CTRL register, do the following to start the calibration:
   - Set the calibration mode bit CALMODE.
   - Clear the ASYNC bit.
   - Write a divider value to the CLKDIV bit field that divides the system clock to yield an ADC clock of about 500 kHz.
   - Clear the LPWR bit.
3. Poll the CALMODE bit until it is cleared.

Before launching a new A/D conversion, restore the contents of the CTRL register or use the default values.

Remark: The API calibration routine saves any existing Control Register settings, then modifies those settings as required and sets the CAL_MODE bit to launch calibration. When the calibration completes, the API routine will restore the original user settings to the Control Register.

A calibration cycle requires approximately 290 µs to complete. Normal ADC conversions cannot be launched, and the ADC Control Register must not be written to while calibration is in progress. The calibration procedure does not use the CPU or memory, so other processes can be executed during calibration.

28.4 Pin description

The ADC can measure the voltage on any of the input signals on the analog input channel. Digital signals are disconnected from the ADC input pins when the ADC function is selected on that pin in the PINENABLE register.

The ADC analog inputs are connected to pins through the switch matrix which enables or disabled the analog function on a specific pin on the package.

Remark: If the ADC is used, signal levels on analog input pins must not be above the level of V_{DDA} at any time. Otherwise, ADC readings will be invalid. If the ADC is not used in an application, then the pins associated with ADC inputs can be configured as digital I/O pins and are 5 V tolerant.

The ADC pin triggers are movable (digital) functions. If you want to use external pin triggers for ADC conversions, assign the pin triggers to any pin on the package through the pin assign registers in the switch matrix. In addition to assigning the pin triggers to a pin, you must also select them in the conversion sequence registers for each ADC conversion sequence defined.

.The VREFP_ADC and VREFN pins provide a positive and negative reference voltage input. The result of the conversion is (4095 x input voltage VIN)/(VREFP_ADC - VREFN). The result of an input voltage below VREFN is 0, and the result of an input voltage above VREFP is 4095 (0xFFF). VREFP_ADC should be tied to VDD and VREFN should be tied to V_{SS} if the ADC and DAC are not used.

Analog Power and Ground should typically be the same voltages as V_{DD} and V_{SS}, but should be isolated to minimize noise and error. V_{DDA} should be tied to VDD and V_{SSA} should be tied to V_{SS} if the ADC and DAC are not used.

Table 430. ADC common supply and reference pins

Pin	Description
V_{DDA}	Analog supply voltage. VREFP_ADC must not exceed the voltage level on V_{DDA}. **Remark:** The supply voltage V_{DD} must be equal or lower than V_{DDA}.
V_{SSA}	Analog ground.
VREFP_ADC	Positive reference voltage. VREFP must > 2.4 V. For best performance, select VREFP = V_{DDA} and VREFN = V_{SSA}.
VREFN	Negative reference voltage.

Table 431. ADC0/1 pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
ADC0						
ADC0_0	AI	external to pin or internal	PIO0_8 or temp sensor, band gap voltage, power	PINENABLE0, INSEL	Table 123, Table 434	Analog input channel 0.
ADC0_1	AI	external to pin	PIO0_7	PINENABLE0	Table 123	Analog input channel 1.
ADC0_2	AI	external to pin	PIO0_6	PINENABLE0	Table 123	Analog input channel 2.
ADC0_3	AI	external to pin	PIO0_5	PINENABLE0	Table 123	Analog input channel 3.
ADC0_4	AI	external to pin	PIO0_4	PINENABLE0	Table 123	Analog input channel 4.
ADC0_5	AI	external to pin	PIO0_3	PINENABLE0	Table 123	Analog input channel 5.
ADC0_6	AI	external to pin	PIO0_2	PINENABLE0	Table 123	Analog input channel 6.
ADC0_7	AI	external to pin	PIO0_1	PINENABLE0	Table 123	Analog input channel 7.
ADC0_8	AI	external to pin	PIO1_0	PINENABLE0	Table 123	Analog input channel 8.
ADC0_9	AI	external to pin	PIO0_31	PINENABLE0	Table 123	Analog input channel 9.
ADC0_10	AI	external to pin	PIO0_0	PINENABLE0	Table 123	Analog input channel 10.
ADC0_11	AI	external to pin	PIO0_30	PINENABLE0	Table 123	Analog input channel 11.
ADC0_PINTRIG0	I	external to pin	any pin	PINASSIGN10	Table 117	ADC0 pin trigger input 0. Also select in ADC CTRL register as trigger.
ADC0_PINTRIG1	I	external to pin	any pin	PINASSIGN11	Table 118	ADC0 pin trigger input 1. Also select in ADC CTRL registers as trigger.
ADC1						
ADC1_0	AI	external to pin or internal	PIO1_1 or temp sensor, band gap voltage, power	PINENABLE0, INSEL	Table 123, Table 434	Analog input channel 0.
ADC1_1	AI	external to pin	PIO0_9	PINENABLE0	Table 123	Analog input channel 1.
ADC1_2	AI	external to pin	PIO0_10	PINENABLE0	Table 123	Analog input channel 2.
ADC1_3	AI	external to pin	PIO0_11	PINENABLE0	Table 123	Analog input channel 3.
ADC1_4	AI	external to pin	PIO1_2	PINENABLE0	Table 123	Analog input channel 4.
ADC1_5	AI	external to pin	PIO1_3	PINENABLE0	Table 123	Analog input channel 5.
ADC1_6	AI	external to pin	PIO0_13	PINENABLE0	Table 123	Analog input channel 6.
ADC1_7	AI	external to pin	PIO0_14	PINENABLE0	Table 123	Analog input channel 7.
ADC1_8	AI	external to pin	PIO0_15	PINENABLE0	Table 123	Analog input channel 8.
ADC1_9	AI	external to pin	PIO0_16	PINENABLE0	Table 123	Analog input channel 9.
ADC1_10	AI	external to pin	PIO1_4	PINENABLE0	Table 123	Analog input channel 10.

Table 431. ADC0/1 pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
ADC1_11	AI	external to pin	PIO1_5	PINENABLE0	Table 123	Analog input channel 11.
ADC1_PINTRIG0	I	external to pin	any pin	PINASSIGN11	Table 118	ADC1 pin trigger input 0. Also select in ADC CTRL register as trigger.
ADC1_PINTRIG1	I	external to pin	any pin	PINASSIGN11	Table 118	ADC1 pin trigger input 1. Also select in ADC CTRL registers as trigger.

28.5 General description

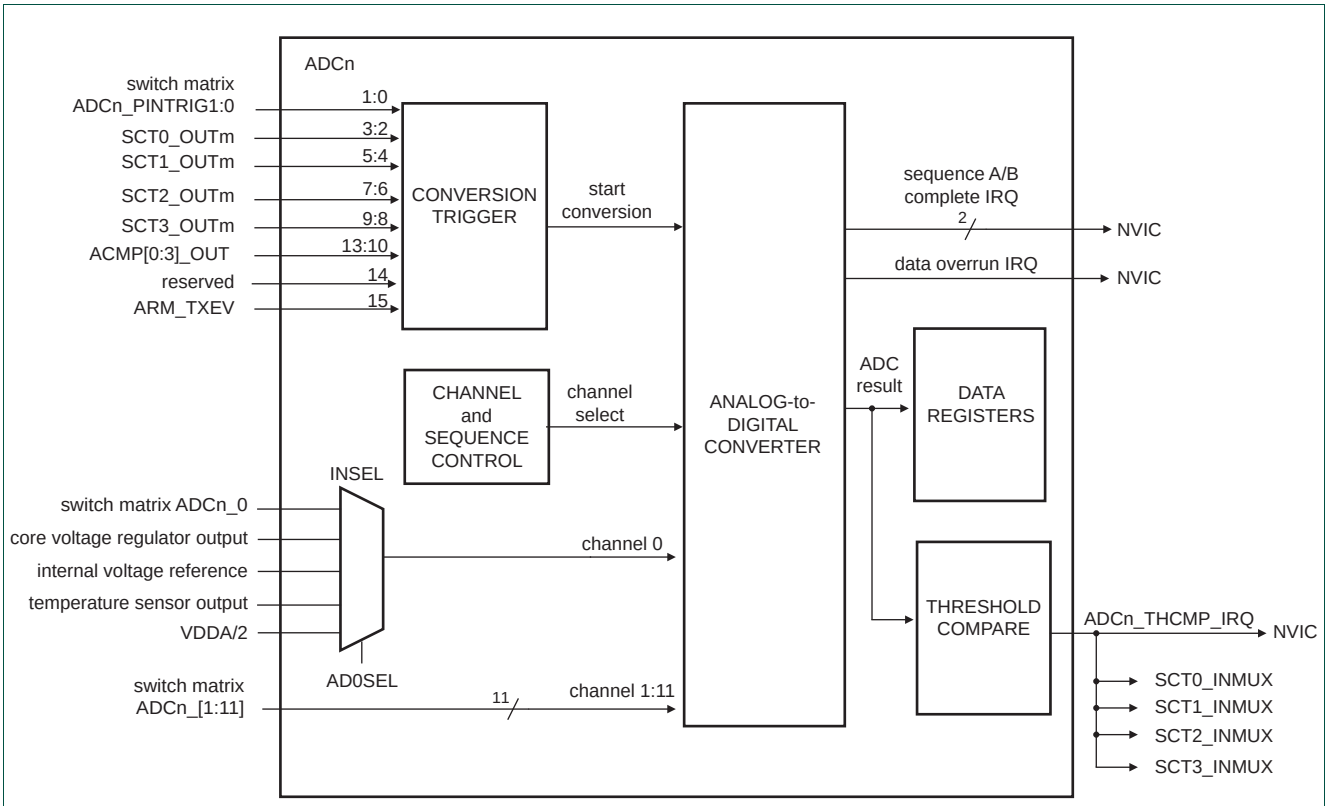


Fig 86. ADC block diagram

The ADC controller provides a great deal of flexibility in launching and controlling sequences of A/D conversions using the associated 12-bit, successive approximation A/D converter. A/D conversion sequences can be initiated under software control or in response to a selected hardware trigger. Each ADC supports 16 possible hardware triggers.

Once the triggers are set up (software and hardware triggers can be mixed), the ADC runs through the pre-defined conversion sequence, converting a sample whenever a trigger signal arrives, until the sequence is disabled.

The ADC controller uses the system clock as a bus clock. The system clock or the asynchronous ADC clock (see Figure 84) can be used to create the ADC clock which drives the successive approximation process:

- In the synchronous operating mode, this ADC clock is derived from the system clock. In this mode, a programmable divider is included to scale the system clock to the maximum ADC clock rate of 50 MHz (72 MHz in 10-bit mode).
- In the asynchronous mode, an independent clock source is used as the ADC clock source without any further divider in the ADC. The maximum ADC clock rate is 50 MHz (72 MHz in 10-bit mode) as well. **In this mode, the ADC clock frequency must not exceed ten times the system clock.**

A fully accurate conversion requires 25 ADC clocks.

28.6 Register description

The reset value reflects the data stored in used bits only. It does not include reserved bits content.

Table 432. Register overview : ADC (base address 0x4000 0000 (ADC0) and 0x4008 0000 (ADC1))

Name	Access	Address offset	Description	Reset value	Reference
CTRL	R/W	0x000	A/D Control Register. Contains the clock divide value, enable bits for each sequence and the A/D power-down bit.	0x0	Table 433
INSEL	R/W	0x004	A/D Input Select Register: Selects between external pin and internal source for channel 0 input.	0x0	Table 434
SEQA_CTRL	R/W	0x008	A/D Conversion Sequence-A control Register: Controls triggering and channel selection for conversion sequence-A. Also specifies interrupt mode for sequence-A.	0x0	Table 435
SEQB_CTRL	R/W	0x00C	A/D Conversion Sequence-B Control Register: Controls triggering and channel selection for conversion sequence-B. Also specifies interrupt mode for sequence-B.	0x0	Table 436
SEQA_GDAT	R/W	0x010	A/D Sequence-A Global Data Register. This register contains the result of the most recent A/D conversion performed under sequence-A	NA	Table 437
SEQB_GDAT	R/W	0x014	A/D Sequence-B Global Data Register. This register contains the result of the most recent A/D conversion performed under sequence-B	NA	Table 438
DAT0	RO	0x020	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0.	NA	Table 439
DAT1	RO	0x024	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	NA	Table 439
DAT2	RO	0x028	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	NA	Table 439

Table 432. Register overview : ADC (base address 0x4000 0000 (ADC0) and 0x4008 0000 (ADC1))

Name	Access	Address offset	Description	Reset value	Reference
DAT3	RO	0x02C	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	NA	Table 439
DAT4	RO	0x030	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	NA	Table 439
DAT5	RO	0x034	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	NA	Table 439
DAT6	RO	0x038	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	NA	Table 439
DAT7	RO	0x03C	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 439
DAT8	RO	0x040	A/D Channel 8 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 439
DAT9	RO	0x044	A/D Channel 9 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 439
DAT10	RO	0x048	A/D Channel 10 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 439
DAT11	RO	0x04C	A/D Channel 11 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	Table 439
THR0_LOW	R/W	0x050	A/D Low Compare Threshold Register 0 : Contains the lower threshold level for automatic threshold comparison for any channels linked to threshold pair 0.	0x0	Table 440
THR1_LOW	R/W	0x054	A/D Low Compare Threshold Register 1: Contains the lower threshold level for automatic threshold comparison for any channels linked to threshold pair 1.	0x0	Table 441
THR0_HIGH	R/W	0x058	A/D High Compare Threshold Register 0: Contains the upper threshold level for automatic threshold comparison for any channels linked to threshold pair 0.	0x0	Table 442
THR1_HIGH	R/W	0x05C	A/D High Compare Threshold Register 1: Contains the upper threshold level for automatic threshold comparison for any channels linked to threshold pair 1.	0x0	Table 443
CHAN_THRSEL	R/W	0x060	A/D Channel-Threshold Select Register. Specifies which set of threshold compare registers are to be used for each channel	0x0	Table 444

Table 432. Register overview : ADC (base address 0x4000 0000 (ADC0) and 0x4008 0000 (ADC1))

Name	Access	Address offset	Description	Reset value	Reference
INTEN	R/W	0x064	A/D Interrupt Enable Register. This register contains enable bits that enable the sequence-A, sequence-B, threshold compare and data overrun interrupts to be generated.	0x0	Table 445
FLAGS	R/W	0x068	A/D Flags Register. Contains the four interrupt/DMA trigger flags and the individual component overrun and threshold-compare flags. (The overrun bits replicate information stored in the result registers).	0x0	Table 446
TRM	R/W	0x06C	ADC trim register.	0x0000 0F00	Table 447

28.6.1 ADC Control Register

This register specifies the clock divider value to be used to generate the ADC clock **in synchronous mode** and general operating mode bits including a low power mode that allows the ADC to be turned off to save power when not in use.

Table 433. A/D Control Register (CTRL, addresses 0x4000 0000 (ADC0) and 0x4008 0000 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
7:0	CLKDIV		In synchronous mode only, the system clock is divided by this value plus one to produce the clock for the A/D converter, which should be less than or equal to 50 MHz (up to 72 MHz in 10-bit mode). Typically, software should program the smallest value in this field that yields this maximum clock rate or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable. **Remark:** This field is ignored in the asynchronous operating mode.	0
8	ASYNMODE		Select clock mode.	0
		0	Synchronous mode. The ADC clock is derived from the system clock based on the divide value selected in the CLKDIV field. The ADC clock will be started in a controlled fashion in response to a trigger to eliminate any uncertainty in the launching of an ADC conversion in response to any synchronous (on-chip) trigger. In Synchronous mode with the SYNCBYPASS bit set, sampling of the A/D input and start of conversion will initiate exactly 2 system clocks after the leading edge of a (synchronous) trigger pulse.	
		1	Asynchronous mode. The ADC clock is based on the output of the asynchronous ADC clock divider ADCASYNCCLKSEL in the SYSCON block. The frequency of this clock is limited to 50 MHz max (72 MHz in 10-bit mode). In addition, the ADC clock must never be faster than 10x the system clock.	
9	MODE10BIT		Select 10-bit conversion. In 10-bit mode higher conversion rates of up to 100 MHz are supported. A/D resolution is reduced to ten bits, but the clock rate (set via the CLKDIV field) can be increased up to 72 MHz to achieve a conversion rate of up to four million samples per second. The two LSBs of the result data are forced to zero.	0
		0	Disabled. The 10-bit/high-conversion rate mode is disabled.	
		1	Enabled. The 10-bit/high-conversion rate is enabled.	

Table 433. A/D Control Register (CTRL, addresses 0x4000 0000 (ADC0) and 0x4008 0000 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
10	LPWRMODE		Select low-power ADC mode. The analog circuitry is automatically powered-down when no conversions are taking place. When any (hardware or software) triggering event is detected, the analog circuitry is enabled. After the required start-up time, the requested conversion will be launched. Once the conversion completes, the analog-circuitry will again be powered-down provided no further conversions are pending. Using this mode can save an appreciable amount of current (approximately 2.5 mA) when conversions are required relatively infrequently. The penalty for using this mode is an approximately 15 ADC clock delay (30 clocks in 10-bit mode), based on the frequency specified in the CLKDIV field, from the time the trigger event occurs until sampling of the A/D input commences. **Remark:** This mode will NOT power-up the A/D when the ADC analog block is powered down in the system control block.	0
		0	Disabled. The low-power ADC mode is disabled. The analog circuitry remains activated even when no conversions are requested.	
		1	Enabled. The low-power ADC mode is enabled.	
29:11			Reserved, user software should not write ones to reserved bits.	0
30	CALMODE		Writing a 1 to this bit initiates a self-calibration cycle. This bit will be automatically cleared by hardware after the calibration cycle is complete. **Remark:** Other bits of this register may be written to concurrently with setting this bit, however once this bit has been set no further writes to this register are permitted until the full calibration cycle has ended.	0
31	-		Reserved.	0

28.6.2 ADC input Select Register

The ADC Input Select Register extends the scope of the ADC to measure certain internal signals as well as the external input using ADC channel 0. This registers controls the analog multiplexer that implements that selection.

If you always use one of the internal voltage sources or VDDA (not ADC0_0) for channel 0, you can disable the ADC0_0 function in the switch matrix and use the pin for a digital function.

Table 434: ADC Input Select Register (INSEL, addresses 0x4000 0004 (ADC0) and 0x4008 0004 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
3:0	CHAN0SEL		This field selects the input source for channel 0. All other values are reserved.	0
		0x0	ADCn_0 pin. Voltage on ADC channel 0 input.	
		0x1	Core voltage regulator output (1.2V to 1.8V). If the WRAPEN field is 0x2, the core voltage regulator output is also is output on the ADCn_0 pin.	
		0x2	Internal voltage reference. If the WRAPEN field is 0x2, the internal voltage reference is also is output on the ADCn_0 pin.	
		0x3	Temperature Sensor. If the WRAPEN field is 0x2, the temperature sensor voltage is also is output on the ADCn_0 pin.	
		0x4	VDDA/2. Divided analog supply voltage input.	
		0xF	No connection or load.	
29:4	-		Reserved.	
31:30	-		Reserved. Only write 0 to these bits.	0

28.6.3 A/D Conversion Sequence A Control Register

There are two, independent conversion sequences that can be configured, each consisting of a set of conversions on one or more channels. This control register specifies the channel selection and trigger conditions for the A sequence and contains bits to allow software to initiate that conversion sequence.

To avoid conversions on spurious triggers, only change the trigger configuration when the conversion sequence is disabled. A conversion can be triggered by software or hardware in the conversion sequence, but if conversions are triggered by software only, spurious hardware triggers must be prevented. See Section 28.3.1 “Perform a single ADC conversion triggered by software”.

Remark: Set the BURST and SEQU_ENA bits at the same time.

Table 435: A/D Conversion Sequence A Control Register (SEQA_CTRL, addresses 0x4000 0008 (ADC0) and 0x4008 0008 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
11:0	CHANNELS		Selects which one or more of the twelve channels will be sampled and converted when this sequence is launched. A 1 in any bit of this field will cause the corresponding channel to be included in the conversion sequence, where bit 0 corresponds to channel 0, bit 1 to channel 1 and so forth. When this conversion sequence is triggered, either by a hardware trigger or via software command, A/D conversions will be performed on each enabled channel, in sequence, beginning with the lowest-ordered channel. **Remark:** This field can ONLY be changed while the SEQA_ENA bit (bit 31) is LOW. It is allowed to change this field and set bit 31 in the same write.	0x00
15:12	TRIGGER		Selects which of the available hardware trigger sources will cause this conversion sequence to be initiated. Program the trigger input number in this field. **Remark:** In order to avoid generating a spurious trigger, it is recommended writing to this field only when the SEQA_ENA bit (bit 31) is low. It is safe to change this field and set bit 31 in the same write.	0x0
17:16	-		Reserved.	-
18	TRIGPOL		Select the polarity of the selected input trigger for this conversion sequence. **Remark:** In order to avoid generating a spurious trigger, it is recommended writing to this field only when the SEQA_ENA bit (bit 31) is low. It is safe to change this field and set bit 31 in the same write.	0
		0	Negative edge. A negative edge launches the conversion sequence on the selected trigger input.	
		1	Positive edge. A positive edge launches the conversion sequence on the selected trigger input.	
19	SYNCBYPASS		Setting this bit allows the hardware trigger input to bypass synchronization flip-flops stages and therefore shorten the time between the trigger input signal and the start of a conversion. There are slightly different criteria for whether or not this bit can be set depending on the clock operating mode: Synchronous mode: Synchronization may be bypassed (this bit may be set) if the selected trigger source is already synchronous with the main system clock (eg. coming from an on-chip, system-clock-based timer). Whether this bit is set or not, a trigger pulse must be maintained for at least one system clock period. Asynchronous mode: Synchronization may be bypassed (this bit may be set) if it is certain that the duration of a trigger input pulse will be at least one cycle of the ADC clock (regardless of whether the trigger comes from and on-chip or off-chip source). If this bit is NOT set, the trigger pulse must at least be maintained for one system clock period.	0
		0	Enable synchronization. The hardware trigger bypass is not enabled.	
		1	Bypass synchronization. The hardware trigger bypass is enabled.	
25:20	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	N/A
26	START		Writing a 1 to this field will launch one pass through this conversion sequence. The behavior will be identical to a sequence triggered by a hardware trigger. Do not write 1 to this bit if the BURST bit is set. **Remark:** This bit is only set to a 1 momentarily when written to launch a conversion sequence. It will consequently always read-back as a zero.	0

Table 435: A/D Conversion Sequence A Control Register (SEQA_CTRL, addresses 0x4000 0008 (ADC0) and 0x4008 0008 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
27	BURST		Writing a 1 to this bit will cause this conversion sequence to be continuously cycled through. Other sequence A triggers will be ignored while this bit is set. Repeated conversions can be halted by clearing this bit. The sequence currently in progress will be completed before conversions are terminated.	0
28	SINGLESTEP		When this bit is set, a hardware trigger or a write to the START bit will launch a single conversion on the next channel in the sequence instead of the default response of launching an entire sequence of conversions. Once all of the channels comprising a sequence have been converted, a subsequent trigger will repeat the sequence beginning with the first enabled channel. Interrupt generation will still occur either after each individual conversion or at the end of the entire sequence, depending on the state of the MODE bit.	0
29	LOWPRIO		Set priority for sequence A.	0
		0	Low priority. Any B trigger which occurs while an A conversion sequence is active will be ignored and lost.	
		1	High priority. Setting this bit to a 1 will permit any enabled B sequence trigger (including a B sequence software start) to immediately interrupt this sequence and launch a B sequence in it's place. The conversion currently in progress will be terminated. The A sequence that was interrupted will automatically resume after the B sequence completes. The channel whose conversion was terminated will be re-sampled and the conversion sequence will resume from that point.	
30	MODE		Indicates whether the primary method for retrieving conversion results for this sequence will be accomplished via reading the global data register (SEQA_GDAT) at the end of each conversion, or the individual channel result registers at the end of the entire sequence. Impacts when conversion-complete interrupt/DMA trigger for sequence-A will be generated and which overrun conditions contribute to an overrun interrupt as described below:	0
		0	End of conversion. The sequence A interrupt/DMA trigger will be set at the end of each individual A/D conversion performed under sequence A. This flag will mirror the DATAVALID bit in the SEQA_GDAT register. The OVERRUN bit in the SEQA_GDAT register will contribute to generation of an overrun interrupt/DMA trigger if enabled.	
		1	End of sequence. The sequence A interrupt/DMA trigger will be set when the entire set of sequence-A conversions completes. This flag will need to be explicitly cleared by software or by the DMA-clear signal in this mode. The OVERRUN bit in the SEQA_GDAT register will NOT contribute to generation of an overrun interrupt/DMA trigger since it is assumed this register may not be utilized in this mode.	

Table 435: A/D Conversion Sequence A Control Register (SEQA_CTRL, addresses 0x4000 0008 (ADC0) and 0x4008 0008 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
31	SEQA_ENA		Sequence Enable. In order to avoid spuriously triggering the sequence, care should be taken to only set the SEQA_ENA bit when the selected trigger input is in its INACTIVE state (as defined by the TRIGPOL bit). If this condition is not met, the sequence will be triggered immediately upon being enabled.	0
		0	Disabled. Sequence A is disabled. Sequence A triggers are ignored. If this bit is cleared while sequence A is in progress, the sequence will be halted at the end of the current conversion. After the sequence is re-enabled, a new trigger will be required to restart the sequence beginning with the next enabled channel.	
		1	Enabled. Sequence A is enabled.	

28.6.4 A/D Conversion Sequence B Control Register

There are two, independent conversion sequences that can be configured, each consisting of a set of conversions on one or more channels. This control register specifies the channel selection and trigger conditions for the B sequence, as well bits to allow software to initiate that conversion sequence.

To avoid conversions on spurious triggers, only change the trigger configuration when the conversion sequence is disabled. A conversion can be triggered by software or hardware in the conversion sequence, but if conversions are triggered by software only, spurious hardware triggers must be prevented. See Section 28.3.1 “Perform a single ADC conversion triggered by software”.

Remark: Set the BURST and SEQU_ENA bits at the same time.

Table 436: A/D Conversion Sequence B Control Register (SEQB_CTRL, 0x4000 000C (ADC0) and 0x4008 000C (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
11:0	CHANNELS		Selects which one or more of the twelve channels will be sampled and converted when this sequence is launched. A 1 in any bit of this field will cause the corresponding channel to be included in the conversion sequence, where bit 0 corresponds to channel 0, bit 1 to channel 1 and so forth. When this conversion sequence is triggered, either by a hardware trigger or via software command, A/D conversions will be performed on each enabled channel, in sequence, beginning with the lowest-ordered channel. **Remark:** This field can ONLY be changed while the SEQB_ENA bit (bit 31) is LOW. It is permissible to change this field and set bit 31 in the same write.	0x00
15:12	TRIGGER		Selects which of the available hardware trigger sources will cause this conversion sequence to be initiated. Program the trigger input number in this field. **Remark:** In order to avoid generating a spurious trigger, it is recommended writing to this field only when the SEQA_ENA bit (bit 31) is low. It is safe to change this field and set bit 31 in the same write.	0x0
17:16	-		Reserved.	-
18	TRIGPOL		Select the polarity of the selected input trigger for this conversion sequence. **Remark:** In order to avoid generating a spurious trigger, it is recommended writing to this field only when the SEQA_ENA bit (bit 31) is low. It is safe to change this field and set bit 31 in the same write.	0
		0	Negative edge. A negative edge launches the conversion sequence on the selected trigger input.	
		1	Positive edge. A positive edge launches the conversion sequence on the selected trigger input.	
19	SYNCBYPASS		Setting this bit allows the hardware trigger input to bypass synchronization flip-flops stages and therefore shorten the time between the trigger input signal and the start of a conversion. There are slightly different criteria for whether or not this bit can be set depending on the clock operating mode: Synchronous mode: Synchronization may be bypassed (this bit may be set) if the selected trigger source is already synchronous with the main system clock (eg. coming from an on-chip, system-clock-based timer). Whether this bit is set or not, a trigger pulse must be maintained for at least one system clock period. Asynchronous mode: Synchronization may be bypassed (this bit may be set) if it is certain that the duration of a trigger input pulse will be at least one cycle of the ADC clock (regardless of whether the trigger comes from and on-chip or off-chip source). If this bit is NOT set, the trigger pulse must at least be maintained for one system clock period.	0
		0	Enable synchronization. The hardware trigger bypass is not enabled.	
		1	Bypass synchronization. The hardware trigger bypass is enabled.	

Table 436: A/D Conversion Sequence B Control Register (SEQB_CTRL, 0x4000 000C (ADC0) and 0x4008 000C (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
25:20	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	N/A
26	START		Writing a 1 to this field will launch one pass through this conversion sequence. The behavior will be identical to a sequence triggered by a hardware trigger. Do not write a 1 to this bit if the BURST bit is set. **Remark:** This bit is only set to a 1 momentarily when written to launch a conversion sequence. It will consequently always read-back as a zero.	0
27	BURST		Writing a 1 to this bit will cause this conversion sequence to be continuously cycled through. Other B triggers will be ignored while this bit is set. Repeated conversions can be halted by clearing this bit. The sequence currently in progress will be completed before conversions are terminated.	0
28	SINGLESTEP		When this bit is set, a hardware trigger or a write to the START bit will launch a single conversion on the next channel in the sequence instead of the default response of launching an entire sequence of conversions. Once all of the channels comprising a sequence have been converted, a subsequent trigger will repeat the sequence beginning with the first enabled channel. Interrupt generation will still occur either after each individual conversion or at the end of the entire sequence, depending on the state of the MODE bit.	0
29	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	N/A
30	MODE		Indicates whether the primary method for retrieving conversion results for this sequence will be accomplished via reading the global data register (SEQB_GDAT) at the end of each conversion, or the individual channel result registers at the end of the entire sequence. Impacts when conversion-complete interrupt/DMA triggers for sequence-B will be generated and which overrun conditions contribute to an overrun interrupt as described below:	0
		0	End of conversion. The sequence B interrupt/DMA trigger will be set at the end of each individual A/D conversion performed under sequence B. This flag will mirror the DATAVALID bit in the SEQB_GDAT register. The OVERRUN bit in the SEQB_GDAT register will contribute to generation of an overrun interrupt if enabled.	
		1	End of sequence. The sequence B interrupt/DMA trigger will be set when the entire set of sequence B conversions completes. This flag will need to be explicitly cleared by software or by the DMA-clear signal in this mode. The OVERRUN bit in the SEQB_GDAT register will NOT contribute to generation of an overrun interrupt since it is assumed this register will not be utilized in this mode.	

Table 436: A/D Conversion Sequence B Control Register (SEQB_CTRL, 0x4000 000C (ADC0) and 0x4008 000C (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
31	SEQB_ENA		Sequence Enable. In order to avoid spuriously triggering the sequence, care should be taken to only set the SEQA_ENA bit when the selected trigger input is in its INACTIVE state (as defined by the TRIGPOL bit). If this condition is not met, the sequence will be triggered immediately upon being enabled.	0
		0	Disabled. Sequence B is disabled. Sequence B triggers are ignored. If this bit is cleared while sequence B is in progress, the sequence will be halted at the end of the current conversion. After the sequence is re-enabled, a new trigger will be required to restart the sequence beginning with the next enabled channel.	
		1	Enabled. Sequence B is enabled.	

28.6.5 A/D Global Data Register A and B

The A/D Global Data Registers contain the result of the most recent A/D conversion completed under each conversion sequence.

Results of A/D conversions can be read in one of two ways. One is to use these A/D Global Data Registers to read data from the ADC at the end of each A/D conversion. Another is to read the individual A/D Channel Data Registers, typically after the entire sequence has completed. It is recommended to use one method consistently for a given conversion sequence.

The global registers are useful in conjunction with DMA operation - particularly when the channels selected for conversion are not sequential (hence the addresses of the individual result registers will not be sequential, making it difficult for the DMA engine to address them). For interrupt-driven code it will more likely be advantageous to wait for an entire sequence to complete and then retrieve the results from the individual channel registers.

Remark: The method to be employed for each sequence should be reflected in the MODE bit in the corresponding ADSEQn_CTRL register since this will impact interrupt and overrun flag generation.

Table 437: A/D Sequence A Global Data Register (SEQA_GDAT, addresses 0x4000 0010 (ADC0) and 0x4008 0010 (ADC1)) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	This field contains the 12-bit A/D conversion result from the most recent conversion performed under conversion sequence associated with this register. The result is the a binary fraction representing the voltage on the currently-selected input channel as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}. DATAVALID = 1 indicates that this result has not yet been read.	NA
17:16	THCMPRANGE	Indicates whether the result of the last conversion performed was above, below or within the range established by the designated threshold comparison registers (THRn_LOW and THRn_HIGH).	
19:18	THCMPCROSS	Indicates whether the result of the last conversion performed represented a crossing of the threshold level established by the designated LOW threshold comparison register (THRn_LOW) and, if so, in what direction the crossing occurred.	
25:20	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
29:26	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 0000 identifies channel 0, 0001 channel 1...).	NA
30	OVERRUN	This bit is set if a new conversion result is loaded into the RESULT field before a previous result has been read - i.e. while the DATAVALID bit is set. This bit is cleared, along with the DATAVALID bit, whenever this register is read. This bit will contribute to an overrun interrupt/DMA trigger if the MODE bit (in SEQA_CTRL) for the corresponding sequence is set to ‘0’ (and if the overrun interrupt is enabled).	0
31	DATAVALID	This bit is set to ‘1’ at the end of each conversion when a new result is loaded into the RESULT field. It is cleared whenever this register is read. This bit will cause a conversion-complete interrupt for the corresponding sequence if the MODE bit (in SEQA_CTRL) for that sequence is set to 0 (and if the interrupt is enabled).	0

Table 438: A/D Sequence B Global Data Register (SEQB_GDAT, 0x4000 0014 (ADC0) and 0x4008 0014 (ADC1)) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	This field contains the 12-bit A/D conversion result from the most recent conversion performed under conversion sequence associated with this register. This will be a binary fraction representing the voltage on the currently-selected input channel as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}. DATAVALID = 1 indicates that this result has not yet been read.	NA
17:16	THCMPRANGE	Indicates whether the result of the last conversion performed was above, below or within the range established by the designated threshold comparison registers (THRn_LOW and THRn_HIGH). Threshold Range Comparison result. 0x0 = In Range: The last completed conversion was greater than or equal to the value programmed into the designated LOW threshold register (THRn_LOW) but less than or equal to the value programmed into the designated HIGH threshold register (THRn_HIGH). 0x1 = Below Range: The last completed conversion on was less than the value programmed into the designated LOW threshold register (THRn_LOW). 0x2 = Above Range: The last completed conversion was greater than the value programmed into the designated HIGH threshold register (THRn_HIGH). 0x3 = Reserved.	
19:18	THCMPCROSS	Indicates whether the result of the last conversion performed represented a crossing of the threshold level established by the designated LOW threshold comparison register (THRn_LOW) and, if so, in what direction the crossing occurred. 0x0 = No threshold Crossing detected: The most recent completed conversion on this channel had the same relationship (above or below) to the threshold value established by the designated LOW threshold register (THRn_LOW) as did the previous conversion on this channel. 0x1 = Reserved. 0x2 = Downward Threshold Crossing Detected. Indicates that a threshold crossing in the downward direction has occurred - i.e. the previous sample on this channel was above the threshold value established by the designated LOW threshold register (THRn_LOW) and the current sample is below that threshold. 0x3 = Upward Threshold Crossing Detected. Indicates that a threshold crossing in the upward direction has occurred - i.e. the previous sample on this channel was below the threshold value established by the designated LOW threshold register (THRn_LOW) and the current sample is above that threshold.	
25:20	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 438: A/D Sequence B Global Data Register (SEQB_GDAT, 0x4000 0014 (ADC0) and 0x4008 0014 (ADC1)) bit description

Bit	Symbol	Description	Reset value
29:26	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 0b0000 identifies channel 0, 0b0001 channel 1...).	NA
30	OVERRUN	This bit is set if a new conversion result is loaded into the RESULT field before a previous result has been read - i.e. while the DATAVALID bit is set. This bit is cleared, along with the DATAVALID bit, whenever this register is read. This bit will contribute to an overrun interrupt/DMA trigger if the MODE bit (in SEQB_CTRL) for the corresponding sequence is set to 0 (and if the overrun interrupt is enabled).	0
31	DATAVALID	This bit is set to 1 at the end of each conversion when a new result is loaded into the RESULT field. It is cleared whenever this register is read. This bit will cause a conversion-complete interrupt for the corresponding sequence if the MODE bit (in SEQB_CTRL) for that sequence is set to 0 (and if the interrupt is enabled).	0

28.6.6 A/D Channel Data Registers 0 to 11

The A/D Channel Data Registers hold the result of the last conversion completed for each A/D channel. They also include status bits to indicate when a conversion has been completed, when a data overrun has occurred, and where the most recent conversion fits relative to the range dictated by the high and low threshold registers.

Results of A/D conversion can be read in one of two ways. One is to use the A/D Global Data Registers for each of the sequences to read data from the ADC at the end of each A/D conversion. Another is to use these individual A/D Channel Data Registers, typically after the entire sequence has completed. It is recommended to use one method consistently for a given conversion sequence.

Remark: The method to be employed for each sequence should be reflected in the MODE bit in the corresponding SEQ_CTRL register since this will impact interrupt and overrun flag generation.

The information presented in the DAT registers always pertains to the most recent conversion completed on that channel regardless of what sequence requested the conversion or which trigger caused it.

The OVERRUN fields for each channel are also replicated in the FLAGS register.

Table 439. A/D Data Registers (DAT[0:11], addresses 0x4000 0020 (DAT0) to 0x4000 004C (DAT11) (ADC0) and 0x4008 0020 (DAT0) to 0x4008 004C (DAT11) (ADC1)) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	This field contains the 12-bit A/D conversion result from the last conversion performed on this channel. This will be a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.	NA
17:16	THCMPRANGE	Threshold Range Comparison result. 0x0 = In Range: The last completed conversion was greater than or equal to the value programmed into the designated LOW threshold register (THRn_LOW) but less than or equal to the value programmed into the designated HIGH threshold register (THRn_HIGH). 0x1 = Below Range: The last completed conversion on was less than the value programmed into the designated LOW threshold register (THRn_LOW). 0x2 = Above Range: The last completed conversion was greater than the value programmed into the designated HIGH threshold register (THRn_HIGH). 0x3 = Reserved.	NA
19:18	THCMPCROSS	Threshold Crossing Comparison result. 0x0 = No threshold Crossing detected: The most recent completed conversion on this channel had the same relationship (above or below) to the threshold value established by the designated LOW threshold register (THRn_LOW) as did the previous conversion on this channel. 0x1 = Reserved. 0x2 = Downward Threshold Crossing Detected. Indicates that a threshold crossing in the downward direction has occurred - i.e. the previous sample on this channel was above the threshold value established by the designated LOW threshold register (THRn_LOW) and the current sample is below that threshold. 0x3 = Upward Threshold Crossing Detected. Indicates that a threshold crossing in the upward direction has occurred - i.e. the previous sample on this channel was below the threshold value established by the designated LOW threshold register (THRn_LOW) and the current sample is above that threshold.	NA
25:20	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 439. A/D Data Registers (DAT[0:11], addresses 0x4000 0020 (DAT0) to 0x4000 004C (DAT11) (ADC0) and 0x4008 0020 (DAT0) to 0x4008 004C (DAT11) (ADC1)) bit description

Bit	Symbol	Description	Reset value
29:26	CHANNEL	This field is hard-coded to contain the channel number that this particular register relates to (i.e. this field will contain 0b0000 for the DAT0 register, 0b0001 for the DAT1 register, etc)	NA
30	OVERRUN	This bit will be set to a 1 if a new conversion on this channel completes and overwrites the previous contents of the RESULT field before it has been read - i.e. while the DONE bit is set. This bit is cleared, along with the DONE bit, whenever this register is read or when the data related to this channel is read from either of the global SEQn_GDAT registers. This bit (in any of the 12 registers) will cause an overrun interrupt/DMA trigger to be asserted if the overrun interrupt is enabled. **Remark:** While it is allowed to include the same channels in both conversion sequences, doing so may cause erratic behavior of the DONE and OVERRUN bits in the data registers associated with any of the channels that are shared between the two sequences. Any erratic OVERRUN behavior will also affect overrun interrupt generation, if enabled.	NA
31	DATAVALID	This bit is set to 1 when an A/D conversion on this channel completes. This bit is cleared whenever this register is read or when the data related to this channel is read from either of the global SEQn_GDAT registers. **Remark:** While it is allowed to include the same channels in both conversion sequences, doing so may cause erratic behavior of the DONE and OVERRUN bits in the data registers associated with any of the channels that are shared between the two sequences. Any erratic OVERRUN behavior will also affect overrun interrupt generation, if enabled.	NA

28.6.7 A/D Compare Low Threshold Registers 0 and 1

These registers set the LOW threshold levels against which A/D conversions on all channels will be compared.

Each channel will either be compared to the THR0_LOW/HIGH registers or to the THR1_LOW/HIGH registers depending on what is specified for that channel in the CHAN_THRSEL register.

A conversion result LESS THAN this value on any channel will cause the THCMP_RANGE status bits for that channel to be set to 0b01. This result will also generate an interrupt/DMA trigger if enabled to do so via the ADCMPINTEN bits associated with each channel in the INTEN register.

If, for two successive conversions on a given channel, one result is below this threshold and the other is equal-to or above this threshold, than a threshold crossing has occurred. In this case the MSB of the THCMP_CROSS status bits will indicate that a threshold crossing has occurred and the LSB will indicate the direction of the crossing. A threshold crossing event will also generate an interrupt/DMA trigger if enabled to do so via the ADCMPINTEN bits associated with each channel in the INTEN register.

Table 440. A/D Compare Low Threshold register 0 (THR0_LOW, addresses 0x4000 0050 (ADC0) and 0x4008 0050 (ADC1)address offset 0x050) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	THRLOW	Low threshold value against which A/D results will be compared	0x000
31:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 441. A/D Compare Low Threshold register 1 (THR1_LOW, address offset 0x054) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	THRLOW	Low threshold value against which A/D results will be compared	0x000
31:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

28.6.8 A/D Compare High Threshold Registers 0 and 1

These registers set the HIGH threshold level against which A/D conversions on all channels will be compared.

Each channel will either be compared to the THR0_LOW/HIGH registers or to the THR1_LOW/HIGH registers depending on what is specified for that channel in the CHAN_THRSEL register.

A conversion result greater than this value on any channel will cause the THCMP status bits for that channel to be set to 0b10. This result will also generate an interrupt/DMA trigger if enabled to do so via the ADCMPINTEN bits associated with each channel in the INTEN register.

Table 442: Compare High Threshold register0 (THR0_HIGH, addresses 0x4000 0058 (ADC0) and 0x4008 0058 (ADC1)) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	THRHIGH	High threshold value against which A/D results will be compared	0x000
31:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 443: Compare High Threshold register 1 (THR1_HIGH, addresses 0x4000 005C (ADC0) and 0x4008 005C (ADC1)address offset 0x05C) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	THRHIGH	High threshold value against which A/D results will be compared	0x000
31:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

28.6.9 A/D Channel Threshold Select register

For each channel, this register indicates which pair of threshold registers conversion results should be compared to.

Table 444: A/D Channel Threshold Select register (CHAN_THRSEL, addresses 0x4000 0060 (ADC0) and 0x4008 0060 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
0	CH0_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 0 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 0 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
1	CH1_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 1 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 1 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
2	CH2_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 2 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 2 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
3	CH3_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 3 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 3 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
4	CH4_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 4 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 4 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
5	CH5_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 5 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 5 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	

Table 444: A/D Channel Threshold Select register (CHAN_THRSEL, addresses 0x4000 0060 (ADC0) and 0x4008 0060 (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
6	CH6_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 6 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 6 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
7	CH7_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 7 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 7 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
8	CH8_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 8 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 8 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
9	CH9_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 9 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 9 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
10	CH10_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 10 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 10 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
11	CH11_THRSEL		Threshold select by channel.	0
		0	Threshold 0. Channel 11 results will be compared against the threshold levels indicated in the THR0_LOW and THR0_HIGH registers	
		1	Threshold 1. Channel 11 results will be compared against the threshold levels indicated in the THR1_LOW and THR1_HIGH registers	
31:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

28.6.10 A/D Interrupt Enable Register

There are four separate interrupt requests generated by the ADC: conversion-complete or sequence-complete interrupts for each of the two sequences, a threshold-comparison out-of-range interrupt, and a data overrun interrupt. The two conversion/sequence-complete interrupts can also serve as DMA triggers.

These interrupts may be combined into one request on some chips if there is a limited number of interrupt slots. This register contains the interrupt-enable bits for each interrupt.

In this register, threshold events selected in the ADCMPINTENn bits are described as follows:

- Disabled: Threshold comparisons on channel n will not generate an A/D threshold-compare interrupt/DMA trigger.
- Outside threshold: A conversion result on channel n which is outside the range specified by the designated HIGH and LOW threshold registers will set the channel n THCMP flag in the FLAGS register and generate an A/D threshold-compare interrupt/DMA trigger.
- Crossing threshold: Detection of a threshold crossing on channel n will set the channel n THCMP flag in the ADFLAGS register and generate an A/D threshold-compare interrupt/DMA trigger.

Remark: Overrun and threshold-compare interrupts related to a particular channel will occur regardless of which sequence was in progress at the time the conversion was performed or what trigger caused the conversion.

Table 445: A/D Interrupt Enable register (INTEN, addresses 0x4000 0064 (ADC0) and 0x4008 0064 (ADC1)address offset 0x064) bit description

Bit	Symbol	Value	Description	Reset value
0	SEQA_INTEN		Sequence A interrupt enable.	0
		0	Disabled. The sequence A interrupt/DMA trigger is disabled.	
		1	Enabled. The sequence A interrupt/DMA trigger is enabled and will be asserted either upon completion of each individual conversion performed as part of sequence A, or upon completion of the entire A sequence of conversions, depending on the MODE bit in the SEQA_CTRL register.	
1	SEQB_INTEN		Sequence B interrupt enable.	0
		0	Disabled. The sequence B interrupt/DMA trigger is disabled.	
		1	Enabled. The sequence B interrupt/DMA trigger is enabled and will be asserted either upon completion of each individual conversion performed as part of sequence B, or upon completion of the entire B sequence of conversions, depending on the MODE bit in the SEQB_CTRL register.	
2	OVR_INTEN		Overrun interrupt enable.	0
		0	Disabled. The overrun interrupt is disabled.	
		1	Enabled. The overrun interrupt is enabled. Detection of an overrun condition on any of the 12 channel data registers will cause an overrun interrupt/DMA trigger. In addition, if the MODE bit for a particular sequence is 0, then an overrun in the global data register for that sequence will also cause this interrupt/DMA trigger to be asserted.	
4:3	ADCMPINTEN0		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
6:5	ADCMPINTEN1		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved.	

Table 445: A/D Interrupt Enable register (INTEN, addresses 0x4000 0064 (ADC0) and 0x4008 0064 (ADC1)address offset 0x064) bit description

Bit	Symbol	Value	Description	Reset value
8:7	ADCMPINTEN2		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
10:9	ADCMPINTEN3		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
12:11	ADCMPINTEN4		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
14:13	ADCMPINTEN5		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
16:15	ADCMPINTEN6		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved.	
18:17	ADCMPINTEN7		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
20:19	ADCMPINTEN8		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
22:21	ADCMPINTEN9		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	

Table 445: A/D Interrupt Enable register (INTEN, addresses 0x4000 0064 (ADC0) and 0x4008 0064 (ADC1)address offset 0x064) bit description

Bit	Symbol	Value	Description	Reset value
		0x2	Crossing threshold.	
		0x3	Reserved	
24:23	ADCMPINTEN10		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
26:25	ADCMPINTEN11		Threshold comparison interrupt enable.	00
		0x0	Disabled.	
		0x1	Outside threshold.	
		0x2	Crossing threshold.	
		0x3	Reserved	
31:27	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

28.6.11 A/D Flag register

The A/D Flags registers contains the four interrupt/DMA trigger flags along with the individual overrun flags that contribute to an overrun interrupt and the component threshold-comparison flags that contribute to that interrupt.

The channel OVERRUN flags mirror those appearing in the individual DAT registers for each channel and indicate a data overrun in each of those registers.

Likewise, the SEQA_OVR and SEQB_OVR bits mirror the OVERRUN bits in the two global data registers (SEQA_GDAT and SEQB_GDAT).

Remark: The SEQn_INT conversion/sequence-complete flags also serve as DMA triggers.

Table 446: A/D Flags register (FLAGS, addresses 0x4000 0068 (ADC0) and 0x4008 0068 (ADC1)) bit description

Bit	Symbol	Description	Reset value
0	THCMP0	Threshold comparison event on Channel 0. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	0
1	THCMP1	Threshold comparison event on Channel 1. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
2	THCMP2	Threshold comparison event on Channel 2. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
3	THCMP3	Threshold comparison event on Channel 3. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00

Table 446: A/D Flags register (FLAGS, addresses 0x4000 0068 (ADC0) and 0x4008 0068 (ADC1)) bit description

Bit	Symbol	Description	Reset value
4	THCMP4	Threshold comparison event on Channel 4. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
5	THCMP5	Threshold comparison event on Channel 5. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
6	THCMP6	Threshold comparison event on Channel 6. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
7	THCMP7	Threshold comparison event on Channel 7. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
8	THCMP8	Threshold comparison event on Channel 8. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
9	THCMP9	Threshold comparison event on Channel 9. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
10	THCMP10	Threshold comparison event on Channel 10. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
11	THCMP11	Threshold comparison event on Channel 11. Set to 1 upon either an out-of-range result or a threshold-crossing result if enabled to do so in the INTEN register. This bit is cleared by writing a 1.	00
12	OVERRUN0	Mirrors the OVERRRUN status flag from the result register for A/D channel 0	0
13	OVERRUN1	Mirrors the OVERRRUN status flag from the result register for A/D channel 1	0
14	OVERRUN2	Mirrors the OVERRRUN status flag from the result register for A/D channel 2	0
15	OVERRUN3	Mirrors the OVERRRUN status flag from the result register for A/D channel 3	0
16	OVERRUN4	Mirrors the OVERRRUN status flag from the result register for A/D channel 4	0
17	OVERRUN5	Mirrors the OVERRRUN status flag from the result register for A/D channel 5	0
18	OVERRUN6	Mirrors the OVERRRUN status flag from the result register for A/D channel 6	0
19	OVERRUN7	Mirrors the OVERRRUN status flag from the result register for A/D channel 7	0
20	OVERRUN8	Mirrors the OVERRRUN status flag from the result register for A/D channel 8	0
21	OVERRUN9	Mirrors the OVERRRUN status flag from the result register for A/D channel 9	0
22	OVERRUN10	Mirrors the OVERRRUN status flag from the result register for A/D channel 10	0
23	OVERRUN11	Mirrors the OVERRRUN status flag from the result register for A/D channel 11	0
24	SEQA_OVR	Mirrors the global OVERRUN status flag in the SEQA_GDAT register	0
25	SEQB_OVR	Mirrors the global OVERRUN status flag in the SEQB_GDAT register	0
27:26	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 446: A/D Flags register (FLAGS, addresses 0x4000 0068 (ADC0) and 0x4008 0068 (ADC1)) bit description

Bit	Symbol	Description	Reset value
28	SEQA_INT	Sequence A interrupt/DMA trigger. If the MODE bit in the SEQA_CTRL register is 0, this flag will mirror the DATAVALID bit in the sequence A global data register (SEQA_GDAT), which is set at the end of every A/D conversion performed as part of sequence A. It will be cleared automatically when the SEQA_GDAT register is read. If the MODE bit in the SEQA_CTRL register is 1, this flag will be set upon completion of an entire A sequence. In this case it must be cleared by writing a 1 to this SEQA_INT bit. This interrupt must be enabled in the INTEN register.	0
29	SEQB_INT	Sequence A interrupt/DMA trigger. If the MODE bit in the SEQB_CTRL register is 0, this flag will mirror the DATAVALID bit in the sequence A global data register (SEQB_GDAT), which is set at the end of every A/D conversion performed as part of sequence B. It will be cleared automatically when the SEQB_GDAT register is read. If the MODE bit in the SEQB_CTRL register is 1, this flag will be set upon completion of an entire B sequence. In this case it must be cleared by writing a 1 to this SEQB_INT bit. This interrupt must be enabled in the INTEN register.	0
30	THCMP_INT	Threshold Comparison Interrupt. This bit will be set if any of the 12 THCMP flags in the lower bits of this register are set to 1 (due to an enabled out-of-range or threshold-crossing event on any channel). Each type of threshold comparison interrupt on each channel must be individually enabled in the INTEN register to cause this interrupt. This bit will be cleared when all of the component flags in bits 11:0 are cleared via writing 1s to those bits.	0
31	OVR_INT	Overrun Interrupt flag. Any overrun bit in any of the individual channel data registers will cause this interrupt. In addition, if the MODE bit in either of the SEQn_CTRL registers is 0 then the OVERRUN bit in the corresponding SEQn_GDAT register will also cause this interrupt. This interrupt must be enabled in the INTEN register. This bit will be cleared when all of the individual overrun bits have been cleared via reading the corresponding data registers.	0

28.6.12 A/D trim register

The A/D trim register configures the ADC for the appropriate operating range of the analog supply voltage VDDA.

Remark: Failure to set the VRANGE bit correctly causes the ADC to return incorrect conversion results.

Table 447: A/D Flags register (TRM, addresses 0x4000 006C (ADC0) and 0x4008 006C (ADC1)) bit description

Bit	Symbol	Value	Description	Reset value
4:0	-		Reserved.	0
5	VRANGE		Reserved.	00
		0	High voltage. VDDA = 2.7 V to 3.6 V.	
		1	Low voltage. VDDA = 2.4 V to 2.7 V.	
31:6	-		Reserved.	00

28.7 Functional description

28.7.1 Conversion Sequences

A conversion sequence is a single pass through a series of A/D conversions performed on a selected set of A/D channels. Software can set-up two independent conversion sequences, either of which can be triggered by software or by a transition on one of the hardware triggers. Each sequence can be triggered by a different hardware trigger. One of these conversion sequences is referred to as the A sequence and the other as the B sequence. It is not necessary to employ both sequences.

An optional single-step mode allows advancing through the channels of a sequence one at a time on each successive occurrence of a trigger.

The user can select whether a trigger on the B sequence can interrupt an already-in-progress A sequence. The B sequence, however, can never be interrupted by an A trigger.

28.7.2 Hardware-triggered conversion

Software can select which of these hardware triggers will launch each conversion sequence and it can specify the active edge for the selected trigger independently for each conversion sequence.

For each conversion sequence, if a designated trigger event occurs, one single cycle through that conversion sequence will be launched unless:

- The BURST bit in the ADSEQn_CTRL register for this sequence is set to 1.
- The requested conversion sequence is already in progress.
- A set of conversions for the alternate conversion sequence is already in progress except in the case of a B trigger interrupting an A sequence if the A sequence is set to LOWPRIO.

If any of these conditions is true, the new trigger event will be ignored and will have no effect.

In addition, if the single-step bit for a sequence is set, each new trigger will cause a single conversion to be performed on the next channel in the sequence rather than launching a pass through the entire sequence.

If the A sequence is enabled to be interrupted (i.e. the LOWPRIO bit in the SEQA_CTRL register is set) and a B trigger occurs while an A sequence is in progress, then the following will occur:

- The A/D conversion which is currently in-progress will be aborted.
- The A sequence will be paused, and the B sequence will immediately commence.
- The interrupted A sequence will resume after the B sequence completes, beginning with the conversion that was aborted when the interruption occurred. The channel for that conversion will be re-sampled.

28.7.2.1 Avoiding spurious hardware triggers

Care should be taken to avoid generating a spurious trigger when writing to the SEQn_CTRL register to change the trigger selected for the sequence, switch the polarity of the selected trigger, or to enable the sequence for operation.

In general, the TRIGGER and TRIGPOL bits can be set only when the SEQn_ENA bit is LOW. The SEQn_ENA bit itself should only be set when the selected trigger input is in its INACTIVE state (as designated by the TRIGPOL bit). If this condition is not met, a trigger will be generated immediately upon enabling the sequence - even though no actual transition has occurred on the trigger input.

28.7.3 Software-triggered conversion

There are two ways that software can trigger a conversion sequence:

1. **Start Bit:** The first way to software-trigger an sequence is by setting the START bit in the corresponding SEQn_CTRL register. The response to this is identical to occurrence of a hardware trigger on that sequence. Specifically, one cycle of conversions through that conversion sequence will be immediately triggered except as indicated above.
2. **Burst Mode**: The other way to initiate conversions is to set the BURST bit in the SEQn_CTRL register. As long as this bit is 1 the designated conversion sequence will be continuously and repetitively cycled through. Any new software or hardware trigger on this sequence will be ignored.

If a bursting A sequence is allowed to be interrupted (i.e. the LOWPRIO bit in its SEQA_CTRL register is set to 1 and a software or hardware trigger for the B sequence occurs, then the burst will be immediately interrupted and a B sequence will be initiated. The interrupted A sequence will resume continuous cycling, starting with the aborted conversion, after the alternate sequence has completed.

28.7.4 Interrupts

There are four interrupts that can be generated by the ADC:

- Conversion-Complete or Sequence-Complete interrupts for sequences A and B
- Threshold-Compare Out-of-Range Interrupt
- Data Overrun Interrupt

Any of these interrupt requests may be individually enabled or disabled in the INTEN register.

28.7.4.1 Conversion-Complete or Sequence-Complete interrupts

For each of the two sequences, an interrupt/DMA trigger can either be asserted at the end of each A/D conversion performed as part of that sequence or when the entire sequence of conversions is completed. The MODE bits in the SEQn_CTRL registers select between these alternative behaviors.

If the MODE bit for a sequence is 0 (conversion-complete mode) then the interrupt flag for that sequence will reflect the state of the DATAVALID bit in the global data register (SEQn_GDAT) for that sequence. In this case, reading the SEQn_GDAT register will automatically clear the interrupt/DMA trigger.

If the MODE bit for the sequence is 1 (sequence-complete mode) then the interrupt flag must be written-to by software to clear it (except when used as a DMA trigger, in which case it will be cleared in hardware by the DMA engine).

28.7.4.2 Threshold-Compare Out-of-Range Interrupt

Every conversion performed on any channel is automatically compared against a designated set of low and high threshold levels specified in the THRn_HIGH and THRn_LOW registers. The results of this comparison on any individual channel(s) can be enabled to cause a threshold-compare interrupt if that result was above or below the range specified by the two thresholds or, alternatively, if the result represented a crossing of the low threshold in either direction.

This flag must be cleared by a software write to clear the individual THCMP flags in the FLAGS register.

28.7.4.3 Data Overrun Interrupt

This interrupt/DMA trigger will be asserted if any of the OVERRUN bits in the individual channel data registers are set. In addition, the OVERRUN bits in the two sequence global data (SEQn_GDAT) registers will cause this interrupt/DMA trigger IF the MODE bit for that sequence is set to 0 (conversion-complete mode).

This flag will be cleared when the OVERRUN bit that caused it is cleared via reading the register containing it.

Note that the OVERRUN bits in the individual data registers are cleared when data related to that channel is read from either of the global data registers as well as when the individual data registers themselves are read.

28.7.5 Optional Operating Modes

There are three optional modes of A/D operation which may be selected in the CTRL register:

- 10-bit mode: In this mode two bits of ADC accuracy are sacrificed in order to double the conversion rate. The maximum ADC clock rate when this mode is selected is increased to 72 MHz. When this mode is selected, the two LSBs of result data will automatically be forced to zero.
- Low-power mode. When this mode is selected, the analog portions of the ADC are automatically shut down when no conversions are in progress. The ADC is automatically restarted whenever any hardware or software trigger event occurs. This mode can save an appreciable amount of power when the ADC is not in continuous use, but at the expense of a delay between the trigger event and the onset of sampling and conversion.
- Asynchronous mode: In this mode the user has greater flexibility in selecting the ADC clock frequency to better achieve the maximum ADC conversion rate without restricting the clock rate for other peripherals. The penalty for using this mode may be longer latency and greater uncertainty in response to a hardware trigger.

None of these three optional modes are mutually exclusive.

28.7.6 ADC vs. digital receiver

The analog ADC input must be selected via the switch matrix registers in order to get accurate voltage readings on the monitored pin. In the IOCON, the pull-up and pull-down resistors should both be disabled using the MODE bits. For a pin hosting an ADC input, it is not possible to have a have a digital function selected and yet get valid ADC readings. An inside circuit disconnects ADC hardware from the associated pin whenever a digital function is selected on that pin.

28.7.7 DMA control

The sequence-A or sequence-B conversion/sequence-complete interrupts may also be used to generate a DMA transfer trigger. To generate a DMA transfer the same conditions must be met as the conditions for generating an interrupt (see Section 28.7.4 and Section 28.6.10).

Remark: If the DMA is used, the ADC interrupt must be disabled in the NVIC.

For DMA transfers, only burst requests are supported. The burst size can be set to one in the DMA channel control register. If the number of ADC channels is not equal to one of the other DMA-supported burst sizes (applicable DMA burst sizes are 1, 4, 8), set the burst size to one.

The DMA transfer size determines when a DMA interrupt is generated. The transfer size can be set to the number of ADC channels being converted. Non-contiguous channels can be transferred by the DMA using the scatter/gather linked lists.

28.7.8 Hardware trigger source selection

Each ADC has multiple on-chip and off-chip hardware trigger sources (see Table 428 and Table 429). The trigger to be used for each conversion sequence is specified in the TRIGGER fields in the two SEQn_CTRL registers.

29.1 How to read this chapter

The DAC is available on all parts.

29.2 Features

- 12-bit digital to analog converter.
- Supports DMA.
- Internal timer or external pin trigger for staged, jitter-free DAC conversion sequencing.

29.3 Basic configuration

Configure the DAC as follows:

- Use the SYSAHBCLKCTRL0 register (Table 50) to enable the clock to the DAC.
- The DAC interrupt is connected to slot #39 in the NVIC and to DMA channel #13.
- Use the switch matrix to enable the DAC analog output. See Table 449.
- The power to the DAC blocks is controlled by the PDRUNCFG register in the SYSCON block. See Table 75.

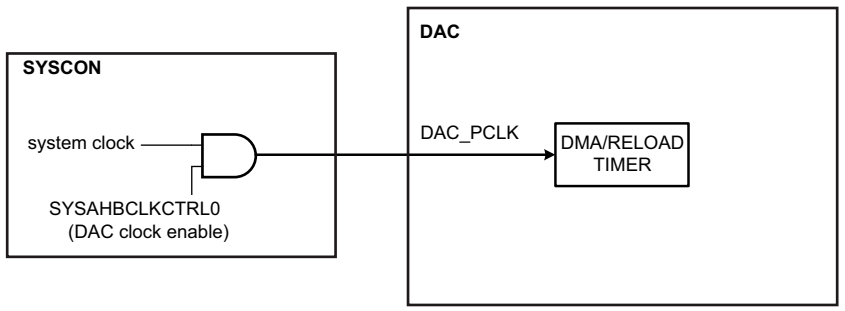


Fig 87. DAC timing

29.4 Pin description

Remark: Analog power and ground should typically be the same voltage levels as V_{DD} and V_{SS} but should be isolated to minimize noise and error.

Table 448. DAC common supply and reference pins

Pin	Description
V_{DDA}	Analog supply voltage. VREFP_DAC_VDDCMP must not exceed the voltage level on V_{DDA}. **Remark:** The supply voltage VDD must be equal or lower than VDDA.
V_{SSA}	Analog ground.
VREFP_DAC_VDDCMP	DAC Positive reference voltage. Also connected to the ladder reference voltage source of the analog comparators.
VREFN	Negative reference voltage.

Table 449. DAC pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
DAC_OUT	AO	external	PIO0_12	PINENABLE0	Table 123	Analog Output. The voltage on this pin (with respect to V_{SSA}) is VALUE × ((VREFP_DAC - VREFN)/4096) + VREFN, where VALUE is the contents of the DAC VAL register. The DAC output is disabled in deep-sleep, power-down, or deep power-down mode. **Remark:** This pin is not 5 V tolerant.
DAC_PINTRIG	I	external	any pin	PINASSIGN11	Table 118	DAC trigger input.
DAC_SHUTOFF	I	external	any pin	PINASSIGN12	Table 119	Hardware shut-off input.

29.5 General description

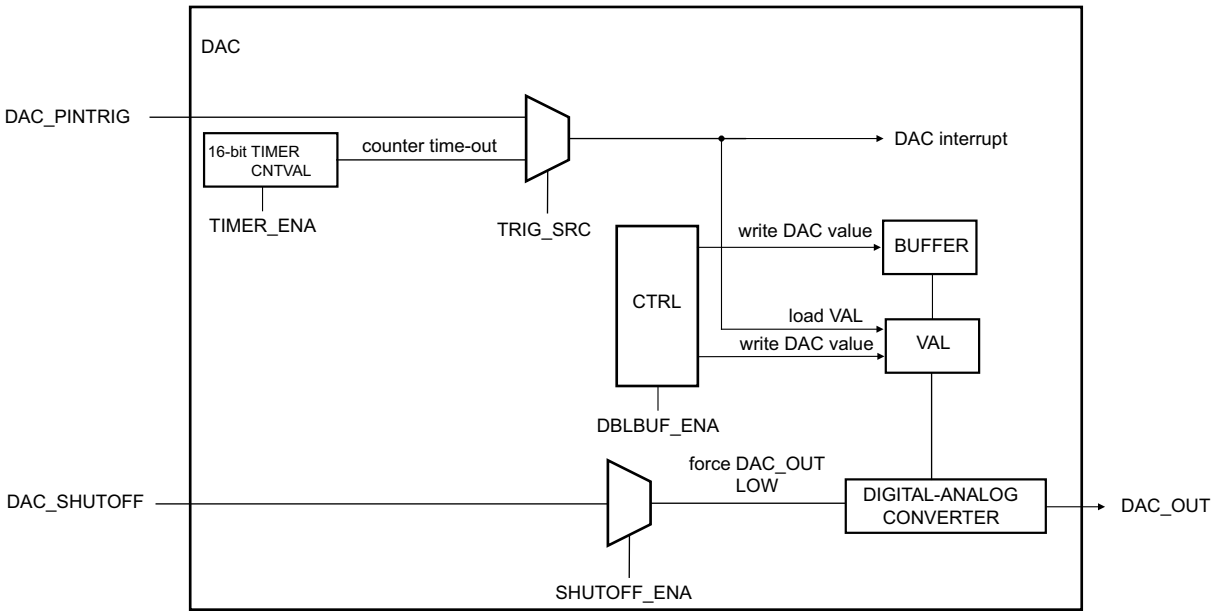


Fig 88. DAC block diagram

29.5.1 DMA/Reload Timer

The first hardware trigger option is an internal, dedicated timer. This 16-bit timer, clocked by the system clock, can be programmed to time-out at regular, periodic intervals. When the timer is enabled, it counts down from the programmed CNTVAL. When the timer reaches 0, an interrupt/DMA flag is set to request a new DAC value from the CPU or the DMA engine and the timer loads the CNTVAL and starts counting again. If double-buffering is enabled, the DAC VAL register is simultaneously reloaded from the DAC VAL pre-buffer.

The interrupt/DMA flag is automatically cleared whenever the DAC VAL register (or the pre-buffer if double-buffering is enabled) is written-to.

In order to use the internal timer, it must be selected as the trigger source in the TRIG_SRC field of the DAC CTRL register, and counting must also be enabled by setting the CNT_ENA bit, also in the DAC CTRL register.

The count value for the timer is specified in the DAC CNTVAL register. The timer will repeat at the fixed rate of the system clock divided by CNTVAL+1, as long as the CNT_ENA bit is set.

Note that the contents of the DAC CNTVAL register is read and write accessible, but the timer itself is not accessible for either read or write.

29.5.2 Alternative DMA/Reload Triggers

Aside from the timer, the DAC_PINTRIG function is available as an external conversion trigger.

If the external trigger input is selected as the trigger source, the polarity of the trigger (rising or falling edge) can be selected via the POLARITY bit in the DAC CTRL register. The SYNC_BYPASS bit, in the same register, provides the capability to save one clock of latency if the selected trigger source is guaranteed to already be synchronous to the system clock. This bit must not be set otherwise.

Response to a detected edge of the designated polarity on the selected trigger input is the same as described above for a timer time-out. The interrupt/DMA flag will immediately be set and, if double-buffering is enabled, the DAC VAL register will be simultaneously reloaded from the pre-buffer. The interrupt/DMA flag will be cleared when the DAC VAL register or pre-buffer is written-to.

29.5.3 Double buffering

Double-buffering is enabled only when the DBLBUF_ENA bit in the DAC CTRL register is set. In this case, any write to the DAC VAL register will only load the pre-buffer, which shares its register address with the DAC VAL register. The DAC VAL itself will be loaded from the pre-buffer whenever the designated hardware trigger occurs.

Reading the DAC VAL register will only return the contents of the DAC VAL register itself, not the contents of the pre-buffer register.

If the DBLBUF_ENA bit is 0, any writes to the DAC VAL address will go directly to the DACVAL register.

It does not make sense to use the double-buffering feature unless hardware triggering is also enabled. Setting the DBLBUF_ENA bit will have no effect if hardware triggering is disabled.

29.6 Register description

Table 450. Register overview: DAC (base address 0x4000 4000)

Name	Access	Address offset	Description	Reset value	Reference
VAL	R/W	0x000	D/A Converter Value Register. This register contains the digital value to be converted to analog.	0x0	Table 451
CTRL	R/W	0x004	DAC Control register. This register contains bits to configure DAC operation and the interrupt/DMA request flag.	0x0	Table 452
CNTVAL	R/W	0x008	DAC Counter Value register. This register contains the reload value for the internal DAC DMA/Interrupt timer.	0x0	Table 453

29.6.1 D/A Converter Value Register

This read/write register contains the digital value to be converted to analog. Bits 3:0 are reserved for future, higher-resolution D/A converters.

This register can be directly written-to or automatically loaded from a pre-buffer if the double-buffering option is enabled and a hardware reload trigger is selected.

Table 451: D/A Converter Register (VAL, address 0x4000 0000) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved. Software should only write zeros to unused bits.	NA
15:4	VALUE	The voltage on the DAC_OUT pin (with respect to V_{SSA}) is VALUE × ((VREFP_DAC - VREFN)/4096) + VREFN. This voltage will be stable the selected settling time (specified by the BIAS field in the DAC Control Register) after this field is modified.	0x0
31:16	-	Reserved. Software should only write zeros to unused bits.	NA

29.6.2 D/A Converter Control register

This read/write register controls DAC operation including hardware triggering and Interrupt/DMA generation. It also contains the Interrupt/DMA request flag.

The settling times noted in the description of the BIAS bits are valid for a capacitance load on the DAC_OUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling times longer than specified. See LPC15xx data sheet.

Table 452. D/A Control register (CTRL, address 0x4000 4004) bit description

Bit	Symbol	Value	Description	Reset Value
0	INT_DMA_FLAG		Interrupt/DMA request flag. This bit is read-only. 0 = This bit is cleared upon any write to the DAC VAL register. 1 = This bit is set by hardware only if a hardware trigger has been selected as follows: - If the internal timer is selected, this bit will be set when the timer times-out. - If an external trigger input is selected, this bit will be set when a transition of the specified polarity is detected on the selected input.	0
3:1	TRIG_SRC		Hardware Trigger Source: If any of these hardware trigger sources are selected, an interrupt/DMA request will be generated when the specified trigger occurs. In addition, if double-buffering is enabled (the DBLBUF_ENA bit is set), the DACVAL register will be loaded from the pre-buffer at the same time. All other values are reserved.	000
		0x0	Internal. Selects the internal timer as the trigger source provided the timer is enabled (the TIMER_ENA bit is set). Otherwise (if the timer is not enabled), hardware triggering is disabled. If hardware triggering is disabled no interrupt or DMA requests will be generated. Double-buffering of the DAC VAL register is not useful and cannot be enabled when hardware triggering is disabled.	
		0x1	Pin. External DAC_TRIG port input is selected. Also select this function in the register in the switch matrix.	
4	POLARITY		Specifies the polarity of the selected external trigger input. Does not apply if the TRIG_SRC field is set to 0.	0
		0	Rising. A trigger will be asserted when a RISING edge is detected on the selected external trigger input.	
		1	Falling. A trigger will be asserted when a FALLING edge is detected on the selected external trigger input.	
5	SYNC_BYPASS		Permits bypassing of one synchronization flip-flop, if not required. Does not apply if the TRIG_SRC field is set to 0.	0
		0	Synchronize. The selected trigger input will be synchronized to the system clock prior to edge-detection.	
		1	Not synchronize. The selected trigger input will not be synchronized to the system clock prior to edge-detection. This will save one clock of latency. This bit should only be set f the selected hardware input trigger is from a source that is guaranteed to already be synchronous to the system clock.	
6	TIMER_ENA		Timer Enable	0
		0	Disabled. The internal timer is disabled. If the TRIG_SEL field is also set to 000 then hardware triggering is disabled.	
		1	Enabled. The internal timer is enabled and counting. Note: This bit should only be set after a valid count value has been programmed into the DAC CNTVAL register.	

Table 452. D/A Control register (CTRL, address 0x4000 4004) bit description

Bit	Symbol	Value	Description	Reset Value
7	DBLBUF_ENA		Double-Buffer Enable.	0
		0	Disabled. Double-buffering of the DAC VAL register is disabled. Software writes to the DACVAL address will directly modify the DAC data presented to the D/A converter. Hardware trigger events, if selected, will not affect the DAC VAL contents.	
		1	Enabled. The double-buffering feature in the DAC VAL register is enabled. Writes to the DACVAL register are written to a pre-buffer and then transferred to the DACVAL when the specified hardware trigger occurs. Setting this bit will have no effect if hardware triggering is disabled. Double-buffering is of no value under this condition.	
8	SHUTOFF_ENA		Shutoff Enable	0
		0	Disabled. The hardware DAC-shutoff feature is disabled.	
		1	Enabled. The hardware DAC-shutoff feature is enabled. Whenever the DAC_SHUTOFF (port pin) input is HIGH, the DAC output voltage will be forced to zero. The DAC output will return to the value specified in the DAC VAL register once the input pin returns to the LOW state.	
9	SHUTOFF_FLAG		Shutoff Flag. This is a read-only bit. Reflects the state of the DAC_SHUTOFF input if the Shutoff feature is enabled. 0 = DAC_SHUTOFF (port pin) input is low. DAC is outputting the voltage specified in the DAC VAL register. 1 = DAC_SHUTOFF (port pin) input is high. The DAC output is forced to zero. **Remark:** This bit serves as a flag only, If a processor interrupt is desired when a DAC shutoff condition occurs, that can be accomplished by enabling the port pin selected as the DAC_SHUTOFF pin to directly generate a port interrupt.	0
12:10	-		Reserved.	0b111
31:13	-		Reserved. Software should only write zeros to unused bits.	NA

29.6.3 D/A Converter Counter Value register

This read/write register contains the reload value for the Interrupt/DMA timer.

Table 453: D/A Converter Counter Value register (CNTVAL, address 0x4000 4008) bit description

Bit	Symbol	Description	Reset Value
15:0	CNTVAL	16-bit reload value for the internal DAC interrupt/DMA timer. The timer will overflow at the fixed rate of the system clock divided by CNTVAL+1. Upon each overflow an interrupt/DMA request will be generated and the DAC VAL register contents will be updated if double-buffering is enabled.	0
31:16	-	Reserved. Software should only write zeros to unused bits.	

30.1 How to read this chapter

Four comparators are available on all parts. The number of available external comparator inputs depends on the package size.

Table 454. ACMP available analog inputs

Package	Common inputs	ACMP0	ACMP1	ACMP2	ACMP3
LQFP48	ACMP_I1	ACMP0_I3; ACMP0_I4	ACMP1_I3	ACMP2_I3	-
LQFP64	ACMP_I1; ACMP_I2	ACMP0_I3; ACMP0_I4	ACMP1_I3; ACMP1_I4	ACMP2_I3; ACMP2_I4	ACMP3_I3; ACMP3_I4
LQFP100	ACMP_I1; ACMP_I2	ACMP0_I3; ACMP0_I4	ACMP1_I3; ACMP1_I4	ACMP2_I3; ACMP2_I4	ACMP3_I3; ACMP3_I4

30.2 Features

- 0.9 V internal band gap voltage reference selectable as either positive or negative input on each comparator.
- Temperature sensor voltage selectable as either positive or negative input on each comparator.
- 32-stage voltage ladder internal reference for selectable voltages on each comparator; configurable on either positive or negative comparator input.
- Voltage ladder source voltage is selectable from an external pin or the 3.3 V analog voltage supply.
- Voltage ladder can be separately powered down for applications only requiring the comparator function.
- Relaxation oscillator circuitry output for a 555 style timer operation using comparator blocks 0 and 1.
- Individual comparator outputs can be connected internally to the SCT and ADC trigger inputs or the external pins.
- Separate interrupt for each comparator.
- Pin filter included on each comparator output.
- Three propagation delay values are programmable to optimize between speed and power consumption.
- Seven selectable inputs. Fully configurable on either the positive side or the negative input channel.

30.3 Basic configuration

The analog comparator block consists of four independent comparators. The reset control and clock for register access is common for all four comparators. Each comparator has its own power control and interrupt.

- Use the SYSAHBCLKCTRL0 register (Table 50) to enable the clock to the comparator register interface.
- Clear the peripheral reset for the entire comparator block using the PRESETCTRL0 register (Table 35).
- Each comparator creates one interrupt connected to slots #40 to #43 in the NVIC.
- Use the switch matrix to enable the comparator inputs (Table 456).
- The power to each comparator is controlled by the PDRUNCFG register in the SYSCON block. See Table 75.

  Remark: On reset, the comparators are disabled.
- Comparator outputs are connected to SCT inputs via SCT input mux registers. See Table 127 to Table 130.
- Comparator outputs can trigger ADC conversion via ADC input trigger selectors. See Table 428 and Table 429.

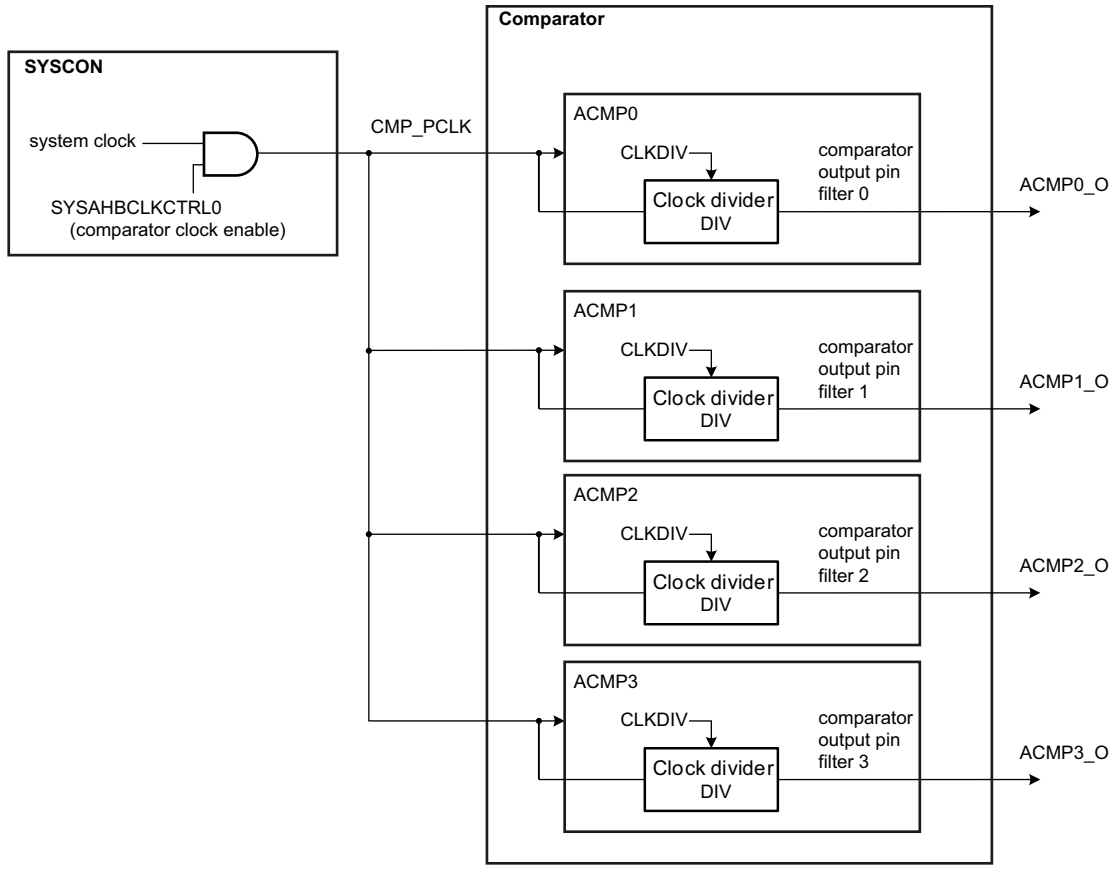


Fig 89. Comparator clocking

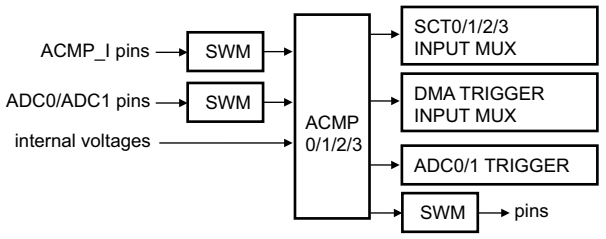


Fig 90. Analog comparator connections

30.3.1 Comparator inputs and outputs

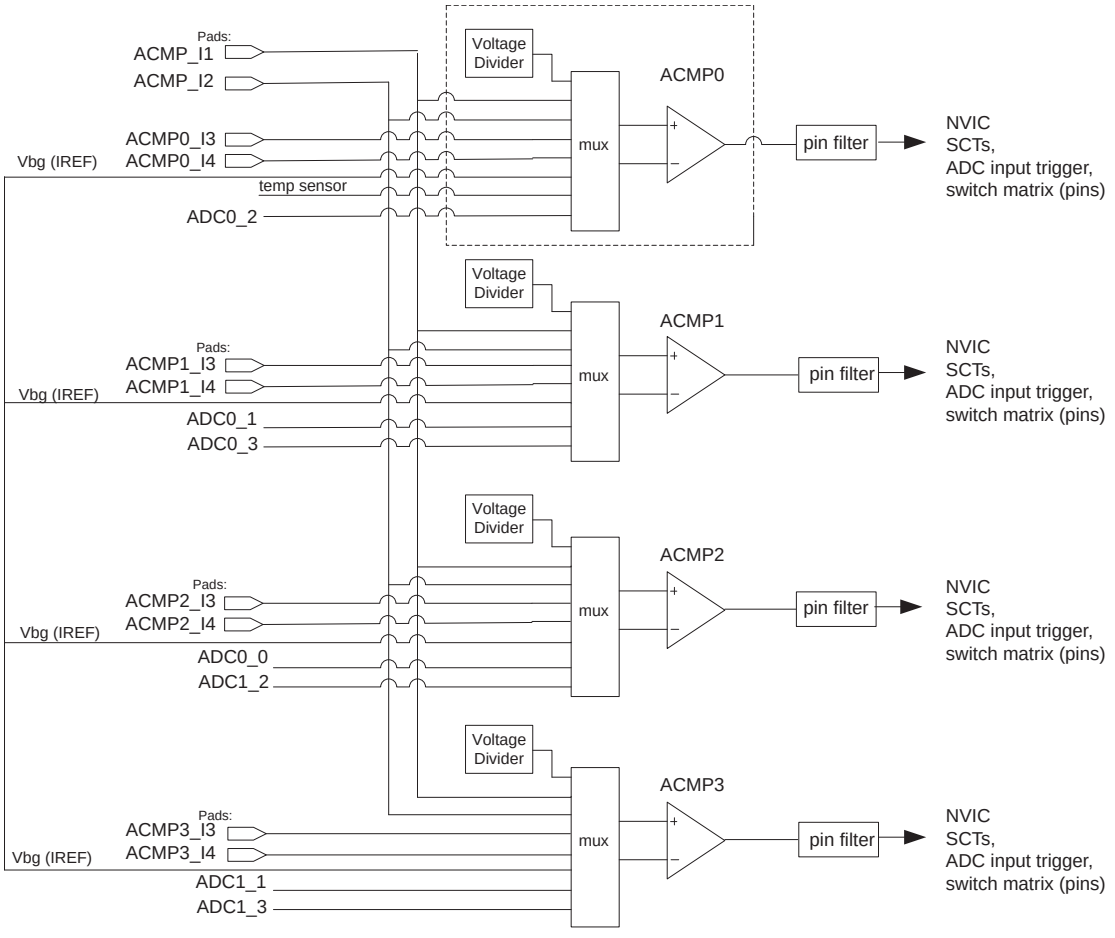


Fig 91. Comparator inputs and outputs

30.4 Pin description

Table 455. Analog comparator common supply and reference pins

Pin	Description
VREFP_DAC_VDDCMP	Voltage ladder reference (VDDCMP). Also used as the positive reference voltage for the DAC.
V_{DDA}	Analog supply voltage. VREFP_DAC_VDDCMP must not exceed the voltage level on V_{DDA}.
V_{SSA}	Analog ground.

Table 456. Analog comparator 0/1/2/3 pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
ACMP0_I3	AI	external	PIO0_26	PINENABLE0	Table 123	Comparator 0 input a
ACMP0_I4	AI	external	PIO0_25	PINENABLE0	Table 123	Comparator 0 input b
ACMP1_I3	AI	external	PIO0_28	PINENABLE0	Table 123	Comparator 1 input a
ACMP1_I4	AI	external	PIO1_10	PINENABLE0	Table 123	Comparator 1 input b
ACMP2_I3	AI	external	PIO0_29	PINENABLE0	Table 123	Comparator 2 input a
ACMP2_I4	AI	external	PIO1_9	PINENABLE1	Table 124	Comparator 2 input b
ACMP3_I3	AI	external	PIO1_8	PINENABLE1	Table 124	Comparator 3 input a
ACMP3_I4	AI	external	PIO1_7	PINENABLE1	Table 124	Comparator 3 input b
ACMP_I1	AI	external	PIO0_27	PINENABLE0	Table 123	Common input to all comparators
ACMP_I2	AI	external	PIO1_06	PINENABLE0	Table 123	Common input to all comparators
ROSC	O	external	any	PINASSIGN13	Table 120	Ring oscillator output
ROSC_RST	I	external	any	PINASSIGN13	Table 120	Ring oscillator reset
ACMP0_O	O	external	any	PINASSIGN12	Table 119	Comparator 0 output
		internal	SCT0 input mux	SCT0_INMUX	Table 127	
		internal	SCT1 input mux	SCT1_INMUX	Table 128	
		internal	SCT2 input mux	SCT1_INMUX	Table 129	
		internal	SCT3 input mux	SCT1_INMUX	Table 130	
		internal	ADC0 trigger input	SEQA_CTRL, SEQB_CTRL	Table 435, Table 436	
ACMP1_O	O	external	any	PINASSIGN12	Table 119	Comparator1 output
		internal	SCT0 input mux	SCT0_INMUX	Table 127	
		internal	SCT1 input mux	SCT1_INMUX	Table 128	
		internal	SCT2 input mux	SCT1_INMUX	Table 129	
		internal	SCT3 input mux	SCT1_INMUX	Table 130	
		internal	ADC0 trigger input	SEQA_CTRL, SEQB_CTRL	Table 435, Table 436	

Table 456. Analog comparator 0/1/2/3 pin description

Function	Direction	Type	Connect to	Use register	Reference	Description
ACMP2_O	O	external	any	PINASSIGN12	Table 119	Comparator 2 output
		internal	SCT0 input mux	SCT0_INMUX	Table 127	
		internal	SCT1 input mux	SCT1_INMUX	Table 128	
		internal	SCT2 input mux	SCT1_INMUX	Table 129	
		internal	SCT3 input mux	SCT1_INMUX	Table 130	
		internal	ADC0 trigger input	SEQA_CTRL, SEQB_CTRL	Table 435, Table 436	
ACMP3_O	O	external	any	PINASSIGN12	Table 119	Comparator 3 output
		internal	SCT0 input mux	SCT0_INMUX	Table 127	
		internal	SCT1 input mux	SCT1_INMUX	Table 128	
		internal	SCT2 input mux	SCT1_INMUX	Table 129	
		internal	SCT3 input mux	SCT1_INMUX	Table 130	
		internal	ADC0 trigger input	SEQA_CTRL, SEQB_CTRL	Table 435, Table 436	

30.5 General description

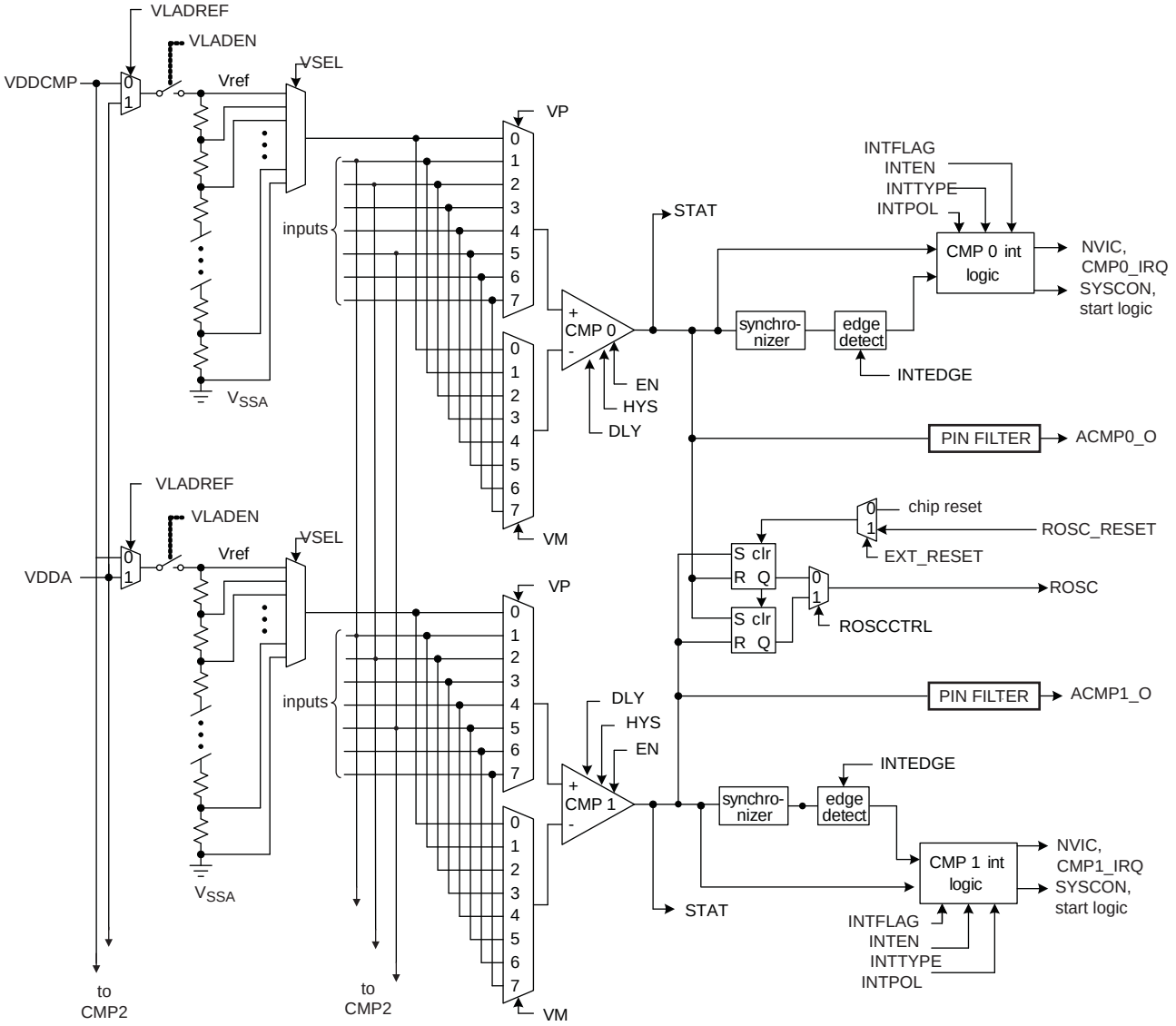


Fig 92. Comparator block diagram

30.5.1 Comparator interrupt configurations

Table 457. Interrupt configurations (CMPn register settings)

INTEN	INTEDGE	INTTYPE	INTPOL	Interrupt function
1	00	0	INTPOL has no effect on the comparator interrupt but can separately select a wake-up polarity.	Falling edge interrupt.
1	01	0	INTPOL has no effect on the comparator interrupt but can separately select a wake-up polarity.	Rising edge interrupt.
1	1x	0	INTPOL has no effect on the comparator interrupt but can separately select a wake-up polarity.	Interrupt on both edges.
1	INTEDGE has no effect on the comparator interrupt.	1	0	High level interrupt. The same signal goes to the start logic.
1	INTEDGE has no effect on the comparator interrupt.	1	1	Low level interrupt. The same signal goes to the start logic.

30.6 Register description

The comparator block contains four independent comparators and one common control register for the ring oscillator (ROSC) control.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

Table 458. Register overview: Analog comparator ACMP (base address 0x4000 8000)

Name	Access	Address offset	Description	Reset value	Reference
CTRL	R/W	0x000	Comparator block control register	0x0	Table 459
CMP0	R/W	0x004	Comparator 0 source control		Table 460
CMPFILTR0	R/W	0x008	Comparator 0 pin filter set-up		Table 464
CMP1	R/W	0x00C	Comparator 1 source control		Table 461
CMPFILTR1	R/W	0x010	Comparator 1 pin filter set-up		Table 459
CMP2	R/W	0x014	Comparator 2 source control		Table 462
CMPFILTR2	R/W	0x018	Comparator 2 pin filter set-up		Table 459
CMP3	R/W	0x01C	Comparator 3 source control		Table 463
CMPFILTR3	R/W	0x020	Comparator 3 pin filter set-up		Table 459

30.6.1 Comparator block control register

This register controls aspects of the comparator block that apply to both comparators.

Table 459. Comparator block control register (CTRL, address 0x4000 8000) bit description

Bit	Symbol	Value	Description	Reset Value
7:0	-		Reserved.	NA
8	ROSCCTL		Selects which comparators set and reset the ROSC output.	0
		0	ACMP1/ACMP0. The ROSC output is set by ACMP1 and reset by ACMP0.	
		1	ACMP0/ACMP1. The ROSC output is set by ACMP0 and reset by ACMP1.	
9	EXT_RESET		Selects the reset source for the ROSC output.	0
		0	Internal. The ROSC output is reset by the internal chip reset.	
		1	From pin ROSC_RESET. The ROSC output is reset by the ROSC_RESET input.	
31:10	-		Reserved.	NA

30.6.2 Comparator 0 register

This register enables and configures many aspects of comparator 0 operation. See Table 457 for details on how to set the output polarity configured by bits INTPOL, INTTYPE, and INTEDGE.

Table 460. Comparator 0 register (CMP0, address 0x4000 8004) bit description

Bit	Symbol	Value	Description	Reset Value
0	EN		Comparator enable control.	00
		0	Disabled. Comparator disabled.	
		1	Enabled. Comparator is enabled.	
1	-		Reserved.	NA
2	INTEN		Interrupt enable.	0
		0	Disabled. Interrupts are disabled..	
		1	Enabled. Interrupts are enabled.. Must set to 1 for interrupts to propagate to the NVIC and start-up logic.	
3	STAT		Comparator status. This bit reflects the comparator output	0
6:4	VM		VM input select.	000
		0x0	Vref divider 0.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP0_I3.	
		0x4	ACMP0_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	Temp sensor.	
		0x7	ADC0_2. Input for ADC0 channel 2.	
7	-		Reserved.	NA

Table 460. Comparator 0 register (CMP0, address 0x4000 8004) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
10:8	VP		VP input select.	000
		0x0	Vref divider 0.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP0_I3.	
		0x4	ACMP0_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	Temp sensor.	
		0x7	ADC0_2. Input for ADC0 channel 2.	
12:11	-		Reserved.	NA
14:13	HYS		Hysteresis control. When enabled, hysteresis determines the difference required between the comparator inputs before the comparator output switches. The difference must be in the direction opposite of the current comparator output.	00
		0x0	Hysteresis is turned off, comparator output will change as the input voltages cross.	
		0x1	Hysteresis = 5 mV.	
		0x2	Hysteresis = 10 mV.	
		0x3	Hysteresis = 15 mV.	
15	INTPOL		Selects the polarity of the CMP output for purposes of generating level interrupts.	0
		0	Not inverted. The output is used as-is for generating interrupts.	
		1	Inverted. The output is used inverted for generating interrupts.	
16	INTTYPE		Select interrupt type.	0
		0	Edge. Comparator interrupt is edge triggered.	
		1	Level. Comparator interrupt is level triggered.	
18:17	INTEDGE		Select edge triggered interrupt to be active on either high or low transitions, when INTTYPE = 0.	0
		0x0	Falling. Comparator interrupt is active on falling edges.	
		0x1	Rising. Comparator interrupt is active on rising edges.	
		0x2	Both edges. Comparator Interrupt is active on both edges.	
		0x3	Reserved.	
19	INTFLAG		Interrupt flag.	0
		0	Not pending. The Comparator interrupt is not pending.	
		1	Pending. The Comparator interrupt is pending. Writing a 1 to this bit clears the flag.	
20	VLADEN		Voltage ladder enable for comparator 0.	00
		0	Disabled. The Comparator voltage ladder is disabled.	
		1	Enabled. The Comparator voltage ladder is enabled.	
21	-		Reserved.	NA

Table 460. Comparator 0 register (CMP0, address 0x4000 8004) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
22	VLADREF		Voltage reference select for comparator 0 voltage ladder.	0
		0	VDDCMP. Voltage level on pin VREFP_DAC_VDDCMP.	
		1	V_{DDA} pin.	
23	-		Reserved.	NA
28:24	VSEL		Voltage ladder value for comparator 0. The reference voltage Vref depends on the setting of VLADREF (either V_{DDA} or voltage on pin VDDCMP). 00000 = Vss. 00001 = 1 × Vref / 31. 00010 = 2 × Vref / 31. ... 11111 = Vref	0x00
30:29	DLY		Configure the propagation delay. A shorter propagation delay means higher power consumption. Use values from 0x0 (shortest propagation delay and highest power consumption) to 0x2 (longest propagation delay and lowest power consumption). See the data sheet for details.	0x0
31	-		Reserved.	NA

30.6.3 Comparator 1 register

This register enables and configures many aspects of comparator 1 operation. See Table 457 for details on how to set the output polarity configured by bits INTPOL, INTTYPE, and INTEDGE.

Table 461. Comparator 1 register (CMP1, address 0x4000 800C) bit description

Bit	Symbol	Value	Description	Reset Value
0	EN		Comparator enable control.	00
		0	Disabled. Comparator disabled.	
		1	Enabled. Comparator is enabled.	
1	-		Reserved.	NA
2	INTEN		Interrupt enable.	0
		0	Disabled. Interrupts are disabled..	
		1	Enabled. Interrupts are enabled.. Must set to 1 for interrupts to propagate to the NVIC and start-up logic.	
3	STAT		Comparator status. This bit reflects the comparator output	0

Table 461. Comparator 1 register (CMP1, address 0x4000 800C) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
6:4	VM		VM input select.	000
		0x0	Vref divider 1.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP1_I3.	
		0x4	ACMP1_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC0_1. Input for ADC0 channel 1.	
		0x7	ADC0_3. Input for ADC0 channel 3.	
7	-		Reserved.	NA
10:8	VP		VP input select.	000
		0x0	Vref divider 1.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP1_I3.	
		0x4	ACMP1_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC0_1. Input for ADC0 channel 1.	
		0x7	ADC0_3. Input for ADC0 channel 3.	
12:11	-		Reserved.	NA
14:13	HYS		Hysteresis control. When enabled, hysteresis determines the difference required between the comparator inputs before the comparator output switches. The difference must be in the direction opposite of the current comparator output.	00
		0x0	Hysteresis is turned off, comparator output will change as the input voltages cross.	
		0x1	Hysteresis = 5 mV.	
		0x2	Hysteresis = 10 mV.	
		0x3	Hysteresis = 15 mV.	
15	INTPOL		Selects the polarity of the CMP output for purposes of generating level interrupts.	0
		0	Not inverted. The output is used as-is for generating interrupts.	
		1	Inverted. The output is used inverted for generating interrupts.	
16	INTTYPE		Select interrupt type.	0
		0	Edge. Comparator interrupt is edge triggered.	
		1	Level. Comparator interrupt is level triggered.	
18:17	INTEDGE		Select edge triggered interrupt to be active on either high or low transitions, when INTTYPE = 0.	0
		0x0	Falling. Comparator interrupt is active on falling edges.	
		0x1	Rising. Comparator interrupt is active on rising edges.	
		0x2	Both edges. Comparator Interrupt is active on both edges.	
		0x3	Reserved.	

Table 461. Comparator 1 register (CMP1, address 0x4000 800C) bit description *...continued*

Bit	Symbol	Value	Description	Reset Value
19	INTFLAG		Interrupt flag.	0
		0	Not pending. The Comparator interrupt is not pending.	
		1	Pending. The Comparator interrupt is pending. Writing a 1 to this bit clears the flag.	
20	VLADEN		Voltage ladder enable for comparator 1.	00
		0	Disabled. The Comparator voltage ladder is disabled.	
		1	Enabled. The Comparator voltage ladder is enabled.	
21	-		Reserved.	NA
22	VLADREF		Voltage reference select for comparator 1 voltage ladder.	0
		0	VDDCMP. Voltage level on pin VREFP_DAC_VDDCMP.	
		1	V_{DDA} pin.	
23	-		Reserved.	NA
28:24	VSEL		Voltage ladder value for comparator 1. The reference voltage Vref depends on the setting of VLADREF (either V_{DDA} or voltage on pin VDDCMP). 00000 = Vss. 00001 = 1 × Vref / 31. 00010 = 2 × Vref / 31. ... 11111 = Vref	0x00
30:29	DLY		Configure the propagation delay. A shorter propagation delay means higher power consumption. Use values from 0x0 (shortest propagation delay and highest power consumption) to 0x2 (longest propagation delay and lowest power consumption). See the data sheet for details.	0x0
31	-		Reserved.	NA

30.6.4 Comparator 2 register

This register enables and configures many aspects of comparator 2 operation. See Table 457 for details on how to set the output polarity configured by bits INTPOL, INTTYPE, and INTEDGE.

Table 462. Comparator 2 register (CMP2, address 0x4000 8014) bit description

Bit	Symbol	Value	Description	Reset Value
0	EN		Comparator enable control.	00
		0	Disabled. Comparator disabled.	
		1	Enabled. Comparator is enabled.	
1	-		Reserved.	NA
2	INTEN		Interrupt enable.	0
		0	Disabled. Interrupts are disabled..	
		1	Enabled. Interrupts are enabled.. Must set to 1 for interrupts to propagate to the NVIC and start-up logic.	
3	STAT		Comparator status. This bit reflects the comparator output	0

Table 462. Comparator 2 register (CMP2, address 0x4000 8014) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
6:4	VM		VM input select.	000
		0x0	Vref divider 2.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP2_I3.	
		0x4	ACMP2_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC0_0. Input for ADC0 channel 0.	
		0x7	ADC1_2. Input for ADC1 channel 2.	
7	-		Reserved.	NA
10:8	VP		VP input select.	000
		0x0	Vref divider 2.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP2_I3.	
		0x4	ACMP2_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC0_0. Input for ADC0 channel 0.	
		0x7	ADC1_2. Input for ADC1 channel 2.	
12:11	-		Reserved.	NA
14:13	HYS		Hysteresis control. When enabled, hysteresis determines the difference required between the comparator inputs before the comparator output switches. The difference must be in the direction opposite of the current comparator output.	00
		0x0	Hysteresis is turned off, comparator output will change as the input voltages cross.	
		0x1	Hysteresis = 5 mV.	
		0x2	Hysteresis = 10 mV.	
		0x3	Hysteresis = 15 mV.	
15	INTPOL		Selects the polarity of the CMP output for purposes of generating level interrupts.	0
		0	Not inverted. The output is used as-is for generating interrupts.	
		1	Inverted. The output is used inverted for generating interrupts.	
16	INTTYPE		Select interrupt type.	0
		0	Edge. Comparator interrupt is edge triggered.	
		1	Level. Comparator interrupt is level triggered.	
18:17	INTEDGE		Select edge triggered interrupt to be active on either high or low transitions, when INTTYPE = 0.	0
		0x0	Falling. Comparator interrupt is active on falling edges.	
		0x1	Rising. Comparator interrupt is active on rising edges.	
		0x2	Both edges. Comparator Interrupt is active on both edges.	
		0x3	Reserved.	

Table 462. Comparator 2 register (CMP2, address 0x4000 8014) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
19	INTFLAG		Interrupt flag.	0
		0	Not pending. The Comparator interrupt is not pending.	
		1	Pending. The Comparator interrupt is pending. Writing a 1 to this bit clears the flag.	
20	VLADEN		Voltage ladder enable for comparator 2.	00
		0	Disabled. The Comparator voltage ladder is disabled.	
		1	Enabled. The Comparator voltage ladder is enabled.	
21	-		Reserved.	NA
22	VLADREF		Voltage reference select for comparator 2 voltage ladder.	0
		0	VDDCMP. Voltage level on pin VREFP_DAC_VDDCMP.	
		1	V_{DDA} pin.	
23	-		Reserved.	NA
28:24	VSEL		Voltage ladder value for comparator 2. The reference voltage Vref depends on the setting of VLADREF (either V_{DDA} or voltage on pin VDDCMP). 00000 = Vss. 00001 = 1 × Vref / 31. 00010 = 2 × Vref / 31. ... 11111 = Vref	0x00
30:29	DLY		Configure the propagation delay. A shorter propagation delay means higher power consumption. Use values from 0x0 (shortest propagation delay and highest power consumption) to 0x2 (longest propagation delay and lowest power consumption). See the data sheet for details.	0x0
31	-		Reserved.	NA

30.6.5 Comparator 3 register

This register enables and configures many aspects of comparator 3 operation. See Table 457 for details on how to set the output polarity configured by bits INTPOL, INTTYPE, and INTEDGE.

Table 463. Comparator 3 register (CMP3, address 0x4000 801C) bit description

Bit	Symbol	Value	Description	Reset Value
0	EN		Comparator enable control.	00
		0	Disabled. Comparator disabled.	
		1	Enabled. Comparator is enabled.	
1	-		Reserved.	NA
2	INTEN		Interrupt enable.	0
		0	Disabled. Interrupts are disabled..	
		1	Enabled. Interrupts are enabled.. Must set to 1 for interrupts to propagate to the NVIC and start-up logic.	
3	STAT		Comparator status. This bit reflects the comparator output	0

Table 463. Comparator 3 register (CMP3, address 0x4000 801C) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
6:4	VM		VM input select.	000
		0x0	Vref divider 3.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP3_I3.	
		0x4	ACMP3_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC1_1. Input for ADC1 channel 1.	
		0x7	ADC1_3. Input for ADC1 channel 3.	
7	-		Reserved.	NA
10:8	VP		VP input select.	000
		0x0	Vref divider 3.	
		0x1	ACMP_I1.	
		0x2	ACMP_I2.	
		0x3	ACMP3_I3.	
		0x4	ACMP3_I4.	
		0x5	Internal 0.9 V band gap reference.	
		0x6	ADC1_1. Input for ADC1 channel 1.	
		0x7	ADC1_3. Input for ADC1 channel 3.	
12:11	-		Reserved.	NA
14:13	HYS		Hysteresis control. When enabled, hysteresis determines the difference required between the comparator inputs before the comparator output switches. The difference must be in the direction opposite of the current comparator output.	00
		0x0	Hysteresis is turned off, comparator output will change as the input voltages cross.	
		0x1	Hysteresis = 5 mV.	
		0x2	Hysteresis = 10 mV.	
		0x3	Hysteresis = 15 mV.	
15	INTPOL		Selects the polarity of the CMP output for purposes of generating level interrupts.	0
		0	Not inverted. The output is used as-is for generating interrupts.	
		1	Inverted. The output is used inverted for generating interrupts.	
16	INTTYPE		Select interrupt type.	0
		0	Edge. Comparator interrupt is edge triggered.	
		1	Level. Comparator interrupt is level triggered.	
18:17	INTEDGE		Select edge triggered interrupt to be active on either high or low transitions, when INTTYPE = 0.	0
		0x0	Falling. Comparator interrupt is active on falling edges.	
		0x1	Rising. Comparator interrupt is active on rising edges.	
		0x2	Both edges. Comparator Interrupt is active on both edges.	
		0x3	Reserved.	

Table 463. Comparator 3 register (CMP3, address 0x4000 801C) bit description *…continued*

Bit	Symbol	Value	Description	Reset Value
19	INTFLAG		Interrupt flag.	0
		0	Not pending. The Comparator interrupt is not pending.	
		1	Pending. The Comparator interrupt is pending. Writing a 1 to this bit clears the flag.	
20	VLADEN		Voltage ladder enable for comparator 3.	00
		0	Disabled. The Comparator voltage ladder is disabled.	
		1	Enabled. The Comparator voltage ladder is enabled.	
21	-		Reserved.	NA
22	VLADREF		Voltage reference select for comparator 3 voltage ladder.	0
		0	VDDCMP. Voltage level on pin VREFP_DAC_VDDCMP.	
		1	V_{DDA} pin.	
23	-		Reserved.	NA
28:24	VSEL		Voltage ladder value for comparator 3. The reference voltage Vref depends on the setting of VLADREF (either V_{DDA} or voltage on pin VDDCMP). 00000 = Vss. 00001 = 1 × Vref / 31. 00010 = 2 × Vref / 31. ... 11111 = Vref	0x00
30:29	DLY		Configure the propagation delay. A shorter propagation delay means higher power consumption. Use values from 0x0 (shortest propagation delay and highest power consumption) to 0x2 (longest propagation delay and lowest power consumption). See the data sheet for details.	0x0
31	-		Reserved.	NA

30.6.6 Comparator pin filter 0 to 3 registers

Table 464. Comparator pin filter registers 0 to 3 (CMPFILTR[0:3], address 0x4000 8008 (CMPFILTR0) to 0x4000 8020 (CMPFILTR3)) bit description

Bit	Symbol	Value	Description	Reset value
1:0	S_MODE		Digital filter sample mode.	0
		0x0	Bypass input filter.	
		0x1	1 clock cycle. Input pulses shorter than one filter clock are rejected.	
		0x2	2 clock cycles. Input pulses shorter than two filter clocks are rejected.	
		0x3	3 clock cycles. Input pulses shorter than three filter clocks are rejected.	

Table 464. Comparator pin filter registers 0 to 3 (CMPFILTR[0:3], address 0x4000 8008 (CMPFILTR0) to 0x4000 8020 (CMPFILTR3)) bit description

Bit	Symbol	Value	Description	Reset value
4:2	CLKDIV		Select clock divider for comparator clock CMP_PCLK.	0
		0x0	CMP_PCLK.	
		0x1	CMP_PCLK/2.	
		0x2	CMP_PCLK/4.	
		0x3	CMP_PCLK/8.	
		0x4	CMP_PCLK/16.	
		0x5	CMP_PCLK/32.	
		0x6	CMP_PCLK/64.	
31:5	-		Reserved	-

31.1 How to read this chapter

The temperature sensor is available on all parts.

31.2 Features

- Linear temperature sensor
- Sensor output internally connected to the ADCs and the analog comparator 0 for temperature monitoring

31.3 Basic configuration

- Enable the power to the temperature sensor by setting the TS_PD bit in the PDRUNCFG register. See Table 75.

  Remark: The internal reference voltage must be powered before the temperature sensor can be turned on. See bit IREF_PD in the PDRUNCFG register (Table 75).
- To monitor the temperature continually, select the temperature sensor as source for channel 0 of either ADC0 or ADC1. See Table 434. The digital temperature reading is available after an analog-to-digital conversion.
- To use the temperature sensor as a trip sensor, select the temperature sensor voltage as input to the analog comparator 0 ACMP0. See Table 460.

31.3.1 Perform a single ADC conversion with the temperature sensor as ADC input

To perform a single ADC conversion for ADC0 channel 0 using the temperature sensor output:

1. Enable the temperature sensor output as input to channel 0. See Table 434.
2. Configure the system clock to be 50 MHz and select a CLKDIV value of 0 for a sampling rate of 50 MHz using the ADC CTRL register.
3. Select the synchronous mode in the CTRL register.
4. Select ADC channel 0 to perform the conversion by setting the CHANNELS bits to 0x1 in the SEQA_CTL register.
5. Set the START bit to 1 in the SEQA_CTRL register.
6. Read the RESULT bits in the DAT0 register for the conversion result.

31.4 Pin description

The temperature sensor has no configurable pins.

31.5 Register description

The temperature sensor has no configurable registers.

32.1 How to read this chapter

The CRC engine is available on all LPC15xx parts.

32.2 Features

- Supports three common polynomials CRC-CCITT, CRC-16, and CRC-32.
  - CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$
  - CRC-16: $x^{16} + x^{15} + x^2 + 1$
  - CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Bit order reverse and 1's complement programmable setting for input data and CRC sum.
- Programmable seed number setting.
- Accept any size of data width per write: 8, 16 or 32-bit.
  - 8-bit write: 1-cycle operation
  - 16-bit write: 2-cycle operation (8-bit x 2-cycle)
  - 32-bit write: 4-cycle operation (8-bit x 4-cycle)

32.3 Basic configuration

Enable the clock to the CRC engine in the SYSAHBCLKCTRL0 register (Table 50, bit 21).

32.4 Pin description

The CRC engine has no configurable pins.

32.5 General description

The Cyclic Redundancy Check (CRC) generator with programmable polynomial settings supports several CRC standards commonly used.

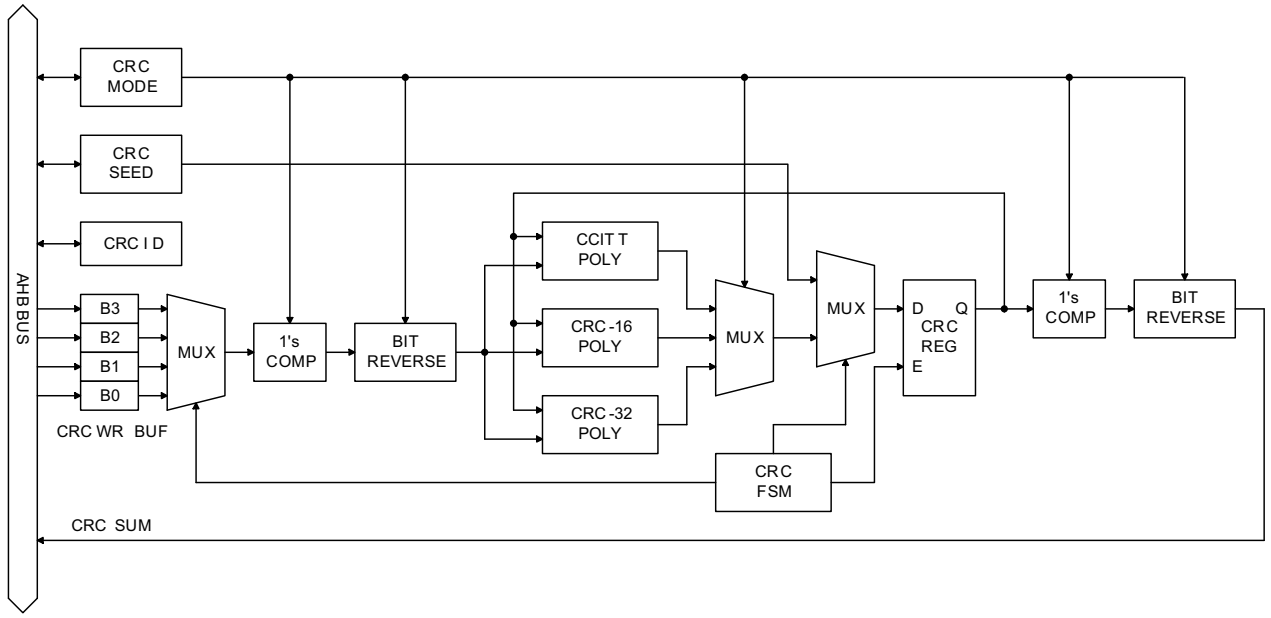


Fig 93. CRC block diagram

32.6 Register description

Table 465. Register overview: CRC engine (base address 0x1C01 0000)

Name	Access	Address offset	Description	Reset value	Reference
MODE	R/W	0x000	CRC mode register	0x0000 0000	Table 466
SEED	R/W	0x004	CRC seed register	0x0000 FFFF	Table 467
SUM	RO	0x008	CRC checksum register	0x0000 FFFF	Table 468
WR_DATA	WO	0x008	CRC data register	-	Table 469

32.6.1 CRC mode register

Table 466. CRC mode register (MODE, address 0x1C01 0000) bit description

Bit	Symbol	Description	Reset value
1:0	CRC_POLY	CRC polynom: 1X= CRC-32 polynomial 01= CRC-16 polynomial 00= CRC-CCITT polynomial	00
2	BIT_RVS_WR	Data bit order: 1= Bit order reverse for CRC_WR_DATA (per byte) 0= No bit order reverse for CRC_WR_DATA (per byte)	0
3	CMPL_WR	Data complement: 1= 1's complement for CRC_WR_DATA 0= No 1's complement for CRC_WR_DATA	0
4	BIT_RVS_SUM	CRC sum bit order: 1= Bit order reverse for CRC_SUM 0= No bit order reverse for CRC_SUM	0
5	CMPL_SUM	CRC sum complement: 1= 1's complement for CRC_SUM 0=No 1's complement for CRC_SUM	0
31:6	Reserved	Always 0 when read	0x0000000

32.6.2 CRC seed register

Table 467. CRC seed register (SEED, address 0x1C01 0004) bit description

Bit	Symbol	Description	Reset value
31:0	CRC_SEED	A write access to this register will load CRC seed value to CRC_SUM register with selected bit order and 1's complement pre-processes. **Remark:** A write access to this register will overrule the CRC calculation in progresses.	0x0000 FFFF

32.6.3 CRC checksum register

This register is a Read-only register containing the most recent checksum. The read request to this register is automatically delayed by a finite number of wait states until the results are valid and the checksum computation is complete.

Table 468. CRC checksum register (SUM, address 0x1C01 0008) bit description

Bit	Symbol	Description	Reset value
31:0	CRC_SUM	The most recent CRC sum can be read through this register with selected bit order and 1's complement post-processes.	0x0000 FFFF

32.6.4 CRC data register

This register is a Write-only register containing the data block for which the CRC sum will be calculated.

Table 469. CRC data register (WR_DATA, address 0x1C01 0008) bit description

Bit	Symbol	Description	Reset value
31:0	CRC_WR_DATA	Data written to this register will be taken to perform CRC calculation with selected bit order and 1's complement pre-process. Any write size 8, 16 or 32-bit are allowed and accept back-to-back transactions.	-

32.7 Functional description

The following sections describe the register settings for each supported CRC standard:

32.7.1 CRC-CCITT set-up

Polynomial = $x^{16} + x^{12} + x^5 + 1$

Seed Value = 0xFFFF

Bit order reverse for data input: NO

1's complement for data input: NO

Bit order reverse for CRC sum: NO

1's complement for CRC sum: NO

CRC_MODE = 0x0000 0000

CRC_SEED = 0x0000 FFFF

32.7.2 CRC-16 set-up

Polynomial = $x^{16} + x^{15} + x^2 + 1$

Seed Value = 0x0000

Bit order reverse for data input: YES

1's complement for data input: NO

Bit order reverse for CRC sum: YES

1's complement for CRC sum: NO

CRC_MODE = 0x0000 0015

CRC_SEED = 0x0000 0000

32.7.3 CRC-32 set-up

Polynomial = $x^{32}+ x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Seed Value = 0xFFFF FFFF

Bit order reverse for data input: YES

1's complement for data input: NO

Bit order reverse for CRC sum: YES

1's complement for CRC sum: YES

CRC_MODE = 0x0000 0036

CRC_SEED = 0xFFFF FFFF

33.1 How to read this chapter

The flash controller is identical on all LPC15xx parts.

33.2 Features

Provides registers for flash signature generation.

33.3 General description

The flash controller is accessible for generating the flash signature.

33.4 Register description

Remark: To configure flash access times, use the FLASHCFG register in the SYSCON block. See Table 60.

Table 470. Register overview: FMC (base address 0x400B C000)

Name	Access	Address offset	Description	Reset value	Reference
FMSSTART	R/W	0x020	Signature start address register	0	Table 471
FMSSTOP	R/W	0x024	Signature stop-address register	0	Table 472
FMSW0	R	0x02C	Signature word	-	Table 473

33.4.1 Flash signature start address register

Table 471. Flash Module Signature Start register (FMSSTART, 0x400B 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

33.4.2 Flash signature stop address register

Table 472. Flash Module Signature Stop register (FMSSTOP, 0x400B C024) bit description

Bit	Symbol	Value	Description	Reset value
16:0	STOPA		Stop address for signature generation (the word specified by STOPA is included in the address range). The address is in units of memory words, not bytes.	0
30:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
31	STRTBIST		When this bit is written to 1, signature generation starts. At the end of signature generation, this bit is automatically cleared.	0

33.4.3 Flash signature generation result register

The signature generation result register returns the flash signature produced by the embedded signature generator.

The generated flash signature can be used to verify the flash memory contents. The generated signature can be compared with an expected signature and thus makes saves time and code space. The method for generating the signature is described in Section 33.5.1.

Table 473. FMSW0 register bit description (FMSW0, address: 0x400B C02C)

Bit	Symbol	Description	Reset value
31:0	SIG	32-bit signature.	-

33.5 Functional description

33.5.1 Flash signature generation

The flash module contains a built-in signature generator. This generator can produce a 32-bit signature from a range of flash memory. A typical usage is to verify the flashed contents against a calculated signature (e.g. during programming).

The address range for generating a signature must be aligned on flash-word boundaries, i.e. 32-bit boundaries. Once started, signature generation completes independently. While signature generation is in progress, the flash memory cannot be accessed for other purposes, and an attempted read will cause a wait state to be asserted until signature generation is complete. Code outside of the flash (e.g. internal RAM) can be executed during signature generation. This can include interrupt services, if the interrupt vector table is re-mapped to memory other than the flash memory. The code that initiates signature generation should also be placed outside of the flash memory.

33.5.1.1 Signature generation address and control registers

These registers control automatic signature generation. A signature can be generated for any part of the flash memory contents. The address range to be used for generation is defined by writing the start address to the signature start address register (FMSSTART) and the stop address to the signature stop address register (FMSSTOP. The start and stop addresses must be aligned to 32-bit boundaries.

Signature generation is started by setting the STRTBIST bit in the FMSSTOP register. Setting the STRTBIST bit is typically combined with the signature stop address in a single write.

Table 471 and Table 472 show the bit assignments in the FMSSTART and FMSSTOP registers respectively.

33.5.1.2 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 32-bit signature can be read from the FMSW0 register. The 32-bit signature reflects the corrected data read from the flash and the flash parity bits and check bit values.

33.5.1.3 Content verification

The signature as it is read from the FMSW0 register must be equal to the reference signature. The following pseudo-code shows the algorithm to derive the reference signature:

```
sign = 0
FOR address = FMSSTART.START to FMSSTOP.STOPA
{
        FOR i = 0 TO 30
        {
                nextSign[i] = f_Q[address][i] XOR sign[i + 1]
        }
        nextSign[31] = f_Q[address][31] XOR sign[0] XOR sign[10] XOR sign[30] XOR sign[31]
        sign = nextSign
```

```
}
signature32 = sign
```

34.1 How to read this chapter

See Table 474 for different flash configurations.

Table 474. LPC15xx flash configurations

Type number	Flash	EEPROM	ISP via UART	ISP via USB	ISP via CAN
LPC1549	256 kB	4 kB	yes	yes	yes
LPC1548	128 kB	4 kB	yes	yes	yes
LPC1547	64 kB	4 kB	yes	yes	yes
LPC1519	256 kB	4 kB	yes	no	yes
LPC1518	128 kB	4 kB	yes	no	yes
LPC1517	64 kB	4 kB	yes	no	yes

Remark: In addition to the ISP and IAP commands, a register can be accessed in the SYSCON block to configure flash memory access times, see Section 3.6.32.

34.2 Features

- In-System Programming: In-System programming (ISP) is programming or reprogramming the on-chip flash memory, using the boot loader software and the UART serial port. This can be done when the part resides in the end-user board.
- In Application Programming: In-Application (IAP) programming is performing erase and write operation on the on-chip flash memory, as directed by the end-user application code.
- Small size (256 Byte) page erase programming.
- Flash access times can be configured through a register in the flash controller block.

34.3 General description

34.3.1 Boot loader

For the boot loader operation and boot pins, see Chapter 5 “LPC15xx Boot process”.

The boot loader version can be read by ISP/IAP calls (see Section 34.6.12 or Section 34.8.6).

34.3.2 Memory map after any reset

The boot block is 32 kB in size and is located in the memory region starting from the address 0x0300 0000. The boot loader 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 Section 34.3.6. The interrupt vectors residing in the boot block of the on-chip flash memory also become active after reset, i.e., the bottom 512 bytes of the boot block are also visible in the memory region starting from the address 0x0000 0000.

34.3.3 Flash content protection mechanism

The LPC15xx is equipped with the Error Correction Code (ECC) capable Flash memory. The purpose of an error correction module is twofold. Firstly, it decodes data words read from the memory into output data words. Secondly, it encodes data words to be written to the memory. The error correction capability consists of single bit error correction with Hamming code.

The operation of ECC is transparent to the running application. The ECC content itself is stored in a flash memory not accessible by user's code to either read from it or write into it on its own. A byte of ECC corresponds to every consecutive 128 bits of the user accessible Flash. Consequently, Flash bytes from 0x0000 0000 to 0x0000 000F are protected by the first ECC byte, Flash bytes from 0x0000 0010 to 0x0000 001F are protected by the second ECC byte, etc.

Whenever the CPU requests a read from user's Flash, both 128 bits of raw data containing the specified memory location and the matching ECC byte are evaluated. If the ECC mechanism detects a single error in the fetched data, a correction will be applied before data are provided to the CPU. When a write request into the user's Flash is made, write of user specified content is accompanied by a matching ECC value calculated and stored in the ECC memory.

When a sector of Flash memory is erased, the corresponding ECC bytes are also erased. Once an ECC byte is written, it can not be updated unless it is erased first. Therefore, for the implemented ECC mechanism to perform properly, data must be written into the flash memory in groups of 16 bytes (or multiples of 16), aligned as described above.

34.3.4 Flash partitions

Some IAP and ISP commands operate on sectors and specify sector numbers. In addition, a page erase command is available. The size of a sector is 4 kB and the size of a page is 256 Byte. One sector contains 16 pages. Sector 0 and page 0 are located at address 0x0000 0000.

Table 475. LPC15xx flash sectors

Sector number	Sector size [kB]	Page number	Address range	LPC1547/17	LPC1548/18	LPC1549/19
0 - 15	4	0 -255	0x0000 0000 - 0x0000 FFFF	yes	yes	yes
16 - 31	4	256 - 511	0x0001 0000 - 0x0001 FFFF	no	yes	yes
32 - 47	4	512 - 767	0x0002 0000 - 0x0002 FFFF	no	no	yes
48 - 63	4	768 - 1023	0x0003 0000 - 0x0003 FFFF	no	no	yes

34.3.5 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 476. Code Read Protection (CRP) options

Name	Pattern programmed in 0x0000 02FC	Description
NO_ISP	0x4E69 7370	Prevents sampling of pins for entering ISP mode. ISP sampling pins are available for other applications.
CRP1	0x12345678	Access to chip via the SWD pins is disabled. This mode allows partial flash update using the following ISP commands and restrictions: • Write to RAM command cannot access RAM below 0x0200 0300. • Copy RAM to flash command can not write to Sector 0. • Erase command can erase Sector 0 only when all sectors are selected for erase. • Compare command is disabled. • Read Memory command is disabled. This mode is useful when CRP is required and flash field updates are needed but all sectors can not be erased. Since compare command is disabled in case of partial updates the secondary loader should implement checksum mechanism to verify the integrity of the flash.
CRP2	0x87654321	Access to chip via the SWD pins is disabled. The following ISP commands are disabled: • Read Memory • Write to RAM • Go • Copy RAM to flash • Compare When CRP2 is enabled the ISP erase command only allows erasure of all user sectors.
CRP3	0x43218765	Access to chip via the SWD pins is disabled. ISP entry selected via the ISP entry pins is disabled if a valid user code is present in flash sector 0. This mode effectively disables ISP override using the pins. It is up to the user's application to provide a flash update mechanism using IAP calls or call reinvoke ISP command to enable flash update via UART0. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Table 477. ISP commands allowed for different CRP levels

ISP command	CRP1	CRP2	CRP3 (no entry in ISP mode allowed)
Unlock	yes	yes	n/a
Set Baud Rate	yes	yes	n/a
Echo	yes	yes	n/a

Table 477. ISP commands allowed for different CRP levels

ISP command	CRP1	CRP2	CRP3 (no entry in ISP mode allowed)
Write to RAM	yes; above 0x0200 0300 only	no	n/a
Read Memory	no	no	n/a
Prepare sectors 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 sectors	yes; sector 0 can only be erased when all sectors are erased.	yes; all sectors only	n/a
Blank check sectors	no	no	n/a
Read Part ID	yes	yes	n/a
Read Boot code version	yes	yes	n/a
Compare	no	no	n/a
ReadUID	yes	yes	n/a

In case a CRP mode is enabled and access to the chip is allowed via the ISP, an unsupported or restricted ISP command will be terminated with return code CODE_READ_PROTECTION_ENABLED.

34.3.5.1 ISP entry protection

In addition to the three CRP modes, the user can prevent the sampling of the ISP sampling pins (see Table 90) for entering ISP mode and thereby release the pins for other applications. This is called the NO_ISP mode. The NO_ISP mode can be entered by programming the pattern 0x4E69 7370 at location 0x0000 02FC.

The NO_ISP mode is identical to the CRP3 mode except for SWD access, which is allowed in NO_ISP mode but disabled in CRP3 mode. The NO_ISP mode does not offer any code protection.

34.3.6 ISP interrupt and SRAM use

34.3.6.1 Interrupts during ISP

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

34.3.6.2 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.

34.3.6.3 RAM used by ISP command handler

USB and C_CAN ISP commands use a fixed on-chip RAM location from 0x0200 0100 to 0x0200 09E4. The user could use this area, but the contents may be lost upon reset. Flash programming commands use the top 32 bytes of on-chip RAM. The stack is located at RAM top − 32 bytes. The maximum stack usage is 256 bytes and grows downwards. Memory for the UART ISP commands is allocated dynamically.

34.3.6.4 RAM used by IAP command handler

Flash programming commands use the top 32 bytes of on-chip RAM. The maximum stack usage in the user allocated stack space is 128 bytes and grows downwards.

34.4 UART communication protocol

All UART 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 plain binary format.

If the UART is selected, the host should send a '?' (0x3F) as a synchronization character and wait for a response. The host side serial port settings should be 8 data bits, 1 stop bit and no parity. The auto-baud routine measures the bit time of the received synchronization character in terms of its own frequency and programs the baud rate generator of the serial port. It also sends an ASCII string ("Synchronized<CR><LF>") to the host. In response to this, the host should send back 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. The response is required for backward compatibility of the boot loader code and is ignored. "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 clock frequency should be greater than or equal to 10 MHz. In USART ISP mode, the part is clocked by the IRC and the crystal frequency is ignored.

Once the crystal frequency is received the part is initialized and the ISP command handler is invoked. For safety reasons an "Unlock" command is required before executing the commands resulting in flash erase/write operations and the "Go" command. The rest of the commands can be executed without the unlock command. The Unlock command is required to be executed once per ISP session. The Unlock command is explained in Section 34.6 “UART ISP commands” on page 556.

34.4.1 UART ISP command format

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

34.4.2 UART 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).

34.4.3 UART ISP data format

The data stream is in plain binary format.

34.5 USB communication protocol

The LPC154x parts are enumerated as a Mass Storage Class (MSC) device to a PC or another embedded system. In order to connect via the USB interface, the part must use an external crystal at a frequency of 12 MHz. The MSC device presents an easy integration with the PC's operating system. The flash memory space is represented as a drive in the host file system. The entire available user flash is mapped to a file of the size of the LPC154x flash in the host's folder with the default name firmware.bin. The firmware.bin file can be deleted and a new file can be copied into the directory, thereby updating the user code in flash. Note that the filename of the new flash image file is not important. After a reset or a power cycle, the new file is visible in the host's file system under it's default name firmware.bin.

The code read protection (CRP, see Table 478) level determines how the flash is reprogrammed:

If CRP1 or CRP2 is enabled, the user flash is erased when the file is deleted.

If CRP1 is enabled or no CRP is selected, the user flash is erased and reprogrammed when the new file is copied. However, only the area occupied by the new file is erased and reprogrammed.

Remark: The only commands supported for the LPC154x flash image folder are copy and delete.

Three Code Read Protection (CRP) levels can be enabled for flash images updated through USB (see Section 34.3.5 for details). The volume label on the MSCD indicates the CRP status.

Table 478. CRP levels for USB boot images

CRP status	Volume label	Description
No CRP	CRP DISABLD	The user flash can be read or written.
CRP1	CRP1 ENABLD	The user flash content cannot be read but can be updated. The flash memory sectors are updated depending on the new firmware image.
CRP2	CRP2 ENABLD	The user flash content cannot be read but can be updated. The entire user flash memory is erased before writing the new firmware image.
CRP3	CRP3 ENABLD	The user flash content cannot be read or updated. The boot loader always executes the user application if valid.

34.5.1 Usage note

When programming flash images via Flash Magic or Serial Wire Debugger (SWD), the user code valid signature is automatically inserted by the programming utility. When using USB ISP, the user code valid signature must be either part of the vector table, or the axf or binary file must be post-processed to insert the checksum.

34.6 UART ISP commands

The following commands are accepted by the ISP command handler. Detailed status codes are supported for each command. The command handler sends the return code INVALID_COMMAND when an undefined command is received. Commands and return codes are in ASCII format.

CMD_SUCCESS is sent by ISP command handler only when received ISP command has been completely executed and the new ISP command can be given by the host. Exceptions from this rule are "Set Baud Rate", "Write to RAM", "Read Memory", and "Go" commands.

Table 479. UART ISP command summary

ISP Command	Usage	Described in
Unlock	U <Unlock Code>	Table 480
Set Baud Rate	B <Baud Rate> <stop bit>	Table 481
Echo	A <setting>	Table 482
Write to RAM	W <start address> <number of bytes>	Table 483
Read Memory	R <address> <number of bytes>	Table 484
Prepare sectors for write operation	P <start sector number> <end sector number>	Table 485
Copy RAM to flash	C <Flash address> <RAM address> <number of bytes>	Table 486
Go	G <address> <Mode>	Table 487
Erase sectors	E <start sector number> <end sector number>	Table 488
Blank check sectors	I <start sector number> <end sector number>	Table 489
Read Part ID	J	Table 490
Read Boot code version	K	Table 492
Compare	M <address1> <address2> <number of bytes>	Table 493
ReadUID	N	Table 494
Read CRC checksum	S <address> <number of bytes>	Table 495

34.6.1 Unlock

Table 480. UART ISP Unlock command

Command	U
Input	Unlock code: 23130_{10}

Table 480. UART ISP Unlock command

Command	U
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.

34.6.2 Set Baud Rate

Table 481. UART ISP Set Baud Rate command

Command	B
Input	Baud Rate: 9600 | 19200 | 38400 | 57600 | 115200 Stop bit: 1 | 2
Return Code	CMD_SUCCESS | INVALID_BAUD_RATE | INVALID_STOP_BIT | PARAM_ERROR
Description	This command is used to change the baud rate. The new baud rate is effective after the command handler sends the CMD_SUCCESS return code.
Example	"B 57600 1<CR><LF>" sets the serial port to baud rate 57600 bps and 1 stop bit.

34.6.3 Echo

Table 482. UART 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.

34.6.4 Write to RAM

The host should send the plain binary code after receiving the CMD_SUCCESS return code. This ISP command handler responds with “OK<CR><LF>” when the transfer has finished.

Table 483. UART 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. This command is blocked when code read protection levels 2 or 3 are enabled. Writing
Example	"W 33555200 4<CR><LF>" writes 4 bytes of data to address 0x0200 0300.

34.6.5 Read Memory

Reads the plain binary code of the data stream, followed by the CMD_SUCCESS return code.

Table 484. UART 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 (plain binary)> | 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 33554432 4<CR><LF>" reads 4 bytes of data from address 0x0200 0000.

34.6.6 Prepare sectors for write operation

This command makes flash write/erase operation a two step process.

Remark: Do not read the flash memory while the write operation is ongoing.

Table 485. UART ISP Prepare sectors for write operation command

Command	P
Input	**Start Sector Number** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR | PARAM_ERROR
Description	This command must be executed before executing "Copy RAM to flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot block can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.
Example	"P 0 0<CR><LF>" prepares the flash sector 0.

34.6.7 Copy RAM to flash

When writing to the flash, the following limitations apply:

1. The smallest amount of data that can be written to flash by the copy RAM to flash command is 256 byte (equal to one page).
2. One page consists of 16 flash words (lines), and the smallest amount that can be modified per flash write is one flash word (one line). This limitation exists because ECC is applied during the flash write operation, see Section 34.3.3.
3. To avoid write disturbance (a mechanism intrinsic to flash memories), an erase should be performed after 16 consecutive writes inside the same page. Note that the erase operation then erases the entire sector.

   Remark: Once a page has been written to 16 times, it is still possible to write to other pages within the same sector without performing a sector erase (assuming that those pages have been erased previously).

   Remark: Do not read the flash memory while the write operation is ongoing.

Table 486. UART ISP Copy command

Command	C
Input	**Flash Address(DST):** Destination flash address where data bytes are to be written. The destination address should be a 256 byte boundary. **RAM Address(SRC):** Source RAM address from where data bytes are to be read. **Number of Bytes:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096.
Return Code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not on word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR WRITE_OPERATION | BUSY | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to program the flash memory. The "Prepare Sector(s) for Write Operation" command should precede this command. The affected sectors are automatically protected again once the copy command is successfully executed. The boot block cannot be written by this command. This command is blocked when code read protection is enabled. Also see Section 34.3.3 for the number of bytes that can be written.
Example	"C 0 33556480 512<CR><LF>" copies 512 bytes from the RAM address 0x0200 0800 to the flash address 0.

34.6.8 Go

Table 487. UART ISP Go command

Command	G
Input	**Address:** Flash or RAM address from which the code execution is to be started. This address should be on a word boundary. **Mode:** T (Execute program in Thumb Mode) | A (Execute program in ARM mode).
Return Code	CMD_SUCCESS | ADDR_ERROR | ADDR_NOT_MAPPED | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to execute a program residing in RAM or flash memory. It may not be possible to return to the ISP command handler once this command is successfully executed. This command is blocked when code read protection is enabled.
Example	"G 0 A<CR><LF>" branches to address 0x0000 0000 in ARM mode.

34.6.9 Erase sectors

Table 488. UART ISP Erase sector command

Command	E
Input	**Start Sector Number** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to erase one or more sector(s) of on-chip flash memory. The boot block can not be erased using this command. This command only allows erasure of all user sectors when the code read protection is enabled.
Example	"E 2 3<CR><LF>" erases the flash sectors 2 and 3.

34.6.10 Blank check sectors

Table 489. UART ISP Blank check sector command

Command	I
Input	**Start Sector Number:** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | SECTOR_NOT_BLANK (followed by <Offset of the first non blank word location> <Contents of non blank word location>) | INVALID_SECTOR | PARAM_ERROR
Description	This command is used to blank check one or more sectors of on-chip flash memory. **Blank check on sector 0 always fails as first 64 bytes are re-mapped to flash boot block.** When CRP is enabled, the blank check command returns 0 for the offset and value of sectors which are not blank. Blank sectors are correctly reported irrespective of the CRP setting.
Example	"I 2 3<CR><LF>" blank checks the flash sectors 2 and 3.

34.6.11 Read Part Identification number

Table 490. UART ISP Read Part Identification command

Command	J
Input	None.
Return Code	CMD_SUCCESS followed by part identification number (see Table 491 “LPCA15xx device identification numbers”).
Description	This command is used to read the part identification number.

Table 491. LPCA15xx device identification numbers

Device	Hex coding
LPC1549	0x0000 1549
LPC1548	0x0000 1548
LPC1547	0x0000 1547
LPC1519	0x0000 1519
LPC1518	0x0000 1518
LPC1517	0x0000 1517

34.6.12 Read Boot code version number

Table 492. UART ISP Read Boot Code version number command

Command	K
Input	None
Return Code	CMD_SUCCESS followed by 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)>.
Description	This command is used to read the boot code version number.

34.6.13 Compare

Table 493. UART 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 re-mapped to boot ROM**
Example	"M 8192 33587200 4<CR><LF>" compares 4 bytes from the RAM address 0x0200 8000 to the 4 bytes from the flash address 0x2000.

34.6.14 ReadUID

Table 494. UART ReadUID command

Command	N
Input	None
Return Code	CMD_SUCCESS followed by four 32-bit words of a unique serial number in ASCII format. The word sent at the lowest address is sent first.
Description	This command is used to read the unique ID.

34.6.15 Read CRC checksum

Get the CRC checksum of a block of RAM or flash. CMD_SUCCESS followed by 8 bytes of CRC checksum in ASCII format.

The checksum is calculated as follows:

CRC-32 polynomial: x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

Seed Value: 0xFFFF FFFF

No bit order reverse for data input

No 1’s complement for data input

No bit order reverse for CRC sum

No 1’s complement for CRC sum

Table 495. UART ISP Read CRC checksum command

Command	S
Input	**Address:** The data are read from this address for CRC checksum calculation. This address must be on a word boundary. **Number of Bytes:** Number of bytes to be calculated for the CRC checksum; must be a multiple of 4.
Return Code	CMD_SUCCESS followed by data in plain binary format 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 the CRC checksum of a block of RAM or flash memory. This command is blocked when code read protection is enabled.
Example	"S 33587200 4<CR><LF>" reads the CRC checksum for 4 bytes of data from address 0x0200 8000. If checksum value is 0xCBF43926, then the host will receive: "3421780262 <CR><LF>"

34.6.16 ISP Error codes

Table 496. UART ISP Error codes

Return Code	Error code	Description
0x0	ERR_ISP_CMD_SUCCESS	Command is executed successfully. Sent by ISP handler only when command given by the host has been completely and successfully executed.
0x1	ERR_ISP_INVALID_COMMAND	Invalid command.
0x2	ERR_ISP_SRC_ADDR_ERROR	Source address is not on word boundary.
0x3	ERR_ISP_DST_ADDR_ERROR	Destination address is not on a correct boundary.
0x4	ERR_ISP_SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken in to consideration where applicable.
0x5	ERR_ISP_DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable.
0x6	ERR_ISP_COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
0x7	ERR_ISP_INVALID_SECTOR	Sector number is invalid or end sector number is greater than start sector number.
0x8	ERR_ISP_SECTOR_NOT_BLANK	Sector is not blank.
0x9	ERR_ISP_SECTOR_NOT_PREPARED_FOR_ WRITE_OPERATION	Command to prepare sector for write operation was not executed.
0xA	ERR_ISP_COMPARE_ERROR	Source and destination data not equal.
0xB	ERR_ISP_BUSY	Flash programming hardware interface is busy.
0xC	ERR_ISP_PARAM_ERROR	Insufficient number of parameters or invalid parameter.
0xD	ERR_ISP_ADDR_ERROR	Address is not on word boundary.
0xE	ERR_ISP_ADDR_NOT_MAPPED	Address is not mapped in the memory map. Count value is taken in to consideration where applicable.
0xF	ERR_ISP_CMD_LOCKED	Command is locked.
0x10	ERR_ISP_INVALID_CODE	Unlock code is invalid.
0x11	ERR_ISP_INVALID_BAUD_RATE	Invalid baud rate setting.
0x12	ERR_ISP_INVALID_STOP_BIT	Invalid stop bit setting.
0x13	ERR_ISP_CODE_READ_PROTECTION_ ENABLED	Code read protection enabled.
0x14	-	Reserved.
0x15	-	Reserved.
0x16	-	Reserved.
0x17	ERR_ISP_IRC_NO_POWER	IRC not turned on in the PDRUNCFG register.
0x18	ERR_ISP_FLASH_NO_POWER	Flash not turned on in the PDRUNCFG register.
0x19	ERR_ISP_EEPROM_NO_POWER	EEPROM not turned on in the PDRUNCFG register.
0x1A	ERR_ISP_EEPROM_NO_CLOCK	EEPROM clock disabled in the SYSAHBCLKCTRL register.
0x1B	ERR_ISP_FLASH_NO_CLOCK	Flash clock disabled in the SYSAHBCLKCTRL register.
0x1C	ERR_ISP_REINVOKE_ISP_CONFIG	Reinvoke ISP not successful.

```
typedef enum
{
 ERR_ISP_BASE = 0x00000000,
```

```
  /*0x00000001*/ ERR_ISP_INVALID_COMMAND = ERR_ISP_BASE + 1,
  /*0x00000002*/ ERR_ISP_SRC_ADDR_ERROR,
  /*0x00000003*/ ERR_ISP_DST_ADDR_ERROR,
  /*0x00000004*/ ERR_ISP_SRC_ADDR_NOT_MAPPED,
  /*0x00000005*/ ERR_ISP_DST_ADDR_NOT_MAPPED,
  /*0x00000006*/ ERR_ISP_COUNT_ERROR,
  /*0x00000007*/ ERR_ISP_INVALID_SECTOR,
  /*0x00000008*/ ERR_ISP_SECTOR_NOT_BLANK,
  /*0x00000009*/ ERR_ISP_SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION,
  /*0x0000000A*/ ERR_ISP_COMPARE_ERROR,
  /*0x0000000B*/ ERR_ISP_BUSY, /* Flash programming hardware interface is busy */
  /*0x0000000C*/ ERR_ISP_PARAM_ERROR, /* Insufficient number of parameters */
  /*0x0000000D*/ ERR_ISP_ADDR_ERROR, /* Address not on word boundary */
  /*0x0000000E*/ ERR_ISP_ADDR_NOT_MAPPED,
  /*0x0000000F*/ ERR_ISP_CMD_LOCKED, /* Command is locked */
  /*0x00000010*/ ERR_ISP_INVALID_CODE, /* Unlock code is invalid */
  /*0x00000011*/ ERR_ISP_INVALID_BAUD_RATE,
  /*0x00000012*/ ERR_ISP_INVALID_STOP_BIT,
  /*0x00000013*/ ERR_ISP_CODE_READ_PROTECTION_ENABLED,
  /*0x00000014*/ ERR_ISP_INVALID_FLASH_UNIT, /* reserved */
  /*0x00000015*/ ERR_ISP_USER_CODE_CHECKSUM, /* reserved */
  /*0x00000016*/ ERR_ISP_SETTING_ACTIVE_PARTITION, /* reserved */
  /*0x00000017*/ ERR_ISP_IRC_NO_POWER,
  /*0x00000018*/ ERR_ISP_FLASH_NO_POWER,
  /*0x00000019*/ ERR_ISP_EEPROM_NO_POWER,
  /*0x0000001A*/ ERR_ISP_EEPROM_NO_CLOCK,
  /*0x0000001B*/ ERR_ISP_FLASH_NO_CLOCK,
  /*0x0000001C*/ ERR_ISP_REINVOKE_ISP_CONFIG
} ErrorCode_t;
```

34.7 C_CAN ISP commands

The C_CAN boot loader initializes the on-chip oscillator and the CAN controller for a CAN bit rate of 100 kbit/s and sets its own CANopen Node ID to a fixed value. The boot loader then waits for CANopen SDO commands and responds to them. These commands allow to read and write anything in a so-called Object Dictionary (OD). The OD contains entries that are addressed via a 16-bit index and 8-bit subindex. The command interface is part of this OD.

The C_CAN ISP command handler allows to perform all functions that are otherwise available via the UART ISP commands, see Table 497.

The SDO commands are received, processed and responded to “forever” until the command to jump to a certain execution address (“Go”) has been received or the chip is reset.

The C_CAN ISP handler occupies the fixed CANopen node ID 125 (0x7D).

Table 497. C_CAN ISP command summary

ISP Command	C_CAN usage	UART usage
Unlock	Section 34.7.3	Table 480
Set Baud Rate	n/a	Table 481
Echo	n/a	Table 482
Write to RAM	Section 34.7.4	Table 483
Read Memory	Section 34.7.5	Table 484
Prepare sector(s) for write operation	Section 34.7.6	Table 485
Copy RAM to flash	Section 34.7.7	Table 486
Go	Section 34.7.8	Table 487
Erase sector(s)	Section 34.7.9	Table 488
Blank check sector(s)	Section 34.7.10	Table 489
Read Part ID	Section 34.7.11	Table 490
Read Boot code version	Section 34.7.12	Table 492
ReadUID	Section 34.7.13	Table 494
Compare	Section 34.7.14	Table 493

34.7.1 C_CAN ISP SDO communication

The CAN ISP node listens for CAN 2.0A (11-bit) messages with the identifier of 0x600 plus the Node ID 0x7D equaling to 0x67D. The node sends SDO responses with the identifier 0x580 plus Node ID equaling to 0x5FD. The SDO communication protocols “expedited” and “segmented” are supported. This means that communication is always confirmed: Each request CAN message will be followed by a response message from the ISP node.

The SDO block transfer mode is not supported.

For details regarding the SDO protocol, see the *CiA 301 specification*.

34.7.2 C_CAN ISP object directory

Table 498. C_CAN ISP object directory

Index	Subindex	Data type	Access	Description
0x1000	00	UNSIGNED32	RO	Device Type (ASCII “LPC1”)
0x1001	00	-	RO	Error Register (not used, 0x00)
0x1018	00	-		Identity Object
	01	UNSIGNED32	RO	Vendor ID (not used, 0x0000 0000)
	02	UNSIGNED32	RO	Part Identification Number
	03	UNSIGNED32	RO	Boot Code Version Number
0x1F50	00	-		Program Data
	01	DOMAIN	RW	Program Area
0x1F51	00	-		Program Control
	01	UNSIGNED8	RW	Program Control
0x5000	00	UNSIGNED16	WO	Unlock Code

Table 498. C_CAN ISP object directory

Index	Subindex	Data type	Access	Description
0x5010	00	UNSIGNED32	RW	Memory Read Address
0x5011	00	UNSIGNED32	RW	Memory Read Length
0x5015	00	UNSIGNED32	RW	RAM Write Address
0x5020	00	UNSIGNED16	WO	Prepare Sectors for Write
0x5030	00	UNSIGNED16	WO	Erase Sectors
0x5040	00	-		Blank Check Sectors
	01	UNSIGNED16	WO	Check sectors
	02	UNSIGNED32	RO	Offset of the first non-blank location
0x5050	00	-		Copy RAM to Flash
	01	UNSIGNED32	RW	Flash Address (DST)
	02	UNSIGNED32	RW	RAM Address (SRC)
	03	UNSIGNED16	RW	Number of Bytes
0x5060	00	-		Compare Memory
	01	UNSIGNED32	RW	Address 1
	02	UNSIGNED32	RW	Address 2
	03	UNSIGNED16	RW	Number of Bytes
	04	UNSIGNED32	RO	Offset of the first mismatch
0x5070	00	-		Execution Address
	01	UNSIGNED32	RW	Execution Address
	02	UNSIGNED8	RO	Mode (‘T’ or ‘A’), only ‘T’ supported
0x5100	00	-		Serial Number
	01	UNSIGNED32	RO	Serial Number 1
	02	UNSIGNED32	RO	Serial Number 2
	03	UNSIGNED32	RO	Serial Number 3
	04	UNSIGNED32	RO	Serial Number 4

34.7.3 Unlock

Write <Unlock Code> to [0x5000, 0]. Writing an invalid unlock code will return a dedicated abort code.

34.7.4 Write to RAM

Set RAM write address by writing to [0x5015, 0]. Then write the binary data to [0x1F50, 1]. Since this is a DOMAIN entry, the data can be continuously written. The host terminates the write. The write address in [0x5015, 0] auto-increments, so a write of a larger area may be done in multiple successive write cycles to [0x1F50, 1].

34.7.5 Read memory

Set RAM read address by writing to [0x5010, 0] and the read length by writing to [0x5011,0]. Then read the binary data from [0x1F50,1]. Since this is a DOMAIN entry, the data is continuously read. The device terminates the read when the number of bytes in the read length entry has been read. The read address in [0x5010, 0] auto-increments, so a read of a larger area may be done in multiple successive read cycles from [0x1F50,1].

34.7.6 Prepare sectors for write operation

Write a 16-bit value to [0x5020, 0] with the start sector number in the lower eight bits and the end sector number in the upper eight bits.

34.7.7 Copy RAM to flash

Write the parameters into entry [0x5050, 1 to 3]. Writing the number of bytes into [0x5050,3] starts the programming.

See Section 34.6.7 “Copy RAM to flash” for limitations on the write-to-flash process.

34.7.8 Go

Write the start address into [0x5070, 0]. Then trigger the “start application” command by writing the value 0x1 to [0x1F51, 1].

34.7.9 Erase sectors

Write a 16-bit value to [0x5030, 0] with the start sector number in the lower eight bits and the end sector number in the upper eight bits.

34.7.10 Blank check sectors

Write a 16-bit value to [0x5040, 1] with the start sector number in the lower eight bits and the end sector number in the upper eight bits.

If the SECTOR_NOT_BLANK abort code is returned, the entry [0x5040, 2] contains the offset of the first non-blank location.

34.7.11 Read PartID

Read [0x1018, 2]. See Table 491.

34.7.12 Read boot code version

Read [0x1018, 3]

34.7.13 Read serial number

Read [0x5100, 1 to 4]

34.7.14 Compare

Write the parameters into entry [0x5060, 1 to 3]. The write of the number of bytes into[0x5060, 3] starts the comparison.

If the COMPARE_ERROR abort code is returned, the entry [0x5060, 4] can be read to get the offset of the first mismatch.

34.7.15 C_CAN ISP SDO Abort codes

The OD entries that trigger an action return an appropriate SDO abort code when the action returned an error. The abort code is 0x0F00 0000 plus the value of the corresponding ISP return code in the lowest byte. Table 499 shoes the list of abort codes.

In addition, the regular CANopen SDO abort codes for invalid access to OD entries are also supported.

Table 499. C_CAN ISP SDO abort codes

UART ISP Error code	SDO Abort code	Value
ADDR_ERROR	SDOABORT_ADDR_ERROR	0x0F00 000D
ADDR_NOT_MAPPED	SDOABORT_ADDR_ NOT_MAPPED	0x0F00 000E
CMD_LOCKED	SDOABORT_CMD_LOCKED	0x0F00 000F
CODE_READ_PROTECTION_ ENABLED	SDOABORT_CODE_READ _PROTECTION_ENABLED	0x0F00 0013
COMPARE_ERROR	SDOABORT_COMPARE_ERROR	0x0F00 000A
COUNT_ERROR	SDOABORT_COUNT_ERROR	0x0F00 0006
DST_ADDR_ERROR	SDOABORT_DST_ADDR_ERROR	0x0F00 0003
DST_ADDR_NOT_MAPPED	SDOABORT_DST_ADDR_ NOT_MAPPED	0x0F00 0005
INVALID_CODE	SDOABORT_INVALID_CODE	0x0F00 0010
INVALID_COMMAND	SDOABORT_INVALID_COMMAND	0x0F00 0001
INVALID_SECTOR	SDOABORT_INVALID_SECTOR	0x0F00 0007
PARAM_ERROR	SDOABORT_PARAM_ERROR	0x0F00 000C
SECTOR_NOT_BLANK	SDOABORT_SECTOR_ NOT_BLANK	0x0F00 0008
SECTOR_NOT_PREPARED_ FOR_WRITE_OPERATION	SDOABORT_SECTOR_NOT_ PREPARED_FOR_WRITE_ OPERATION	0x0F00 0009
SRC_ADDR_ERROR	SDOABORT_SRC_ADDR_ERROR	0x0F00 0002
SRC_ADDR_NOT_MAPPED	SDOABORT_SRC_ADDR_ NOT_MAPPED	0x0F00 0004

34.7.16 Differences to fully-compliant CANopen

While the boot loader uses the SDO communication protocol and the Object Dictionary data organization method, it is not a fully CiA 301 standard compliant CANopen node. The following features are not available or different to the standard:

- Network Management (NMT) message processing not available.
- Heartbeat message and entry 0x1017 not available.
- Uses proprietary SDO abort codes to indicate device errors.
- To speed up communication, “empty” SDO responses during SDO segmented download/write to the node are shortened to one data byte, rather than full eight data bytes as the standard describes.
- Entry [0x1018, 1] Vendor ID reads 0x0000 0000 rather than an official CiA-assigned unique Vendor ID.
- The host must use a different method to identify the CAN ISP devices.

34.8 IAP commands

For in application programming the IAP routine should be called with a word pointer in register r0 pointing to memory (RAM) containing command code and parameters. The result of the IAP command is returned in the result table pointed to by register r1. The user can reuse the command table for result by passing the same pointer in registers r0 and r1. The parameter table should be big enough to hold all the results in case the number of results are more than number of parameters. Parameter passing is illustrated in the Figure 94.

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 5, 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 0x0300 0200 location and it is thumb code.

The IAP function could be called in the following way using C:

Define the IAP location entry point. Since the 0th bit of the IAP location is set there will be a change to Thumb instruction set when the program counter branches to this address.

```
#define IAP_LOCATION 0x0300 0200
```

Define data structure or pointers to pass IAP command table and result table to the IAP function:

```
unsigned int command_param[5];
unsigned int status_result[5];
```

or

```
unsigned int * command_param;
unsigned int * status_result;
command_param = (unsigned int *) 0x...
status_result =(unsigned int *) 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 the function pointer:

```
#define IAP_LOCATION 0x0300 0204

iap_entry=(IAP) IAP_LOCATION;
```

To call the IAP use the following statement.

```
iap_entry (command_param,status_result);
```

Up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively (see the *ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05)*. Additional parameters are passed on the stack. Up to 4 parameters can be returned in the r0, r1, r2 and r3 registers respectively. Additional parameters are returned indirectly via memory. Some of the IAP calls require more than 4 parameters. If the ARM suggested scheme is used for the parameter passing/returning then it might create problems due to difference in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such risk.

The flash memory is not accessible during a write or erase operation. IAP commands, which results in a flash write/erase operation, use 32 bytes of space in the top portion of the on-chip RAM for execution. The user program should not be use this space if IAP flash programming is permitted in the application.

Table 500. IAP Command Summary

IAP Command	Command code	Reference
Prepare sector(s) for write operation	50 (decimal)	Table 501
Copy RAM to flash	51 (decimal)	Table 502
Erase sector(s)	52 (decimal)	Table 503
Blank check sector(s)	53 (decimal)	Table 504
Read Part ID	54 (decimal)	Table 505
Read Boot code version	55 (decimal)	Table 506
Compare	56 (decimal)	Table 507
Reinvoke ISP	57 (decimal)	Table 508
Read UID	58 (decimal)	Table 509
Erase page	59 (decimal)	Table 510
EEPROM Write	61 (decimal)	Table 511
EEPROM Read	62 (decimal)	Table 512

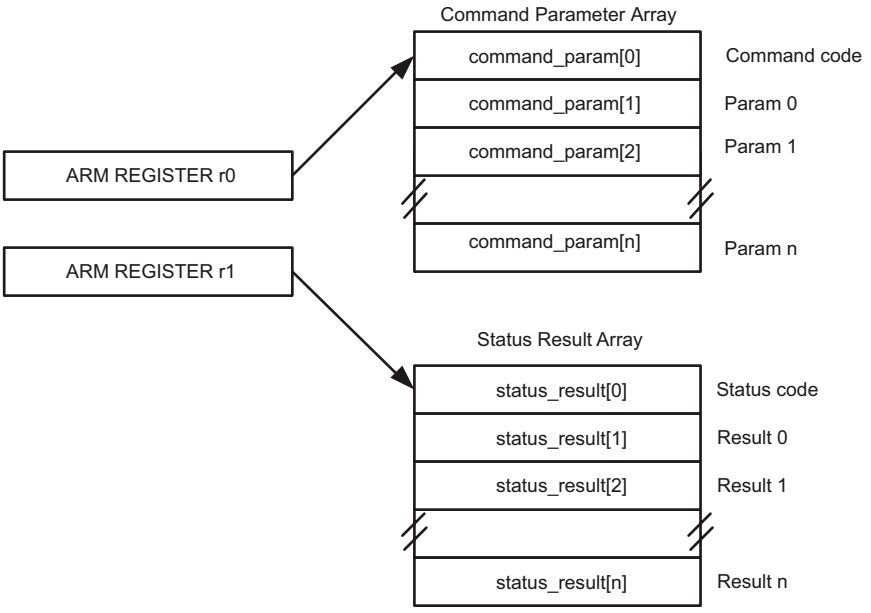


Fig 94. IAP parameter passing

34.8.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two step process.

Table 501. IAP Prepare sector(s) for write operation command

Command	Prepare sector(s) for write operation
Input	**Command code: 50 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Status code	CMD_SUCCESS | BUSY | INVALID_SECTOR
Result	None
Description	This command must be executed before executing "Copy RAM to flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot sector can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.

34.8.2 Copy RAM to flash

See Section 34.6.7 for limitations on the write-to-flash process.

Table 502. IAP Copy RAM to flash command

Command	Copy RAM to flash
Input	**Command code: 51 (decimal)** **Param0(DST):** Destination flash address where data bytes are to be written. This address should be a 256 byte boundary. **Param1(SRC):** Source RAM address from which data bytes are to be read. This address should be a word boundary. **Param2:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096. **Param3:** System Clock Frequency (CCLK) in kHz.
Status code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not a word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | BUSY
Result	None
Description	This command is used to program the flash memory. The affected sectors should be prepared first by calling "Prepare Sector for Write Operation" command. The affected sectors are automatically protected again once the copy command is successfully executed. The boot sector can not be written by this command. Also see Section 34.3.3 for the number of bytes that can be written. The CCLK clock value passed as a parameter is overwritten by the fixed value of 12 MHz, which is the reference clock used by the flash controller. **Remark:** All user code must be written in such a way that no master accesses the flash while this command is executed and the flash is programmed.

34.8.3 Erase Sector(s)

Table 503. IAP Erase Sector(s) command

Command	Erase Sector(s)
Input	**Command code: 52 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number). **Param2:** System Clock Frequency (CCLK) in kHz.

Table 503. IAP Erase Sector(s) command

Command	Erase Sector(s)
Status code	CMD_SUCCESS | BUSY | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | INVALID_SECTOR
Result	None
Description	This command is used to erase a sector or multiple sectors of on-chip flash memory. The boot sector can not be erased by this command. To erase a single sector use the same "Start" and "End" sector numbers. The CCLK clock value passed as a parameter is overwritten by the fixed value of 12 MHz, which is the reference clock used by the flash controller. **Remark:** All user code must be written in such a way that no master accesses the flash while this command is executed and the flash is erased.

34.8.4 Blank check sector(s)

Table 504. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	**Command code: 53 (decimal)** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Status 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.

34.8.5 Read Part Identification number

Table 505. IAP Read Part Identification command

Command	Read part identification number
Input	**Command code: 54 (decimal)** **Parameters:** None
Status code	CMD_SUCCESS
Result	**Result0:** Part Identification Number.
Description	This command is used to read the part identification number.

34.8.6 Read Boot code version number

Table 506. IAP Read Boot Code version number command

Command	Read boot code version number
Input	**Command code: 55 (decimal)** **Parameters:** None
Status code	CMD_SUCCESS
Result	**Result0:** 2 bytes of boot code version number. Read as <byte1(Major)>.<byte0(Minor)>
Description	This command is used to read the boot code version number.

34.8.7 Compare <address1> <address2> <no of bytes>

Table 507. IAP Compare command

Command	Compare
Input	**Command code: 56 (decimal)** **Param0(DST):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param1(SRC):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param2:** Number of bytes to be compared; should be a multiple of 4.
Status code	CMD_SUCCESS | COMPARE_ERROR | COUNT_ERROR (Byte count is not a multiple of 4) | ADDR_ERROR | ADDR_NOT_MAPPED
Result	**Result0:** Offset of the first mismatch if the status code is COMPARE_ERROR.
Description	This command is used to compare the memory contents at two locations. **The result may not be correct when the source or destination includes any of the first 512 bytes starting from address zero. The first 512 bytes can be re-mapped to RAM.**

34.8.8 Reinvoke ISP

Table 508. Reinvoke ISP

Command	Compare
Input	**Command code: 57 (decimal)** Param0: ISP mode. 1 = UART ISP, 2 = USB ISP, 3 = C_CAN ISP.
Status code	None
Result	**None.**
Description	This command is used to invoke the boot loader in ISP mode. It maps boot vectors and configures the peripherals for ISP. Which peripheral (UART, C_CAN, UART, or USB) is selected for ISP mode depends on the parameter written with this command. This command may be used when a valid user program is present in the internal flash memory and the ISP entry pins are not accessible to force the ISP mode. Before calling this command, enable the clocks to the GPIO0/1/2 blocks in the SYSAHBCLKCTRL0 register.

34.8.9 ReadUID

Table 509. IAP ReadUID command

Command	Compare
Input	**Command code: 58 (decimal)**
Status 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.

34.8.10 Erase page

Table 510. IAP Erase page command

Command	Erase page
Input	**Command code: 59 (decimal)** **Param0:** Start page number. **Param1:** End page number (should be greater than or equal to start page) **Param2:** System Clock Frequency (CCLK) in kHz.
Status code	CMD_SUCCESS | BUSY | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | INVALID_SECTOR
Result	None
Description	This command is used to erase a page or multiple pages of on-chip flash memory. To erase a single page use the same "start" and "end" page numbers. The CCLK clock value passed as a parameter is overwritten by the fixed value of 12 MHz, which is the reference clock used by the flash controller. **Remark:** All user code must be written in such a way that no master accesses the flash while this command is executed and the flash is erased.

34.8.11 Write EEPROM

Table 511. IAP Write EEPROM command

Command	Compare
Input	**Command code: 61 (decimal)** **Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be written. **Param3:** System Clock Frequency (CCLK) in kHz.

Table 511. IAP Write EEPROM command

Command	Compare
Status 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. The CCLK clock value passed as a parameter is overwritten by the fixed value of 12 MHz, which is the reference clock used by the flash controller. **Remark:** The top 64 bytes of the EEPROM memory are reserved and cannot be written to.

34.8.12 Read EEPROM

Table 512. IAP Read EEPROM command

Command	Compare
Input	**Command code: 62 (decimal)** **Param0:** EEPROM address. **Param1:** RAM address. **Param2:** Number of bytes to be read. **Param3:** System Clock Frequency (CCLK) in kHz.
Status 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. The CCLK clock value passed as a parameter is overwritten by the fixed value of 12 MHz, which is the reference clock used by the flash controller.

34.8.13 IAP Status Codes

Table 513. IAP Status codes Summary

Status code	Mnemonic	Description
0	CMD_SUCCESS	Command is executed successfully.
1	INVALID_COMMAND	Invalid command.
2	SRC_ADDR_ERROR	Source address is not on a word boundary.
3	DST_ADDR_ERROR	Destination address is not on a correct boundary.
4	SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken in to consideration where applicable.
5	DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable.
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data is not same.
11	BUSY	flash programming hardware interface is busy.

Table 513. IAP Status codes Summary

Status code	Mnemonic	Description
17	ERR_ISP_IRC_NO_POWER	-
18	ERR_ISP_FLASH_NO_POWER	-
19	ERR_ISP_EEPROM_NO_POWER	-
1A	ERR_ISP_EEPROM_NO_CLOCK	-
1B	ERR_ISP_FLASH_NO_CLOCK	-

35.1 How to read this chapter

The power profiles are available for all LPC15xx parts. The description applies to boot ROM version 14.2 and higher.

35.2 Features

- ROM-based application.
- Simple API to control power consumption and wake-up in all power modes.
- Manage power consumption for sleep and active modes
- Configure PLL.
- Prepare the part to enter low power modes (sleep, deep-sleep, power-down, and deep power-down).
- Configure wake-up from Deep-sleep and power-down via BOD, WWDT, analog comparators, temperature sensor, and internal reference voltage.

35.3 General description

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

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

In addition, the power profile includes routines to select the optimal PLL settings for a given system clock and PLL input clock and to configure the part correctly for any of the low power modes.

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.

The API calls to the ROM are performed by executing functions which are pointed by a pointer within the ROM Driver Table. Figure 95 shows the pointer structure used to call the Power Profiles API.

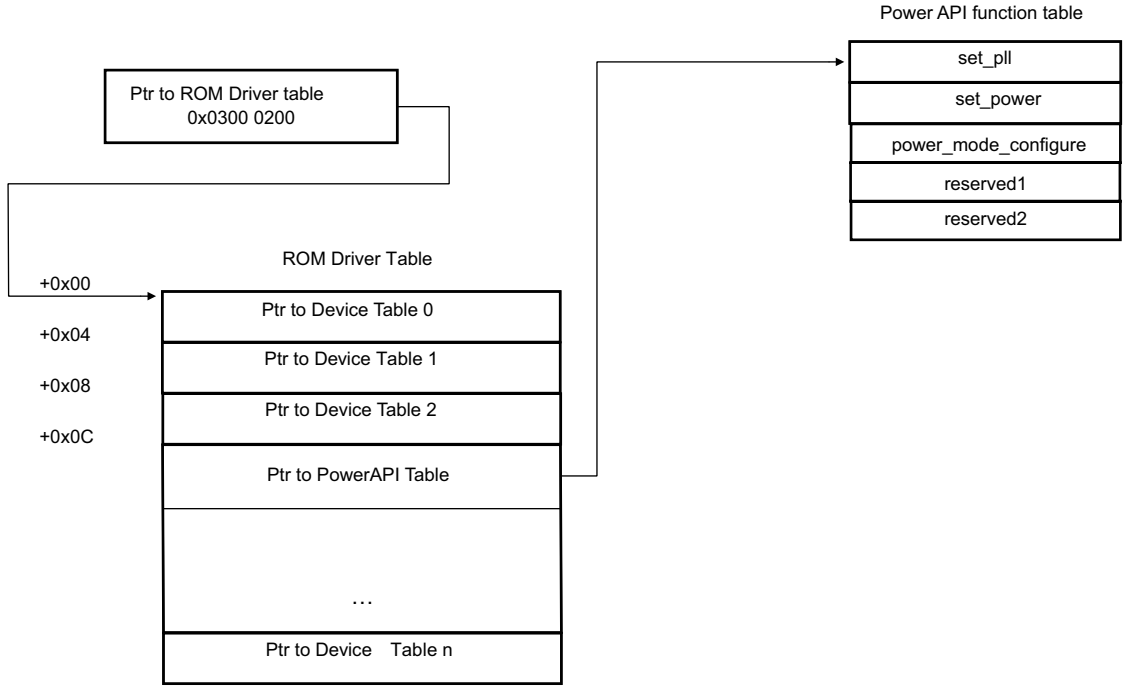


Fig 95. Power profiles pointer structure

35.4 API description

The power profile API provides functions to configure the system clock and optimize the system setting for lowest power consumption.

Table 514. Power profile API calls

API call	Description	Reference
`void set_pll(command, result);`	Power API set_pll routine for active and sleep modes	Table 515
`void set_power(command, result);`	Power API set_power routine for active and sleep modes	Table 516
`void power_mode_configure(mode, peripheral);`	Power API power mode configuration routine.This API prepares the chip for sleep, deep-sleep, power-down, or deep power-down mode and selects which analog peripherals can wake up the part from deep-sleep or power-down modes.	Table 517
`void reserved1();`	Reserved	-
`void reserved2();`	Reserved	-

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

```
typedef struct PWRD_API {
     void (*set_pll)(uint32_t cmd[], uint32_t resp[]);/*!< Set PLL function */
```

```
    void (*set_power)(uint32_t cmd[], uint32_t resp[]);/*!< Set power function */
    /*!< Sets the chip in low power modes */
    void (*power_mode_configure)(unsigned int mode, unsigned int peripheral);
    void (*reserved1)();
    void (*reserved2)();
} PWRD_API_T;

#define rom_driver_ptr (*(ROM) **) 0x0300 0200)
pPWRD = (PWRD_API_T *)(rom_driver_ptr->pPWRD);
```

35.4.1 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 46), the main clock source must be set to the input clock to the system PLL (Table 40). and the system/AHB clock divider must be set to 1 (Table 49).

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

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

Table 515. set_pll routine

Routine	set_pll
Prototype	`void set_pll(command, result);`
Input parameter	**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)
Return	None.
Description	Sets the system PLL.

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
```

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

set_pll configures a setup in which the main clock does not exceed 72 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 72000 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.

35.4.1.2 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.

35.4.1.3 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.

35.4.2 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: Use the set_power routine with SYSAHBCLKDIV = 1 (System clock divider register, see Table 49).

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

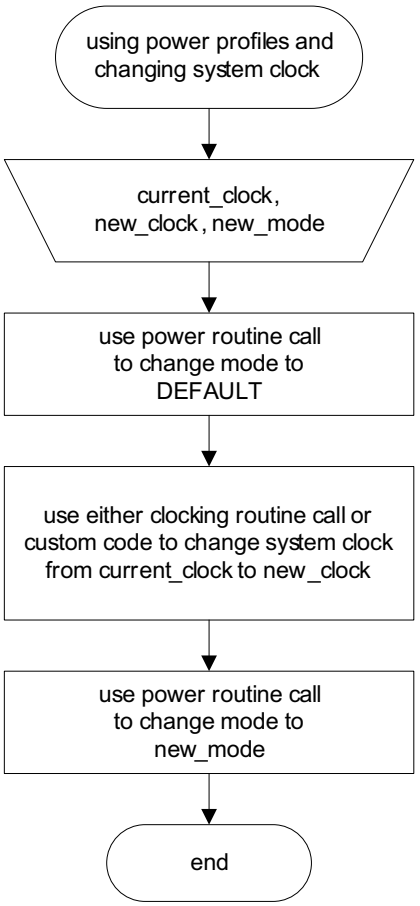


Fig 96. Power profiles usage

Table 516. set_power routine

Routine	set_power
Prototype	`void set_power(command, result);`
Input parameter	**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
Return	None.
Description	Configures the power mode in active and sleep modes.

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
```

35.4.2.1 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 72 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.

35.4.2.2 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.

35.4.2.3 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 72 MHz inclusive.

35.4.3 power_mode_configure

The power_mode_configure API prepares the part to enter any of the low power modes. Specifically for the deep-sleep and power-down modes, the API function configures which analog components remain running in those two modes, so that an interrupt from one of the analog peripherals can wake up the part. For example, a comparator edge signal created by the temperature sensor crossing a reference voltage threshold can wake up the part from deep-sleep mode.

Table 517. power_mode_configure routine

Routine	power_mode_configure
Prototype	`void power_mode_configure(mode, peripheral);`
Input parameter	Param0: mode Param1: peripheral
Result	Void
Return	None
Description	Defines the low power mode (either deep-sleep, power-down, or deep power-down modes). Use this function before issuing WFI command.

Remark: After setting the mode in this API, the user code must execute the WFI instruction to put the chip into the selected low power mode.

Remark: Aside from the analog peripherals listed with this parameter, the serial peripherals can also wake up the chip from deep-sleep or power-down modes on an interrupt triggered by an incoming signal. This wake-up scenario is not configured using the power_mode_config API. For details, see Section 23.7.6 “USB wake-up”, Section 24.3.2 “Configure the USART for wake-up”, Section 26.4.3 “Configure the I2C for wake-up”, or Section 25.3.1 “Configure the SPI for wake-up”.

35.4.3.1 Param0: mode

The mode parameter defines the low power mode and prepares the chip to enter the selected mode.

The following modes are valid:

```
#define SLEEP              0
#define DEEP_SLEEP         1
#define POWER_DOWN         2
#define DEEP_POWER_DOWN    3
```

35.4.3.2 Param1: peripheral

If sleep or deep power-down modes are selected with the mode parameter, the peripheral parameter is ignored.

The peripheral parameter defines which analog peripherals can wake up the chip from deep-sleep or power-down mode. The selected peripherals remain running in deep-sleep or power-down mode. For example, the watchdog oscillator must be running if the WWDT is to remain active in deep-sleep or power-down mode.

The peripheral parameter is an 8-bit value with each bit representing one analog peripheral. If the bit is set to 0, the analog peripheral is running in deep-sleep or power-down mode. Otherwise, the analog peripheral is powered down.

Table 518. Bit values for the power_mode_configure peripheral parameter

Bit	Value	Description
0		WDOSC (watchdog oscillator) power in deep-sleep or power-down modes. Enable this oscillator to run the WWDT in deep-sleep or power-down modes.
	0	WDOSC running.
	1	WDOSC powered down.
1		BOD (Brown-Out Detect) power in deep-sleep or power-down modes.
	0	BOD enabled.
	1	BOD powered down.
2		ACMP0 (Analog comparator 0) power in deep-sleep or power-down modes.
	0	ACMP0 enabled.
	1	ACMP0 powered down.
3		ACMP1 (Analog comparator 1) power in deep-sleep or power-down modes.
	0	ACMP1 enabled.
	1	ACMP1 powered down.
4		ACMP2 (Analog comparator 2) power in deep-sleep or power-down modes.
	0	ACMP2 enabled.
	1	ACMP2 powered down.
5		ACMP3 (Analog comparator 3) power in deep-sleep or power-down modes.
	0	ACMP3 enabled.
	1	ACMP3 powered down.
6		IREF (Internal reference voltage) power in deep-sleep or power-down modes. The IREF wake-up source is only enabled when at least one of the analog comparator bits is set to enabled. Wake-up via the internal reference voltage requires that internal reference voltage output is configured as a comparator input and compared to another voltage level.
	0	IREF enabled.
	1	IREF powered down.
7		TS (Temperature sensor) power in deep-sleep or power-down modes. The temperature sensor wake-up source is only enabled when at least one of the analog comparator bits is set to enabled.Wake-up via the temperature sensor reading requires that the temperature sensor is configured as a comparator input and compared to another voltage level.
	0	TS enabled.
	1	TS powered down.

35.4.4 Error codes

The error code is returned in the result field of the set_pll and set_power API functions.

Table 519. Error codes for set_pll

Return code	Error Code	Description
0	PLL_CMD_SUCCESS	-
1	PLL_INVALID_FREQ	-
2	PLL_INVALID_MODE	-
3	PLL_FREQ_NOT_FOUND	-
4	PLL_NOT_LOCKED	

```
#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
```

Table 520. Error codes for set_power

Return code	Error Code	Description
0	PARAM_CMD_SUCCESS	-
1	PARAM_INVALID_FREQ	-
2	PARAM_INVALID_MODE	-

```
#define                 PARAM_CMD_SUCCESS                  0
#define                 PARAM_INVALID_FREQ          1
#define                 PARAM_INVALID_MODE          2
```

35.5 Functional description

35.5.1 Clock control

See Section 35.5.1.1 to Section 35.5.1.6 for examples of the clock control API.

35.5.1.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 12000;
command[1] = 840000;
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 84 MHz. The application was ready to infinitely wait for the PLL to lock. But the expected system clock of 84 MHz exceeds the maximum of 72 MHz. Therefore set_pll returns PLL_INVALID_FREQ in result[0] and 12000 in result[1] without changing the PLL settings.

35.5.1.2 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.

35.5.1.3 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.

35.5.1.4 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.

35.5.1.5 System clock greater than or equal to the expected value

```
command[0] = 12000;
command[1] = 20000;
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 20 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.

35.5.1.6 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.

35.5.2 Power control

See Section 35.5.1.1 and Section 35.5.2.2 for examples of the power control API.

35.5.2.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 72;
command[1] = PWR_CPU_PERFORMANCE;
command[2] = 84;
(*rom)->pWRD->set_power(command, result);
```

The above setup would be used in a system running at the main and system clock of 30 MHz, with a need for maximum CPU processing power. Since the specified 84 MHz clock is above the 72 MHz maximum, set_power returns PWR_INVALID_FREQ in result[0] without changing anything in the existing power setup.

35.5.2.2 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.

35.5.3 Low power modes control

35.5.3.1 Enter sleep mode

```
/* get Power config API table pointer */
pPWRD = (PWRD *)(rom_drivers_ptr->pPWRD);
/* parameter peripheral is don't care*/
pPWRD->power_mode_configure( SLEEP, 0x0 );
/* going to sleep mode. */
  __WFI();
```

35.5.3.2 Enter deep-sleep mode and set up temperature sensor trip event for wake-up

```
/* configure acmp0 event to wake up the chip from deep-sleep */
LPC_SYSCON->STARTERP1 = 0x100;
/* get Power config API table pointer */
pPWRD = (PWRD *)(rom_drivers_ptr->pPWRD);
/* WDT_OSC and BOD are turned off, COMP0 is turned on, COMP2 and COMP3 are turned off,
     IREF is turned off, TEMP_SENSE is turned on. */
pPWRD->power_mode_configure( DEEP_SLEEP, 0x007B );
/* going to deep sleep mode. */
```

```
  __WFI();
```

35.5.3.3 Enter power-down mode and set up WWDT and BOD for wake-up

```
/* configure wwdt and bod event to wake up the chip from power-down*/
LPC_SYSCON->STARTERP0 = 0x3;
/* get Power config API table pointer */
pPWRD = (PWRD *)(rom_drivers_ptr->pPWRD);
/* WDT_OSC and BOD are turned on, COMP0 is turned off, COMP2 and COMP3 are turned off,
     IREF is turned off, TEMP_SENSE is turned off. */
pPWRD->power_mode_configure( POWER_DOWN, 0xFC );
/* going to power-down mode. */
  __WFI();
```

35.5.3.4 Enter deep power-down mode and wake up using WAKEUP pin

```
/* get Power config API table pointer */
pPWRD = (PWRD *)(rom_drivers_ptr->pPWRD);
/* parameter peripheral is don't care*/
pPWRD->power_mode_configure( DEEP_POWER_DOWN, 0x0 );
/* going to deep power-down mode. */
  __WFI();
```

36.1 How to read this chapter

The USART ROM driver routines are available on all parts.

36.2 Features

- Send and receive characters in asynchronous or synchronous mode
- Send and receive multiple characters (line) in asynchronous or synchronous UART mode
- Support for DMA mode

36.3 General description

The UART API handles sending and receiving characters using any of the USART blocks in asynchronous mode.

Remark: Because all USARTS share a common fractional divider, the uart_init routine returns the value for the common divider.

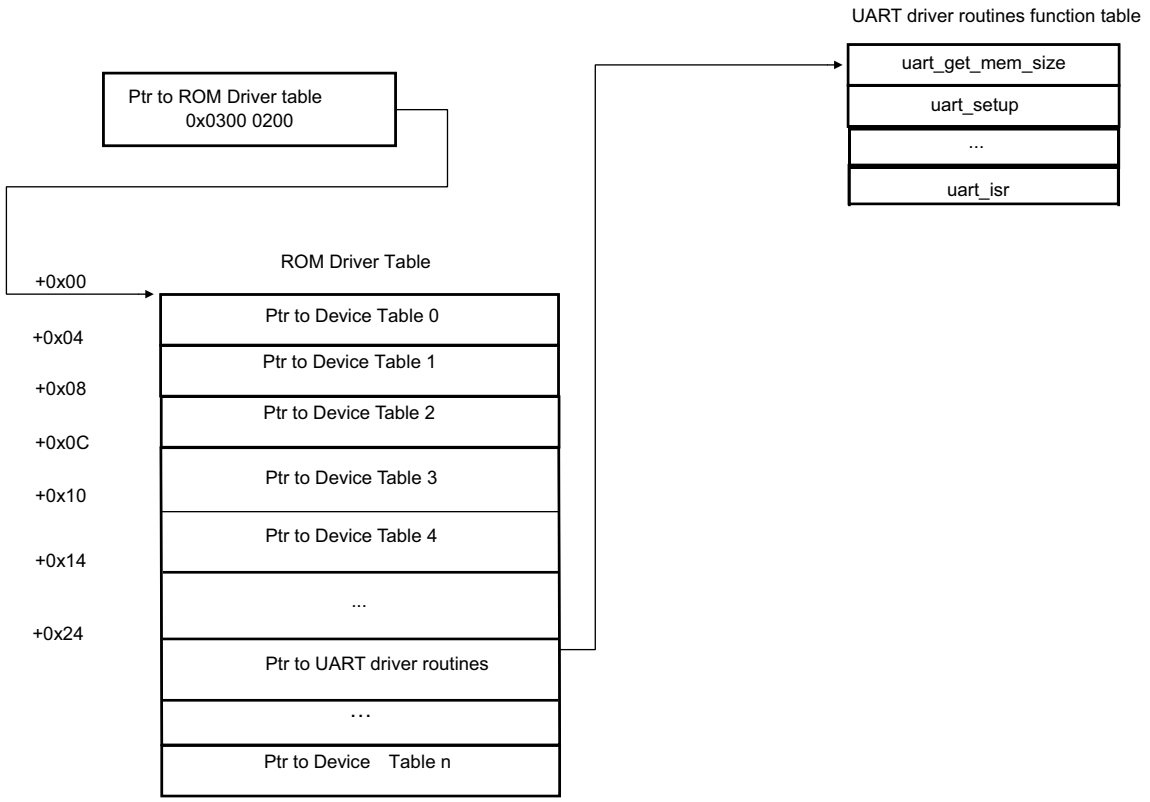


Fig 97. USART driver routines pointer structure

36.4 API description

The UART API contains functions to send and receive characters via any of the USART blocks.

Table 521. UART API calls

API call	Description	Reference
`uint32_t uart_get_mem_size( void) ;`	UART get memory size for UART instance	Table 522
`UART_HANDLE_T* uart_setup(uint32_t base_addr, uint8_t *ram) ;`	UART set-up	Table 523
`uint32_t uart_init(UART_HANDLE_T* handle, UART_CONFIG set);`	UART init	Table 524
`uint8_t uart_get_char(UART_HANDLE_T* handle);`	UART get character	Table 525
`void uart_put_char(UART_HANDLE_T* handle, uint8_t data);`	UART put character	Table 526
`uint32_t uart_get_line(UART_HANDLE_T* handle, UART_PARAM_T param);`	UART get line	Table 527
`uint32_t uart_put_line(UART_HANDLE_T* handle, UART_PARAM_T param);`	UART put line	Table 528
`void uart_isr(UART_HANDLE_T* handle);`	UART interrupt service routine	Table 529

The following structure has to be defined to use the UART API:

```
typedef struct  UART_API {        // index of all the UART driver functions
```

```
                uint32_t (*uart_get_mem_size)(void);
                UART_HANDLE_T (*uart_setup)(uint32_t base_addr, uint8_t *ram);
                uint32_t (*uart_init)(UART_HANDLE_T handle, UART_CONFIG_T *set);
                //--polling only functions--//
                uint8_t (*uart_get_char)(UART_HANDLE_T handle);
                void (*uart_put_char)(UART_HANDLE_T handle, uint8_t data);
                //--polling or interrupt functions--//
                uint32_t (*uart_get_line)(UART_HANDLE_T handle, UART_PARAM_T * param);
                uint32_t (*uart_put_line)(UART_HANDLE_T handle, UART_PARAM_T * param);
                void (*uart_isr)(UART_HANDLE_T handle);
} UARTD_API_T  ;            // end of structure
```

36.4.1 UART get memory size

Table 522. uart_get_mem_size

Routine	uart_get_mem_size
Prototype	`uint32_t uart_get_mem_size( void) ;`
Input parameter	None.
Return	Memory size in bytes.
Description	Get the memory size needed by one UART instance.

36.4.2 UART setup

Table 523. uart_setup

Routine	uart_setup
Prototype	`UART_HANDLE_T* uart_setup(uint32_t base_addr, uint8_t *ram) ;`
Input parameter	base_addr: Base address of register for this UART block. ram: Pointer to the memory space for UART instance. The size of the memory space can be obtained by the uart_get_mem_size function.
Return	The handle to corresponding UART instance.
Description	Setup UART instance with provided memory and return the handle to this instance.

36.4.3 UART init

See Section 36.4.10.1 for the UART_CONFIG and Section 36.4.10.2 for UART_HANDLE_T variables. This function returns the value that must be written into the fractional baud rate generator register FRGCTRL in the SYSCON block to generate the desired baud rate.

Table 524. uart_init

Routine	uart_init
Prototype	`uint32_t uart_init(UART_HANDLE_T* handle, UART_CONFIG set);`
Input parameter	handle: The handle to the UART instance. set: configuration for UART operation.
Return	Fractional divider value if System clock is not integer multiples of baud rate.
Description	Setup baud rate and operation mode for UART, then enable UART.

36.4.4 UART get character

See Section 36.4.10.2 for UART_HANDLE_T variable. This function works in polling mode only.

Table 525. uart_get_char

Routine	uart_get_char
Prototype	`uint8_t uart_get_char(UART_HANDLE_T* handle);`
Input parameter	handle: The handle to the UART instance.
Return	Received data
Description	Receive one Char from UART. This functions is only returned after Char is received.

36.4.5 UART put character

See Section 36.4.10.2 for UART_HANDLE_T variable. This function works in polling mode only.

Table 526. uart_put_char

Routine	uart_put_char
Prototype	`void uart_put_char(UART_HANDLE_T* handle, uint8_t data);`
Input parameter	handle: The handle to the UART instance. data: data to be sent out.
Return	None.
Description	Send one Char through UART. This function is only returned after data is sent.

36.4.6 UART get line

See Section 36.4.10.2 for UART_HANDLE_T variable and Section 36.4.10.3 for the PARAM_T variable.

Table 527. uart_get_line

Routine	uart_get_line
Prototype	`uint32_t  uart_get_line(UART_HANDLE_T* handle, UART_PARAM_T param);`
Input parameter	handle: The handle to the UART instance. param: Refer to UART_PARAM_T definition.
Return	Error code: ERR_UART_RECEIVE_ON - UART receive is ongoing.
Description	Receive multiple bytes from UART.

36.4.7 UART put line

See Section 36.4.10.2 for UART_HANDLE_T variable and Section 36.4.10.3 for the PARAM_T variable.

Table 528. uart_put_line

Routine	uart_put_line
Prototype	`uint32_t uart_put_line(UART_HANDLE_T* handle, UART_PARAM_T param);`
Input parameter	handle: The handle to the UART instance. param: Refer to UART_PARAM_T definition.
Return	Error code: ERR_UART_SEND_ON - UART sending is ongoing.
Description	Send string (end with \0) or raw data through UART.

36.4.8 UART interrupt service routine

See Section 36.4.10.2 for UART_HANDLE_T variable.

Table 529. uart_isr

Routine	uart_isr
Prototype	`void uart_isr(UART_HANDLE_T* handle);`
Input parameter	handle: The handle to the UART instance.
Return	None.
Description	UART interrupt service routine. To use this routine, the corresponding USART interrupt must be enabled. This function is invoked by the user ISR.

36.4.9 Error codes

Table 530. Error codes

Return code	Error Code	Description
0x0008 0001	ERR_UART_RXD_BUSY	UART receive is busy
0x0008 0002	ERR_UART_TXD_BUSY	UART transmit is busy
0x0008 0003	ERR_UART_OVERRUN_FRAME_PARITY_NOISE	Overrun error, Frame error, parity error, RxNoise error
0x0008 0004	ERR_UART_UNDERRUN	Underrun error
0x0008 0005	ERR_UART_PARAM	Parameter error
0x0008 0006	ERR_UART_BAUDRATE	Baudrate setting error

```
typedef enum
{
 ERR_UART_BASE = 0x00080000,
  /*0x00080001*/ ERR_UART_RXD_BUSY = ERR_UART_BASE+1,   //UART rxd is busy
  /*0x00080002*/ ERR_UART_TXD_BUSY,   //UART txd is busy
  /*0x00080003*/ ERR_UART_OVERRUN_FRAME_PARITY_NOISE, //overrun err, frame err, parity
      // err, RxNoise err
  /*0x00080004*/ ERR_UART_UNDERRUN,    //underrun err
  /*0x00080005*/ ERR_UART_PARAM,       //parameter error
  /*0x00080006*/ ERR_UART_BAUDRATE //baudrate setting error
} ErrorCode_t;
```

36.4.10 UART ROM driver variables

36.4.10.1 UART_CONFIG structure

```
typdef  struct UART_CONFIG {
     uint32_t sys_clk_in_hz; // Sytem clock in hz.
     uint32_t baudrate_in_hz; // Baudrate in hz
     uint8_t  config;
               //bit 1:0
               // 00: 7 bits length, 01: 8 bits lenght, others: reserved
               //bit3:2
               //   00: No Parity, 01: reserved, 10: Even, 11: Odd
               //bit4
               //   0: 1 Stop bit, 1: 2 Stop bits
     uint8_t sync_mod;
               //bit0: 0(Async mode), 1(Sync mode)
               //bit1: 0(Un_RXD is sampled on the falling edge of SCLK)
               //        1(Un_RXD is sampled on the rising edge of SCLK)
               //bit2: 0(Start and stop bits are transmitted as in asynchronous
               //mode)
               //        1(Start and stop bits are not transmitted)
               //bit3: 0(the UART is a slave on Sync mode)
               //       1(the UART is a master on Sync mode)
     uint16_t error_en;
               //bit0: OverrunEn, bit1: UnderrunEn, bit2: FrameErrEn,
               //bit3: ParityErrEn, bit4: RxNoiseEn
} UART_CONFIG_T;
```

36.4.10.2 UART_HANDLE_T

The handle to the instance of the UART driver. Each UART has one handle, so there can be several handles for each UART block. This handle is created by Init API and used by the transfer functions for the corresponding UART block.

```
typedef  void     *UART_HANDLE_T   ;    // define TYPE for UART handle pointer
```

36.4.10.3 UART_PARAM_T

```
typedef struct  uart_A { // parms passed to UART driver function
uint8_t  *  buffer ; // The pointer of buffer.
                    // For uart_get_line function, buffer for receiving data.
                    // For uart_put_line function, buffer for transmitting data.
uint32_t   size;     // [IN] The size of buffer.
                    //[OUT] The number of bytes transmitted/received.
uint16_t    transfer_mode ;
                    // 0x00:  For uart_get_line function, transfer without
                    // termination.
                    // For uart_put_line function, transfer without termination.
                    // 0x01:  For uart_get_line function, stop transfer when
                    // <CR><LF> are received.
                    // For uart_put_line function, transfer is stopped after
                    // reaching \0. <CR><LF> characters are sent out after that.
                    // 0x02:  For uart_get_line function, stop transfer when <LF>
```

```
                    // is received.
                    // For uart_put_line function, transfer is stopped after
                    // reaching \0. A <LF> character is sent out after that.
                    // 0x03:  For uart_get_line function, RESERVED.
                    // For uart_put_line function, transfer is stopped after
                    // reaching \0.
                    // NOTE: if (transfer_mode & 0x0F) != 0, transfer also stops
                    // when all data in buffer has been transferred.
uint8_t   driver_mode;
                    // 0x00: Polling mode, function is blocked until transfer is
                    // finished.
                    // 0x01: Interrupt mode, function exit immediately, callback
                    //function is invoked when transfer is finished.
                    // 0x02: DMA mode (transfer_mode must be 0).
                    // DMA req function is called for Uart DMA channel setup, then
                    // DMA ISR indicate that transfer is finished.
uint8_t dma_num;    //DMA channel number in case DMA mode is enabled
UART_CALLBK_T     callback_func_pt;
                    // callback function
                    // In case DMA mode is enabled, callback function is invoked
                    // after transfer. If callback_func_pt = NULL, no DMA interrupt
                    // is issued for this UART channel.
uint32_t        dma; //DMA handler
}  UART_PARAM_T  ;
```

36.4.10.4 CALLBK_T

```
typedef void  (*CALLBK_T) (uint32_t   res0, uint32_t res1 ) ;
     //define callback func TYPE
     //res0: error code
     //res1: number of bytes transferred
```

36.4.11 Functional description

36.4.11.1 Example (no DMA)

Send and receive characters in interrupt mode. Use the UART API as follows:

1. Assign the RXD and TXD functions to pins in the switch matrix and set up the system clock, main clock, and UART clock dividers.
2. Global defines:

```
#define rom_drivers_ptr (* (ROM **) 0x03000200)
UARTD_API_T *  pUartApi ; //define pointer to type API function addr table
UART_HANDLE_T* uart_handle_0; //handle to UART API

UART_PARAM_T    param;

#define RAMBLOCK_H  10
uint32_t  start_of_ram_block[ RAMBLOCK_H ] ;

#define BUFFER_SIZE   100
uint32_t uart_buffer[BUFFER_SIZE];
```

3. Define configuration structure and initialize pointer to the UART API:

```
UART_CONFIG_T   uart_set;
pUartApi = (UARTD_API_T *) (rom_drivers_ptr->pUARTD);
```

4. Define some characters to send:

```
const uint8_t pattern4[] = "Test interrupt mode";
```

5. Initialize memory for one UART API and create handle:

```
 size_in_bytes =  pUartApi->uart_get_mem_size() ;
  if ( RAMBLOCK_H < (size_in_bytes /4 ) ) {
    return 1;
  }
uart_handle_0 = pUartApi->uart_setup(LPC_UART0_BASE, (uint8_t
      *)start_of_ram_block);
```

6. Initialize UART, configure baud rate. Use the result of the UART initialization to configure the fractional baud rate generator register FRGCTRL in the SYSCON block (the lower 8 bits of this register must always be set to 0xFF):

```
uart_set.sys_clk_in_hz = SystemCoreClock/4;
uart_set.baudrate_in_hz = BAUDRATE_IN_HZ;
uart_set.config = 1;// 8 bits data, no parity, 1 stop
uart_set.sync_mod = 0;
uart_set.error_en = 0;
LPC_SYSCON->UARTFRGCTRL = (pUartApi->uart_init(uart_handle_0,
     &uart_set)<<0x08)|0xFF;
```

7. Enable the UART interrupt in the NVIC.

8. Set up the UART parameter structure UART_PARAM_T:

```
param.driver_mode = 1; //INT mode
param.transfer_mode = 1; //stop transfer with \0,, <CR><LF> is sent out.
param.buffer = (uint8_t *)pattern4;
param.size = 100; //Max of buffer
param.callback_func_pt = put_callback;
```

9. Define the receive and transmit callback functions invoked when the transfer has finished:

```
void  get_callback(uint32_t   err_code, uint32_t n ) {
  get_tag = 1;
  if (err_code != LPC_OK)
    while(1);
}

void  put_callback(uint32_t   err_code, uint32_t n ) {
  put_tag = 1;
  if (err_code != LPC_OK)
    while(1);
}
```

10. Send some characters and stop transfer with \0. Then <CR><LF> is sent out.

```
param.driver_mode = 1; //INT mode
```

```
param.transfer_mode = 1; //stop transfer with \0,, <CR><LF> is sent out.
param.buffer = (uint8_t *)pattern4;
param.size = 100; //Max of buffer
param.callback_func_pt = put_callback;
put_tag = 0;
pUartApi->uart_put_line(uart_handle, &param);
while(!put_tag);
```

11. Read five characters until buffer is full:

```
param.transfer_mode = 0; //stop get when buffer is full
param.buffer = (uint8_t *)buffer;
param.size = 5; //size of buffer
param.callback_func_pt = get_callback;
get_tag = 0;
pUartApi->uart_get_line(uart_handle, &param);
while(!get_tag);
```

37.1 How to read this chapter

The DMA ROM driver routines are available on all parts.

37.2 Features

- DMA set-up
- DMA channel control
- DMA transfers

37.3 General description

The DMA API handles DMA set-up and transfers.

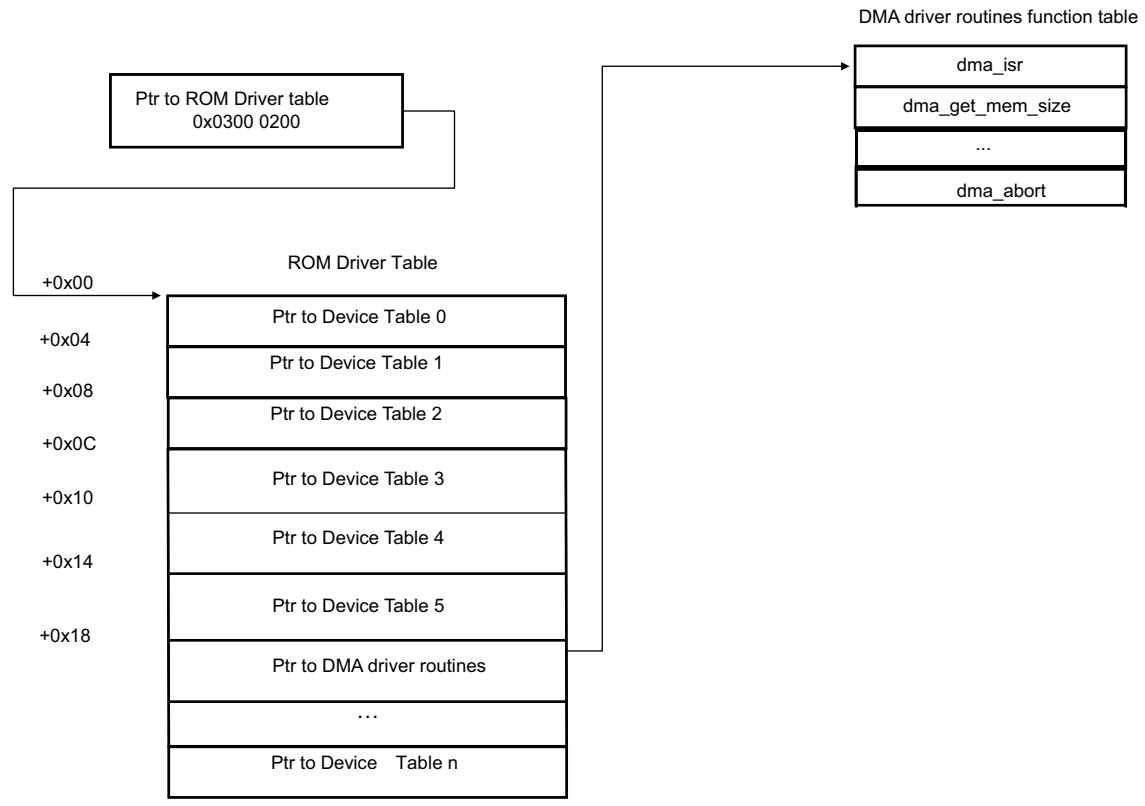


Fig 98. DMA driver routines pointer structure

37.4 API description

The DMA API contains functions set up and operate the DMA controller.

Table 531. DMA API calls

API call	Description	Reference
Interrupt service		
`void dma_isr(DMA_HANDLE_T* handle);`	Interrupt service routine	Table 540
Initialization		
`uint32_t dma_get_mem_size(void) ;`	Get memory size needed for DMA.	Table 532
`DMA_HANDLE_T* dma_setup(uint32_t base_addr, uint8_t *ram);`	Set up DMA.	Table 533
DMA channel operations		
`uint32_t dma_init(DMA_HANDLE_T* handle, DMA_CHANNEL_T *channel, DMA _TASK_T *task);`	Enable DMA channel and set up basic DMA transfer.	Table 534
`uint32_t dma_task_link(DMA_HANDLE_T* handle, DMA _TASK_T *task, uint8_t valid);`	Create linked transfer.	Table 535
`void dma_set_valid(DMA_HANDLE_T* handle, unit8_t chl_num);`	Set a task to valid.	Table 536
`void dma_pause(DMA_HANDLE_T* handle, unit8_t chl_num);`	Pause DMA transfer on one channel.	Table 537
`void dma_unpause(DMA_HANDLE_T* handle, unit8_t chl_num);`	Resume DMA transfer.	Table 538
`void dma_abort(DMA_HANDLE_T* handle, unit8_t chl_num);`	Cancel DMA transfer on one channel.	Table 539

The following structure must be defined to use the DMA API:

```
typedef struct DMAD_API {   // index of all the DMA driver functions
  void               (*dma_isr)(DMA_HANDLE_T*  handle);
  uint32_t          (*dma_get_mem_size)( void);
  DMA_HANDLE_T* (*dma_setup)( uint32_t base_addr,  uint8_t *ram );
  ErrorCode_t   (*dma_init)( DMA_HANDLE_T*  handle, DMA_CHANNEL_T *channel, DMA_TASK_T
     *task);
  ErrorCode_t   (*dma_link)( DMA_HANDLE_T* handle, DMA_TASK_T *task, uint8_t valid);
  ErrorCode_t   (*dma_set_valid)( DMA_HANDLE_T* handle, uint8_t chl_num);
  ErrorCode_t   (*dma_pause)( DMA_HANDLE_T* handle, uint8_t chl_num);
  ErrorCode_t   (*dma_unpause)( DMA_HANDLE_T* handle, uint8_t chl_num);
  ErrorCode_t   (*dma_abort)( DMA_HANDLE_T* handle, uint8_t chl_num);
} DMAD_API_T  ;
```

37.4.1 DMA get memory size

Table 532. dma_get_mem_size

Routine	dma_get_mem_size
Prototype	`uint32_t dma_get_mem_size(void);`
Input parameter	None.
Return	Memory size in bytes.
Description	The memory size for the DMA instance.

37.4.2 DMA set-up

Table 533. dma_setup

Routine	dma_setup
Prototype	`DMA_HANDLE_T* dma_setup(uint32_t base_addr, uint8_t *ram) ;`
Input parameter	base_addr: Base address of register for DMA block. ram: Pointer to the memory space for the DMA Channel descriptor map used by the DMA instance. The size is obtained by the dma_get_mem_size() function.
Return	0: the alignment of address for DMA descriptor map is not correct. Others: The handle to corresponding DMA instance.
Description	Sets up DMA instance with provided memory. Checks the alignment of address for Channel descriptor map according to the number of channels and returns the handle of this instance if the address alignment is correct.

37.4.3 DMA init

After the handler is initialized, the DMA channel API is invoked to set up a channel for data transfer.

Table 534. dma_init

Routine	dma_init
Prototype	`uint32_t dma_init(DMA_HANDLE_T* handle, DMA_CHANNEL_T *channel, DMA _TASK_T *task);`
Input parameter	handle: The handler to the DMA instance. channel: The pointer to the structure for DMA channel setup. task: The pointer to the structure for basic transfer task setup.
Return	Error code.
Description	Enables the DMA channel and sets up a basic transfer task. If no further DMA channel operation API is invoked, a single buffer DMA transfer is performed with DMA request or trigger.

37.4.4 DMA link

Table 535. dma_link

Routine	dma_link
Prototype	`uint32_t dma_task_link(DMA_HANDLE_T* handle, DMA _TASK_T *task, uint8_t valid);`
Input parameter	handle: The handler to the DMA instance. task: The pointer to the structure for transfer task setup. Valid: valid status of task 0: The task is not enabled for DMA, calling dma_task_valid is needed to process this task. 1: The task is valid for DMA, DMA will process this task in case request or trigger is fulfilled.
Return	Error code.
Description	Link an additional transfer task to the DMA channel enabled previously by calling dma_setup.

37.4.5 DMA set valid transfer

Table 536. dma_set_valid

Routine	dma_set_valid
Prototype	`void dma_set_valid(DMA_HANDLE_T* handle, unit8_t chl_num);`
Input parameter	handle: The handler to the DMA instance. chl_num: DMA channel number to be enabled.
Return	None
Description	If the DMA fetches an invalid transfer task, DMA will not process this task until this function is called. If the DMA is transferring date for a valid task, calling this function will make next invalid task served immediately when it is fetched by the DMA.

37.4.6 DMA pause transfer

Table 537. dma_pause

Routine	dma_pause
Prototype	`void dma_pause(DMA_HANDLE_T* handle, unit8_t chl_num);`
Input parameter	handle: The handler to the DMA instance. chl_num: DMA channel number to be paused.
Return	None
Description	Pauses one DMA channel transfer.

37.4.7 DMA resume transfer

Table 538. dma_unpause

Routine	dma_unpause
Prototype	`void dma_unpause(DMA_HANDLE_T* handle, unit8_t chl_num);`
Input parameter	handle: The handler to the DMA instance. chl_num: DMA channel number to be resumed.
Return	None
Description	Resume one DMA channel transfer that has been paused previously.

37.4.8 DMA abort transfer

Table 539. dma_abort

Routine	dma_abort
Prototype	`void dma_abort(DMA_HANDLE_T* handle, unit8_t chl_num);`
Input parameter	handle: The handler to the DMA instance. chl_num: DMA channel number to be aborted.
Return	None
Description	Cancel one DMA channel transfer tasks. Recovering is impossible.

37.4.9 DMA interrupt service routine

Table 540. dma_isr

Routine	dma_isr
Prototype	`void dma_isr(DMA_HANDLE_T* handle);`
Input parameter	handle: The handler to the DMA instance.
Return	None.
Description	DMA interrupt service routine.

37.4.10 Error codes

Table 541. Error codes

Return code	Error Code	Description
0x000D 0001	ERR_DMA_ERROR_INT	-
0x000D 0002	ERR_DMA_CHANNEL_NUMBER	-
0x000D 0003	ERR_DMA_CHANNEL_DISABLED	-
0x000D 0004	ERR_DMA_BUSY	-
0x000D 0005	ERR_DMA_NOT_ALIGNMENT	-
0x000D 0006	ERR_DMA_PING_PONG_EN	Reload bit already set causing ping-pong mode error
0x000D 0007	ERR_DMA_CHANNEL_VALID_PENDING	-

```
typedef enum
{
 ERR_DMA_BASE = 0x000D0000,
  /*0x000D0001*/ ERR_DMA_ERROR_INT=ERR_DMA_BASE+1,
  /*0x000D0002*/ ERR_DMA_CHANNEL_NUMBER,
  /*0x000D0003*/ ERR_DMA_CHANNEL_DISABLED,
  /*0x000D0004*/ ERR_DMA_BUSY,
  /*0x000D0005*/ ERR_DMA_NOT_ALIGNMENT,
  /*0x000D0006*/ ERR_DMA_PING_PONG_EN,
  /*0x000D0007*/ ERR_DMA_CHANNEL_VALID_PENDING
} ErrorCode_t;
```

37.4.11 DMA ROM driver variables

37.4.11.1 DMA_CHANNEL_T channel configuration structure

```
typedef struct  DMA_CHANNEL {
     uint8_t event;        // event type selection for DMA transfer
          //0: software request
          //1: peripheral request
          //2: hardware trigger
          //others: reserved
     uint8_t hd_trigger; //In case hardware trigger is enabled, the trigger burst is
     // set up here.
     //Rising edge triggered is fixed.
     //bit0~bit3: burst size
          //0: burst size =1, 1: 21, 2: 22… 10: 1024, others: reserved.
     //bit4: Source Burst Wrap
          //0: Source burst wrapping is not enabled
```

```
            //1: Source burst wrapping is enabled
        //bit5: Destination Burst Wrap
            //0: Destination burst wrapping is not enabled
            //1: Destination burst wrapping is enabled
        //bit6: Trigger Burst
            //0: Hardware trigger cause a single transfer
            //1: Hardware trigger cause a burst transfer
        //bit7: reserved
    uint8_t priority;       //priority level
        //0 -> 7: Highest priority ->  Lowest priority.
        //other: reserved.
    uint8_t reserved0;
    CALLBK_T     cb_func; // callback function, Callback function is only
        // invoked when INTA or INTB is enabled.
    }  DMA_CHANNEL_T  ;
```

37.4.11.2 DMA_HANDLE_T

The handler to the instance of DMA driver. This handle is created by Init API and used by the other function in the DMA driver.

```
typedef  void     DMA_HANDLE_T   ;    // define TYPE for DMA handle pointer
```

37.4.11.3 DMA_TASK_T

```
typedef struct  DMA_ TASK {
    uint8_t ch_num  // DMA channel number.
    uint8_t config;     //configuration of this task
        //bit0: Ping_Pong transfer
            //0: Not Ping_Pong transfer
            //1: Linked with previous task for Ping_Pong transfer
        //bit1: Software Trigger
            //0: the trigger for this channel is not set.
            //1: the trigger for this channel is set immediately.
        //bit2:  Clear Trigger
            //0: The trigger is not cleared when this task is finished.
            //1: The trigger is cleared when this task is finished.
        //bit3:  Select INTA
            //0: No IntA.
            //1: The IntB flag for this channel will be set when this task is
            // finished.
        //bit4:  Select INTB
            //0: No IntB.
            //1: The IntB flag for this channel will be set when this task is
    finished.
        //bit5~bit7: reserved
    uint8_t   data_type;
        //bit0~bit1: Data width. 0: 8-bit, 1: 16-bit, 2: 32-bit, 3: reserved
        //bit2~bit3: How is source address incremented?
            //0: The source address is not incremented for each transfer.
            //1: The source address is incremented by the amount specified by
            // Width for each transfer.
            //2: The source address is incremented by 2 times the amount specified
```

```
                    // by Width for each transfer.
                    //3: The source address is incremented by 4 times the amount specified
                    // by Width for each transfer.
               //bit4~bit5: How is the destination address incremented?
                    //0: The destination address is not incremented for each transfer.
                    //1: The destination address is incremented by the amount specified by
                    // Width for each transfer.
                    //2: The destination address is incremented by 2 times the amount
                    // specified by Width for each transfer.
                    //3: The destination address is incremented by 4 times the amount
                    // specified by Width for each transfer.
               //bit6~bit7: reserved.
          uint8_t reserved0;
          uint16_t  data_length; //0: 1 transfer, 1: 2 transfer, … 1023: 1024 transfer.
               //Others: reserved.
          uint16_t reserved1;
          uint32_t  src;      // Source data end address
          uint32_t  dst;      // Destination end address
          uint32_t task_addr;    //the address of RAM for saving this task.
               //(NOTE: each task need 16 bytes RAM for storing configuration,
               //  and DMA API could set it according user input parameter,
               //  but it is responsible of user to allocate this RAM space and
               //  make sure that the base address must be 16-byte alignment.
               //  And if user has setup the next_tast(!=0), the dma_task_link
               //  must be called for this task setup, otherwise unpredictable error will
               // happen.)
     }  DMA_TASK_T ;
```

37.4.11.4 CALLBK_T

```
typedef void  (*CALLBK_T) (uint32_t   res0, uint32_t res1 ) ;
     //define callback func TYPE
     //res0: error code
     //res1: : 0 = INTA is issued, 1 = INTB is issued
```

38.1 How to read this chapter

The I2C-bus ROM API is available on all parts.

38.2 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.

38.3 General description

The drivers are callable for use by any application program to send or receive data on the I2C bus. With the I2C drivers it is easy to produce working projects using the I2C interface.

The ROM routines allow the 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.

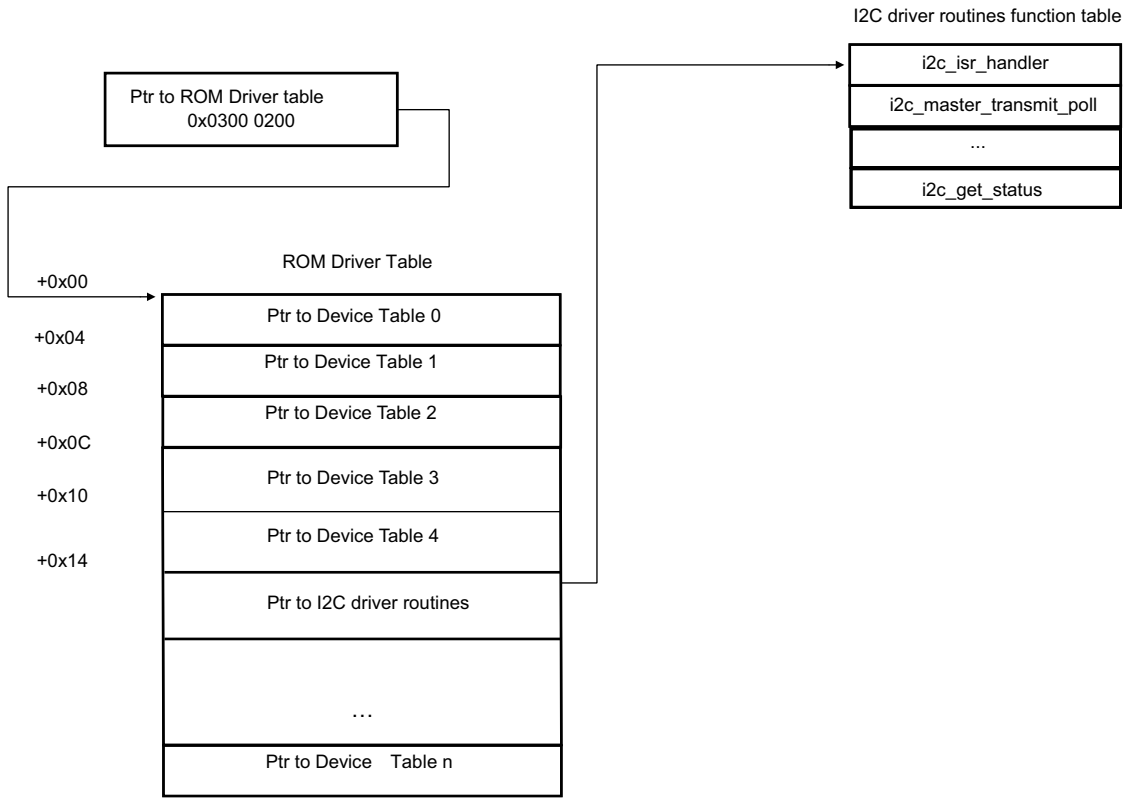


Fig 99. I2C-bus driver routines pointer structure

38.4 API description

The I2C API contains functions to configure the I2C and send and receive data in master and slave modes.

Table 542. I2C API calls

API call	Description	Reference
`void i2c_isr_handler(I2C_HANDLE_T*);`	I2C ROM Driver interrupt service routine.	Table 543
Master functions		
`ErrorCode_t i2c_master_transmit_poll(I2C_HANDLE_T*, I2C_PARAM*, I2C_RESULT*);`	I2C Master Transmit Polling	Table 544
`ErrorCode_t i2c_master_receive_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Master Receive Polling	Table 545
`ErrorCode_t i2c_master_tx_rx_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Master Transmit and Receive Polling	Table 546
`ErrorCode_t i2c_master_transmit_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Master Transmit Interrupt	Table 547
`ErrorCode_t i2c_master_receive_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Master Receive Interrupt	Table 548

Table 542. I2C API calls

API call	Description	Reference
`ErrorCode_t i2c_master_tx_rx_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Master Transmit Receive Interrupt	Table 549
Slave functions		
`ErrorCode_t i2c_slave_receive_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Slave Receive Polling	Table 550
`ErrorCode_t i2c_slave_transmit_poll(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Slave Transmit Polling	Table 551
`ErrorCode_t i2c_slave_receive_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Slave Receive Interrupt	Table 552
`ErrorCode_t i2c_slave_transmit_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`	I2C Slave Transmit Interrupt	Table 553
`ErrorCode_t i2c_set_slave_addr(I2C_HANDLE_T*, slave_addr_0_3, slave_mask_0_3);`	I2C Set Slave Address	Table 554
Set-up functions		
`uint32_t i2c_get_mem_size(void)`	I2C Get Memory Size	Table 555
`I2C_HANDLE_T* i2c_setup(i2c_base_addr, *start_of_ram);`	I2C Setup	Table 556
`ErrorCode_t i2c_set_bitrate(I2C_HANDLE_T*, P_clk_in_hz, bitrate_in_bps);`	I2C Set Bit Rate	Table 557
`uint32_t i2c_get_firmware_version(void );`	I2C Get Firmware Version	Table 558
`I2C_MODE_T i2c_get_status(I2C_HANDLE_T* );`	I2C Get Status	Table 559
`ErrorCode_t i2c_set_timeout(I2C_HANDLE_T* h_i2c, uint32_t timeout);`	I2C time-out value	Table 560

The following structure has to be defined to use the I2C API:

```
typedef struct I2CD_API {   // index of all the i2c driver functions
void (*i2c_isr_handler) (I2C_HANDLE_T* h_i2c) ; // ISR interrupt service request
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 ) ;
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);
```

```
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 ;
```

38.4.1 ISR handler

Table 543. ISR handler

Routine	ISR handler
Prototype	`void i2c_isr_handler(I2C_HANDLE_T*);`
Input parameter	I2C_HANDLE_T:Handle to the I2C instance.
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.

38.4.2 I2C Master Transmit Polling

Table 544. 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 I2C instance. 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.

38.4.3 I2C Master Receive Polling

Table 545. 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 I2C instance. 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.

38.4.4 I2C Master Transmit and Receive Polling

Table 546. 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*);`
Input parameter	I2C_HANDLE_T: Handle to the I2C instance. 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.

38.4.5 I2C Master Transmit Interrupt

Table 547. 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 I2C instance. 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.

38.4.6 I2C Master Receive Interrupt

Table 548. 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 I2C instance. 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.

38.4.7 I2C Master Transmit Receive Interrupt

Table 549. 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 I2C instance. 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.

38.4.8 I2C Slave Receive Polling

Table 550. 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 I2C instance. 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.

38.4.9 I2C Slave Transmit Polling

Table 551. 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 I2C instance. 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.

38.4.10 I2C Slave Receive Interrupt

Table 552. I2C Slave Receive Interrupt

Routine	I2C Slave Receive Interrupt
Prototype	`ErrorCode_t i2c_slave_receive_intr(I2C_HANDLE_T* , I2C_PARAM* , I2C_RESULT*);`
Input parameter	I2C_HANDLE_T: Handle to the I2C instance. 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.

38.4.11 I2C Slave Transmit Interrupt

Table 553. 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 I2C instance. 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.

38.4.12 I2C Set Slave Address

Table 554. 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 I2C instance. 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.

38.4.13 I2C Get Memory Size

Table 555. I2C Get Memory Size

Routine	I2C Get Memory Size
Prototype	`uint32_t i2c_get_mem_size(void);`

Table 555. I2C Get Memory Size

Routine	I2C Get Memory Size
Input parameter	None.
Return	uint32.
Description	Returns the number of bytes in SRAM needed by the I2C driver.

38.4.14 I2C Set-up

Table 556. I2C Setup

Routine	I2C Setup
Prototype	`I2C_HANDLE_T* i2c_setup(i2c_base_addr, *start_of_ram);`
Input parameter	I2C_base addr: unint32 variable. Base address for I2C peripherals. Start_of_ram: unint32 pointer. Pointer to allocated SRAM.
Return	Handle to the I2C instance.
Description	Returns a handle to the I2C instance.

38.4.15 I2C Set Bit Rate

Table 557. 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 I2C instance. 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).

38.4.16 I2C Get Firmware Version

Table 558. 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.

38.4.17 I2C Get Status

Table 559. 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 I2C instance.
Return	Status code.
Description	Returns status code. The status code indicates the state of the I2C bus. Refer to I2C Status Code Table.

38.4.18 I2C time-out value

Table 560. I2C time-out value

Routine	I2C time-out value
Prototype	`ErrorCode_t i2c_set_timeout(I2C_HANDLE_T*  h_i2c, uint32_t timeout);`
Input parameter	I2C_HANDLE_T: Handle to the I2C instance. uint32_t time-out: time value is timeout*16 i2c function clock. If time-out = 0, time-out feature is disabled.
Return	Status code.
Description	Returns status code. The status code indicates the state of the I2C bus. Refer to I2C Status Code Table.

38.4.19 Error codes

Table 561. Error codes

Error Code	Description	Comment
0x0006 0001	ERR_I2C_NAK	-
0x0006 0002	ERR_I2C_BUFFER_OVERFLOW	-
0x0006 0003	ERR_I2C_BYTE_COUNT_ERR	-
0x0006 0004	ERR_I2C_LOSS_OF_ARBRITRATION	-
0x0006 0005	ERR_I2C_SLAVE_NOT_ADDRESSED	-
0x0006 0006	ERR_I2C_LOSS_OF_ARBRITRATION_NAK_BIT	-
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.
0x0006 0009	ERR_I2C_TIMEOUT	-
0x0006 000A	ERR_I2C_BUFFER_UNDERFLOW	-

```
typedef enum
{
 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,
  /*0x00060009*/ ERR_I2C_TIMEOUT,
  /*0x0006000A*/ ERR_I2C_BUFFER_UNDERFLOW
} ErrorCode_t;
```

38.4.20 I2C Status code

Table 562. I2C Status code

Status code	Description
0	IDLE
1	MASTER_SEND
2	MASTER_RECEIVE
3	SLAVE_SEND
4	SLAVE_RECEIVE

38.4.21 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.

38.4.21.1 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. The error code is updated in the callback and the parameter n indicates the number of bytes transferred.

38.4.22 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_PARAM { //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_T ;
```

The RESULT structure is as follows:

```
typedef struct I2C_RESULT{     // RESULTs struct--results are here when returned
     uint32_t n_bytes_sent ;
     uint32_t n_bytes_recd ;
} I2C_RESULT_T ;
```

38.4.23 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 ;
```

38.5 Functional description

38.5.1 I2C Set-up

Before calling any setup functions in the I2C ROM, the application program is responsible for doing 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 have to be set up:

1. Initialize pointer to the I2C API function table.
2. Declare the PARAM and RESULT struct.
3. Declare the transmit and receive buffer.

If interrupts are used, then additional driver variables have to 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.

38.5.2 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(LPC_I2C_BASE, (uint32_t *)start_of_ram_block0 );
error_code = pI2cApi->i2c_set_bitrate((I2C_HANDLE_T*)i2c_handle, PCLK_in_Hz,
     bps_in_hz);
```

38.5.3 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 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 monitored as well.

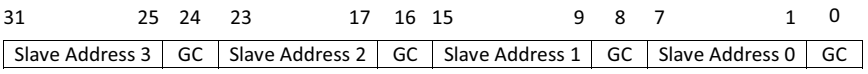


Fig 100. I2C slave mode set-up address packing

```
pI2cApi = ROM_DRIVERS_PTR->pI2CD; //setup I2C function table pointer
size_in_bytes = pI2cApi->i2c_get_mem_size();
i2c_handle = pI2cApi->i2c_setup(LPC_I2C_BASE, (uint32_t *)start_of_ram_block0 );
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) ;
```

38.5.4 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 38.4.22.
- I2C_RESULT* contains the results after the function has executed.

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 update 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.

38.5.5 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_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 38.4.22.
- I2C_RESULT is a containing the results after the function executes. Section 38.4.22.

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.

39.1 How to read this chapter

The ADC ROM driver routines are available on all parts.

39.2 Features

- ADC calibration
- Triggered ADC conversions
- ADC interrupt handling

39.3 General description

The ADC API handles the calibration and set-up of the ADC and allows the user to perform analog-to-digital conversion using the 12-bit ADC.

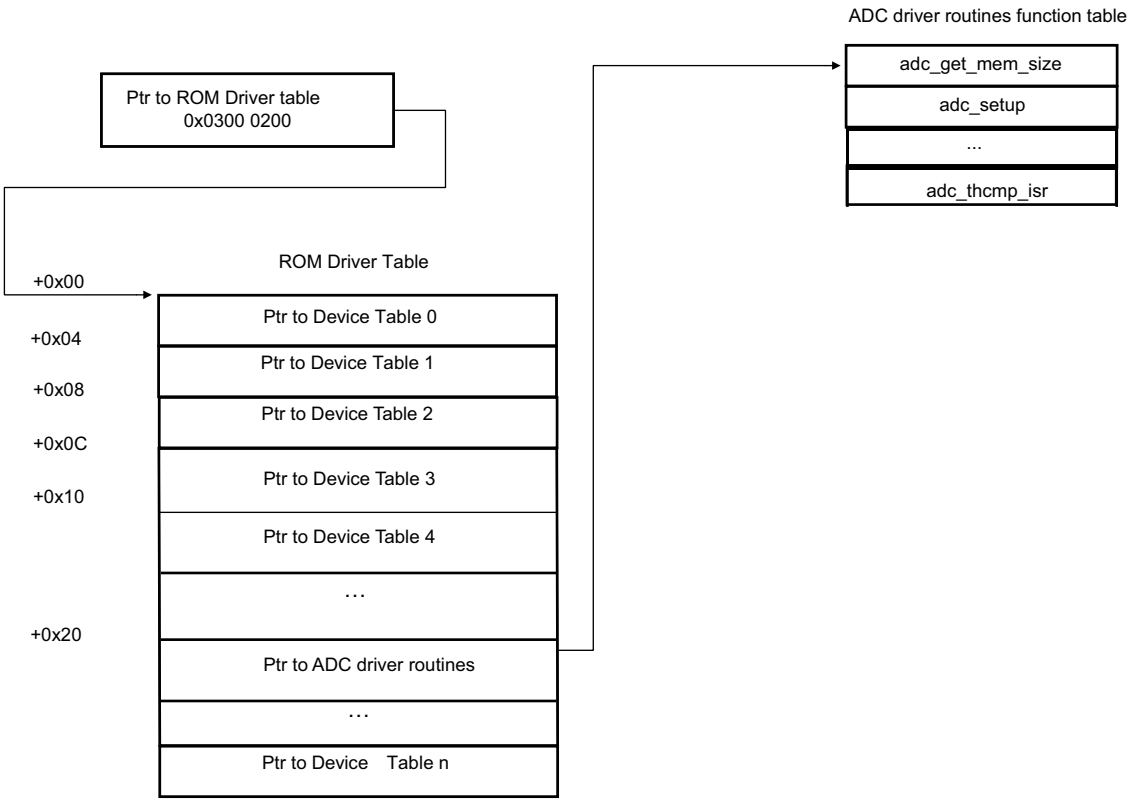


Fig 101. ADC driver routines pointer structure

39.4 API description

The ADC API contains functions set up and operate the 12-bit ADC.

Table 563. ADC API calls

API call	Description	Reference
ADC set-up		
`uint32_t ramsize_in_bytes adc_get_mem_size(void);`	Get memory size for one ADC instance	Table 564
`ADC_HANDLE_T* adc_setup (uint32_t base_addr, uint8_t *ram);`	Set-up ADC instance	Table 565
`void adc_calibration (ADC_HANDLE_T handle, ADC_CONFIG_T *adc_set);`	Calibrate ADC	Table 566
`void adc_init (ADC_HANDLE_T *handle, ADC_CONFIG_T adc_set);`	Set-up operation mode and enable ADC	Table 567
ADC operation		
`uint32_t adc_seqa_read (ADC_HANDLE_T* handle, ADC_PARAM_T param);`	Sequence A conversion	Table 568
`uint32_t adc_seqb_read (ADC_HANDLE_T* handle, ADC_PARAM_T param);`	Sequence B conversion	Table 569
Interrupt service		
`void adc_seqa_isr (ADC_HANDLE_T* handle);`	Sequence A interrupt service	Table 570
`void adc_seqb_isr (ADC_HANDLE_T* handle);`	Sequence B interrupt service	Table 571
`void adc_ovr_isr (ADC_HANDLE_T* handle);`	Overrun interrupt service	Table 572
`void adc_thcmp_isr (ADC_HANDLE_T* handle);`	Threshold compare interrupt service	Table 573

The following structure must be defined to use the ADC API:

```
typedef struct  ADCD_API {   // index of ADC driver functions
     uint32_t (*adc_get_mem_size)(void);
     ADC_HANDLE_T (*adc_setup)(uint32_t base_addr, uint8_t *ram);
     void (*adc_calibration)(ADC_HANDLE_T handle, ADC_CONFIG_T *set);
     void (*adc_init)(ADC_HANDLE_T handle, ADC_CONFIG_T *set);
     uint32_t (*adc_seqa_read)(ADC_HANDLE_T handle, ADC_PARAM_T * param);
     uint32_t (*adc_seqb_read)(ADC_HANDLE_T handle, ADC_PARAM_T * param);
     //--interrupt functions--//
     void (*adc_seqa_isr)(ADC_HANDLE_T handle);
     void (*adc_seqb_isr)(ADC_HANDLE_T handle);
     void (*adc_ovr_isr)(ADC_HANDLE_T handle);
     void (*adc_thcmp_isr)(ADC_HANDLE_T handle);
} ADCD_API_T  ;
```

39.4.1 ADC get memory size

Table 564. adc_get_mem_size

Routine	adc_get_mem_size		
Prototype	`uint32_t`	`ramsize_in_bytes`	`adc_get_mem_size(void);`
Input parameter	None.		
Return	Memory size in bytes.		
Description	The memory size for one ADC instance.		

39.4.2 ADC set-up

Table 565. adc_setup

Routine	adc_setup
Prototype	`ADC_HANDLE_T* adc_setup (uint32_t base_addr, uint8_t *ram);`
Input parameter	base_addr: Base address of register for the ADC block. ram: Pointer to the memory space for ADC instance; the size is obtained from adc_get_mem_size() function.
Return	The handle to corresponding ADC instance.
Description	Sets up the ADC instance with provided memory and the base address of the ADC instance, and returns the handle to this instance.

39.4.3 ADC calibration

Remark: Calibrate the ADC using this function before performing conversions.

Table 566. adc_calibration

Routine	adc_calibration
Prototype	`void adc_calibration (ADC_HANDLE_T handle, ADC_CONFIG_T *adc_set);`
Input parameter	handle: The handler of ADC driver from adc_setup(). adc_set: Configuration for ADC operation.
Return	None.
Description	Calibrate ADC controller before it can be used. See ADC_CONFIG_T structure defined below in adc_init(). Only the system clock and the ADC clock are needed for ADC calibration..

39.4.4 ADC initialize

The ADC_CONFIG_T structure defines the configuration settings for this function. See Section 39.4.12.1 “ADC_CONFIG_T channel configuration structure”.

Table 567. adc_init

Routine	adc_init
Prototype	`void adc_init (ADC_HANDLE_T *handle, ADC_CONFIG_T adc_set);`
Input parameter	handle: The handle to the ADC instance. adc_set: Configuration for ADC operation.
Return	None.
Description	Setup operation mode for ADC, then enable ADC.

39.4.5 ADC start sequence A conversion

Table 568. adc_seqa_read

Routine	adc_seqa_read
Prototype	`uint32_t adc_seqa_read (ADC_HANDLE_T* handle, ADC_PARAM_T param);`

Table 568. adc_seqa_read

Routine	adc_seqa_read
Input parameter	handle: The handle to the ADC instance. param: Pointer to the ADC_PARAM_T structure.
Return	Error code.
Description	Start ADC conversion using sequence A.

39.4.6 ADC start sequence B conversion

Table 569. adc_seqb_read

Routine	adc_seqb_read
Prototype	`uint32_t adc_seqb_read (ADC_HANDLE_T* handle, ADC_PARAM_T param);`
Input parameter	handle: The handle to the ADC instance. param: Pointer to the ADC_PARAM_T structure.
Return	Error code.
Description	Start ADC conversion using sequence B.

39.4.7 ADC service sequence A interrupt

Table 570. adc_seqa_isr

Routine	adc_seqa_isr
Prototype	`void adc_seqa_isr (ADC_HANDLE_T* handle);`
Input parameter	handle: The handle to the ADC instance.
Return	None.
Description	ADC interrupt service routine for sequence A interrupt. When using this function, the corresponding ADC interrupt must be enabled. This function is invoked by the user ISR.

39.4.8 ADC service sequence B interrupt

Table 571. adc_seqb_isr

Routine	adc_seqb_isr
Prototype	`void adc_seqb_isr (ADC_HANDLE_T* handle);`
Input parameter	handle: The handle to the ADC instance.
Return	None.
Description	ADC interrupt service routine for sequence B interrupt. When using this function, the corresponding ADC interrupt must be enabled. This function is invoked by the user ISR.

39.4.9 ADC service overrun error interrupt

Table 572. adc_ovr_isr

Routine	adc_ovr_isr
Prototype	`void adc_ovr_isr (ADC_HANDLE_T* handle);`

Table 572. adc_ovr_isr

Routine	adc_ovr_isr
Input parameter	handle: The handle to the ADC instance.
Return	None.
Description	ADC interrupt service routine for overrun error interrupt. When using this function, the corresponding ADC interrupt must be enabled. This function is invoked by the user ISR.

39.4.10 ADC service threshold compare interrupt

Table 573. adc_thcmp_isr

Routine	adc_thcmp_isr
Prototype	`void adc_thcmp_isr (ADC_HANDLE_T* handle);`
Input parameter	handle: The handle to the ADC instance.
Return	None.
Description	ADC interrupt service routine for threshold compare interrupt. When using this function, the corresponding ADC interrupt must be enabled. This function is invoked by the user ISR.

39.4.11 Error codes

Table 574. Error codes

Return code	Error Code	Description
0x000F 0001	ERR_ADC_OVERRUN	
0x000F 0002	ERR_ADC_INVALID_CHANNEL	
0x000F 0003	ERR_ADC_INVALID_SEQUENCE	
0x000F 0004	ERR_ADC_INVALID_SETUP	
0x000F 0005	ERR_ADC_PARAM	
0x000F 0006	ERR_ADC_INVALID_LENGTH	
0x000F 0007	ERR_ADC_NO_POWER	

```
typedef enum
{
 ERR_ADC_BASE = 0x000F0000,
  /*0x000F0001*/ ERR_ADC_OVERRUN=ERR_ADC_BASE+1,
  /*0x000F0002*/ ERR_ADC_INVALID_CHANNEL,
  /*0x000F0003*/ ERR_ADC_INVALID_SEQUENCE,
  /*0x000F0004*/ ERR_ADC_INVALID_SETUP,
  /*0x000F0005*/ ERR_ADC_PARAM,
  /*0x000F0006*/ ERR_ADC_INVALID_LENGTH,
  /*0x000F0007*/ ERR_ADC_NO_POWER
} ErrorCode_t;
```

39.4.12 ADC ROM driver variables

39.4.12.1 ADC_CONFIG_T channel configuration structure

```
typedef  struct {
```

```
    uint32_t system_clock;// System clock
    uint32_t adc_clock;// ADC clock
    uint32_t async_mode;
    uint32_t tenbit_mode;
    uint32_t lpwr_mode;
    uint32_t input_sel;
    uint32_t seqa_ctrl;
    uint32_t seqb_ctrl;
    uint32_t thrsel;
    uint32_t thr0_low;
    uint32_t thr0_high;
    uint32_t thr1_low;
    uint32_t thr1_high;
    uint32_t error_en;
    uint32_t thcmp_en;
    uint32_t channel_num;
} ADC_CONFIG_T;
```

The variables in this structure are defined as follows:

`system_clock`: Frequency of the system clock generated by the SYSCON block in Hz.

`adc_clock`: Frequency of the ADC clock in Hz. This is the clock rate for analog-to-digital conversions. Maximum clock rate is 50 MHz for 12-bit resolution.

`async_mode`: 0 selects synchronous mode, 1 selects asynchronous mode. See Table 433.

`tenbit_mode`: 0 disables 10-bit mode, 1 enables fast-conversion, 10-bit mode. See Table 433.

`lpwr_mode`: 0 disables low-power mode, 1 enables low-power mode. See Table 433.

`inp_sel`: selects input for channel 0. See Table 434.

0x0 = ADCn_0 pin.

0x1 = Regulator voltage output.

0x2 = Internal voltage reference.

0x3 = Temperature sensor.

0x4 = VDDA/2.

`seqa_ctrl`: Register content of the SEQA_CTRL register. See Table 435. Ensure that the START and SEQA_EN bits are always set to 0.

`seqb_ctrl`: Register content of the SEQB_CTRL register. See Table 435. Ensure that the START and SEQB_EN bits are always set to 0.

`thrsel`: Register content of the CHAN_THRSEL register. See Table 444.

`thr0_low`: Low threshold 0 value.

`thr0_high`: High threshold 0 value.

`thr1_low`: Low threshold 1 value.

`thr1_high`: High threshold 1 value.

`error_en`: 0 disables the overrun interrupt, 1 enables the overrun interrupt.

`thcmp_en`: Value of bits 26:3 in the INTEN register. See Table 445. Each pair of bits controls the threshold comparison interrupt for one selected channel.

channel_num: The highest channel number used in sequence A or sequence B. If this number is lower than the total available number of ADC channels, only the first channel_num channels can be used in either sequence A or sequence B.

39.4.12.2 ADC_HANDLE_T

This structure is the handle to one instance of the ADC driver. Each ADC has one handle. This handle is created by the init API.

```
typedef  void     ADC_HANDLE_T   ;    // define TYPE for ADC handle pointer
```

39.4.12.3 ADC_DMA_CFG_T

This structure sets up the DMA channel used for DMA transfer. The transfer can use a hardware trigger which is selected in the DMA_ITRIG_INMUX register for the selected channel. See Table 132 for the trigger number.

The DMA transfer is configured using the DMA API. After setting up the DMA transfer, a handle is returned and passed to the ADC in this structure.

```
typedef struct{
     uint32_t dma_adc_num;// DMA channel used for ADC data peripheral to memory
                          //transfer
     uint32_t dma_pinmux_num;// H/W trigger number.
     uint32_t dma_handle;// DMA handle passed to ADC
     ADC_CALLBK_T dma_done_callback_pt; // DMA completion callback function
} ADC_DMA_CFG_T;
```

39.4.12.4 ADC_PARAM_T

It's important that some of the parameters in ADC_CONFIG_T such as sequence control register setting and channel numbers needs to be set up before ADC_PARAM_T can be passed to the SEQA and SEQB read routines.

```
typedef struct {// params passed to adc driver function
     uint32_t          *buffer;
     // Considering supporting DMA and non-DMA mode, 32-bit buffer is needed for DMA
     uint32_t          driver_mode;
     // 0x00: Polling mode, function is blocked until transfer is finished.
     // 0x01: Interrupt mode, function exit immediately, callback function is invoked
     // when transfer is finished.
     // 0x02: DMA mode, in case DMA block is available, data transferred by ADC is
     // processed by DMA,
     //       and max buffer size is the
     //total number ADC channels, DMA req function is called for ADC DMA
     //       channel setup, then SEQx
     //completion also used as DMA callback function when that ADC conversion/DMA
     //transfer is finished.
     uint32_t          seqa_hwtrig;// H/W trigger for sequence A
     uint32_t          seqb_hwtrig;// H/W trigger for sequence B
     ADC_CONFIG_T      *adc_cfg;
     uint32_t          comp_flags;
     uint32_t          overrun_flags;
     uint32_t          thcmp_flags;
```

```
    ADC_DMA_CFG_T     *dma_cfg;
    ADC_SEQ_CALLBK_T  seqa_callback_pt;// SEQA callback function/the same callback
    //on DMA completion if DMA is used for ADCx.
    ADC_SEQ_CALLBK_T  seqb_callback_pt;// SEQb callback function/the same callback
    //on DMA completion if DMA is used for ADCx.
    ADC_CALLBK_T      overrun_callback_pt;// Overrun callback function
    ADC_CALLBK_T      thcmp_callback_pt;// THCMP callback function
    ADC_DMA_SETUP_T   dma_setup_func_pt;// ADC DMA channel setup function
} ADC_PARAM_T;
```

The following variables and pointers are used in this structure:

`seqa_buffer`: ADC buffer for the result of the conversion. Two separate buffers are allocated for both SEQA and SEQB. To support DMA mode, the ADC buffer size is set to 32-bit. Once DMA is completed, only bits 4 to 15 of the buffer contain the result of the ADC conversion. For non-DMA mode, the buffers contain the actual data from the conversion.

`seqb_buffer`: See description of `seqa_buffer`.

`driver_mode`: Configures the API function:

0x00: Polling mode, function is blocked until transfer is finished.

0x01: Interrupt mode. Function exits immediately and callback function is invoked when ADC conversion is finished.

0x02: DMA mode. ADC data conversion is handled by DMA. The maximum DMA buffer size is the total number ADC channels. The DMA API function is called for ADC DMA channel setup.

`seqa_hwtrig`: Select the hardware trigger for sequence A conversion. See Section 28.3.3 "ADC hardware trigger inputs" for potential triggers sources.

`seqb_hwtrig`: Select the hardware trigger for sequence A conversion. See Section 28.3.3 "ADC hardware trigger inputs" for potential triggers sources.

`comp_flags`: For each channel with a completed conversion, the corresponding bit is set.

`thcmp_flags`: For each channel incurring an overrun or threshold compare interrupt, the corresponding bit is set.

39.4.12.5 Callback functions

```
typedef void  (*ADC_SEQ_CALLBK_T) (ADC_HANDLE_T handle);
```

`handle`:The handle to the ADC instance.

The following callback function is invoked in the ADC sequence interrupt handler:

```
typedef void  (*ADC_CALLBK_T) (ErrorCode_t error_code, uint32_t num_channel );
```

`error_code`: ADC error code

`num_channel`: In interrupt mode, number of ADC channels that have been converted.

Depending on the driver mode, the ADC_CALLBAK_T is either invoked in the ADC overrun, threshold compare interrupt handler, or DMA interrupt handler.

In the overrun and threshold compare interrupt mode, the error code and the number of ADC channels are passed to the callback function. In DMA mode, the callback functions indicates the completion of the DMA transfer.

39.4.12.6 ADC_DMA_SETUP_T

This function defines the ADC DMA channel set-up function type.

```
typedef ErrorCode_t  (*ADC_DMA_SETUP_T) (ADC_HANDLE_T handle, ADC_DMA_CFG_T *dma_cfg);
```

When the driver mode is DMA, this DMA set-up callback is invoked.

To set up the DMA channel, the source address, destination address, and the DMA transfer length, the DMA request information must be retrieved from the driver structure, which is originally passed from ADC_PARAM_T structure.

39.5 Functional description

39.5.1 Example

Perform a simple analog-to-digital conversion with the ADC0 in interrupt mode (sequence A done) using a hardware trigger (ADC0_PINTRIG0). Triggering the sequence once starts the conversion. All 12 channels are sampled. Use the ADC API as follows:

Remark: For this example, burst mode must be disabled.

1. Assign the ADC0_PINTRIG0 function to a pin in the switch matrix.
2. Global defines:

```
#define rom_drivers_ptr (* (ROM **) 0x03000200)
ADCD_API_T *  pAdcApi ;  //define pointer to type API function addr table
ADC_HANDLE_T*  adc_handle_0; //handle to ADC0 API

ADC_PARAM_T        param;
ADC_CONFIG_T       adc_cfg;

#define RAMBLOCK_H  60
uint32_t  start_of_ram_block0[ RAMBLOCK_H ] ;

#define BUFFER_SIZE   100
uint32_t adc_buffer[BUFFER_SIZE];
```

3. Initialize pointer to the ADC API:

```
 pAdcApi = (ADCD_API_T *) (rom_drivers_ptr->pADCD);
```

4. Initialize ADC0:

```
size_in_bytes = pAdcApi->adc_get_mem_size() ; //size_in_bytes/4 must be
 // < RAMBLOCK_H
adc_handle_0 = pAdcApi->adc_setup(LPC_ADC0_BASE, (uint8_ *)start_of_ram_block0);
```

5. Set up the ADC0 for calibration by defining the ADC_CONFIG_T structure:

```
ADC_CONFIG_T adc_set;
adc_set.system_clock = SYSTEMCLOCK; //system clock
```

```
adc_set.adc_clock = ADC_CALIB_CLK; // ADC clock
```

6. Always calibrate the ADC before performing conversions:

```
pAdcApi->adc_calibration(adc_handle_0, &adc_set);
```

7. Set up the ADC0 for interrupt mode using the hardware trigger pin ADC0_PINTRIG0 by defining the ADC_CONFIG_T structure:

```
adc_set.adc_clock = ADC_ADC_CLK; // this is the ADC clock - in synchronous mode
      //equal to the system clock divided by the CLKDIV value in the CTRL register
adc_set.async_mode = 0; //async mode disabled
adc_set.tenbit_mode = 0; //10-bit mode disabled
adc_set.lpwr_mode = 0; //low-power mode disabled
adc_set.thr0_low = 0; //threshold disabled
adc_set.thr0_high = 0; //threshold disabled
adc_set.thr1_low = 0; //threshold disabled
adc_set.thr1_high = 0; //threshold disabled
adc_set.error_en = OVR_ENA; //overrun error interrupt disabled
adc_cfg.seqa_ctrl = CHANNELS(0xFFF) | TRIGGER(0x0); //use all 12 channels and
     // trigger # 0
adc_cfg.channel_num = 0xC; // highest channel number is 12
```

8. Initialize ADC0 conversion:

```
pAdcApi->adc_init(adc_handle_0, &adc_set);
```

9. Enable the ADC_SEQA interrupt in the NVIC.

10. Set up the ADC parameter structure ADC_PARAM_T:

```
param.driver_mode = 0x01; //interrupt mode.
param.seqa_hwtrig = 0x1;
param.adc_cfg = (ADC_CONFIG_T *)&adc_cfg;
param.buffer = (uint32_t *)adc_buffer;
param.comp_flags = 0;
param.seqa_callback_pt = adc_seqa_callback;
```

11. Define the callback function invoked when the sequence A interrupt is raised. The ADC0 handle is passed as an argument to the callback function.

```
void  adc_seqa_callback( ADC_HANDLE_T handle ) {
     ADC_DRIVER_TypeDef *driver = (ADC_DRIVER_TypeDef *)handle;
     ADC_REGS_T *adcr = ((ADC_REGS_T *)((ADC_DRIVER_TypeDef
          *)handle)->base_addr);

     if ((ErrorCode_t)driver->error_code != LPC_OK) {
          while(1);
     }
     adcr->SEQA_CTRL &= ~( ADC_SEQ_ENA );           // stop ADC SEQA now.
     adcr->INTEN &= ~SEQA_ENA;
     completiona_tag = 1;

}
```

12. Perform conversion. The result is returned in the callback function.

```
pAdcApi->adc_seqa_read(adc_handle, &param);
while (!completiona_tag); // wait until ADC conversion is complete
```

40.1 How to read this chapter

The SPI ROM driver routines are available on all parts.

40.2 Features

- Send and receive data in slave or master mode.
- Support for DMA mode.

40.3 General description

The SPI API handles SPI data transfer in master and slave modes.

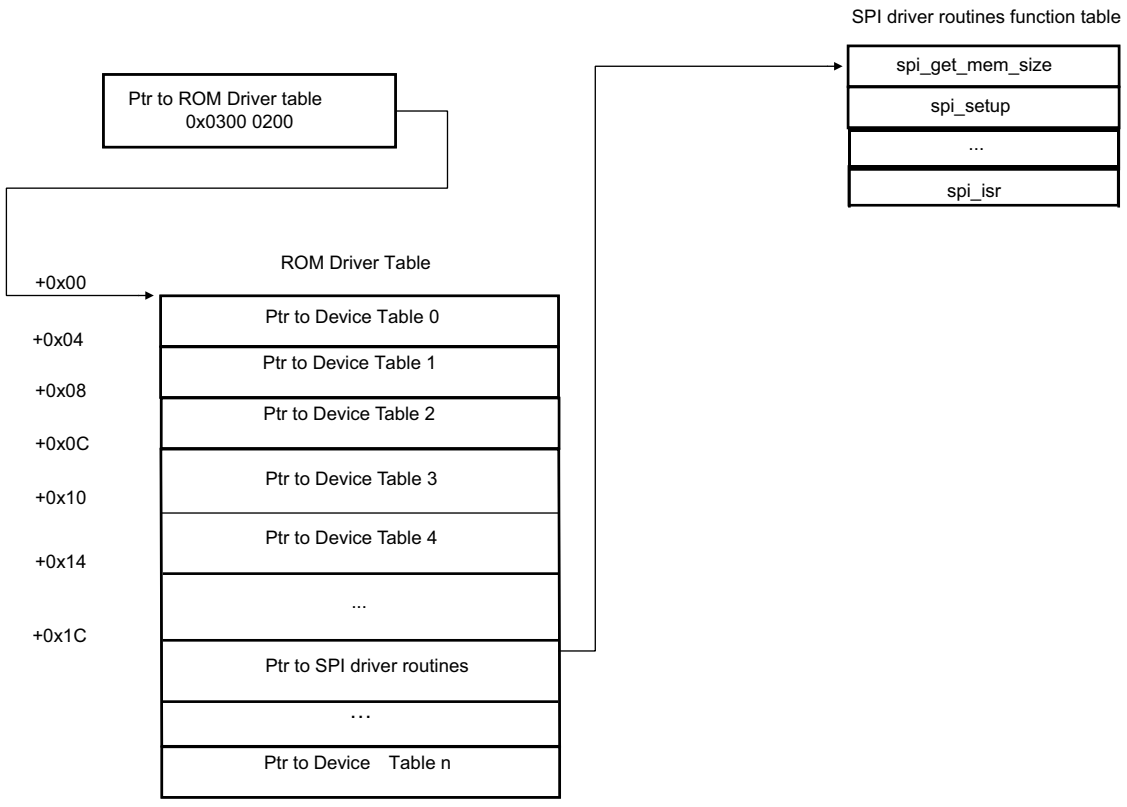


Fig 102. SPI driver routines pointer structure

40.4 API description

The SPI API contains functions to send and receive data via any of the SPI interfaces in master and slave modes.

Table 575. SPI API calls

API call	Description	Reference
`uint32_t spi_get_mem_size(void);`	Memory size for one SPI instance	Table 576
`SPI_HANDLE_T* spi_setup(uint32_t base_addr, uint8_t *ram);`	Set up SPI instance and return handle	Table 577
`uint32_t spi_init(SPI_HANDLE_T* handle, SPI_CONFIG set);`	Set up SPI operating mode	Table 578
`uint32_t spi_master_transfer(SPI_HANDLE_T* handle, SPI_PARAM_T param);`	Send or receive data in master mode	Table 579
`uint8_t spi_slave_transfer(SPI_HANDLE_T* handle, SPI_PARAM_T param);`	Send or receive data in slave mode	Table 580
`void spi_isr(SPI_HANDLE_T* handle);`	Interrupt service routine	Table 581

The following structure has to be defined to use the SPI API:

```
typedef struct {   // index of all the SPI driver functions
     uint32_t (*spi_get_mem_size)(void);
     SPI_HANDLE_T (*spi_setup)(uint32_t base_addr, uint8_t *ram);
     void (*spi_init)(SPI_HANDLE_T handle, SPI_CONFIG_T *set);
     uint32_t (*spi_master_transfer)(SPI_HANDLE_T handle, SPI_PARAM_T * param);
     uint32_t (*spi_slave_transfer)(SPI_HANDLE_T handle, SPI_PARAM_T * param);
     //--interrupt functions--//
     void (*spi_isr)(SPI_HANDLE_T handle);
} SPID_API_T  ;
```

40.4.1 SPI get memory size

Table 576. spi_get_mem_size

Routine	spi_get_mem_size
Prototype	`uint32_t spi_get_mem_size(void);`
Input parameter	None.
Return	Memory size in bytes.
Description	Get the memory size needed by one SPI instance.

40.4.2 SPI setup

Table 577. spi_setup

Routine	spi_setup
Prototype	`SPI_HANDLE_T* spi_setup(uint32_t base_addr, uint8_t *ram);`
Input parameter	base_addr: Register base address for this SPI block. ram: Pointer to the memory space for SPI instance; the memory size is obtained from the spi_get_mem_size() function.
Return	The handle to corresponding SPI instance.
Description	Set up SPI instance with provided memory and return the handle to this instance.

40.4.3 SPI init

See Section 40.4.8.2 for the SPI_CONFIG and Section 40.4.8.1 for SPI_HANDLE_T variables.

Table 578. spi_init

Routine	spi_init
Prototype	`uint32_t spi_init(SPI_HANDLE_T* handle, SPI_CONFIG set);`
Input parameter	handle: The handle to the SPI instance. set: configuration for SPI operation.
Return	None.
Description	Set up operation mode for SPI, then enable SPI.

40.4.4 SPI master data transfer

See Section 40.4.8.1 for SPI_HANDLE_T variable.

Table 579. spi_master_transfer

Routine	spi_master_transfer
Prototype	`uint32_t spi_master_transfer(SPI_HANDLE_T* handle, SPI_PARAM_T param);`
Input parameter	handle: The handle to the SPI instance. param: Refer to SPI_PARAM_T definition.
Return	Error code.
Description	Master send or receive data through SPI.

40.4.5 SPI slave data transfer

See Section 40.4.8.1 for SPI_HANDLE_T variable.

Table 580. spi_slave_transfer

Routine	spi_slave_transfer
Prototype	`uint8_t spi_slave_transfer(SPI_HANDLE_T* handle, SPI_PARAM_T param);`
Input parameter	handle: The handle to the SPI instance. param: Refer to SPI_PARAM_T definition.
Return	Error code.
Description	Slave send or receive data to SPI.

40.4.6 SPI interrupt service routine

See Section 40.4.8.1 for SPI_HANDLE_T variable.

Table 581. spi_isr

Routine	spi_isr
Prototype	`void spi_isr(SPI_HANDLE_T* handle);`
Input parameter	handle: The handle to the SPI instance.
Return	None.
Description	SPI interrupt service routine. To use this routine, the corresponding SPI interrupt must be enabled. This function is invoked by the user ISR.

40.4.7 Error codes

Table 582. Error codes

Return code	Error Code	Description
0x000E 0001	ERR_SPI_RXOVERRUN	
0x000E 0002	ERR_SPI_TXUNDERRUN	
0x000E 0003	ERR_SPI_SELNASSERT	
0x000E 0004	ERR_SPI_SELNDEASSERT	
0x000E 0005	ERR_SPI_CLKSTALL	
0x000E 0006	ERR_SPI_PARAM	
0x000E 0007	ERR_SPI_INVALID_LENGTH	

```
typedef enum
{
 ERR_SPI_BASE = 0x000E0000,
  /*0x000E0001*/ ERR_SPI_RXOVERRUN=ERR_SPI_BASE+1,
  /*0x000E0002*/ ERR_SPI_TXUNDERRUN,
  /*0x000E0003*/ ERR_SPI_SELNASSERT,
  /*0x000E0004*/ ERR_SPI_SELNDEASSERT,
  /*0x000E0005*/ ERR_SPI_CLKSTALL,
  /*0x000E0006*/ ERR_SPI_PARAM,
  /*0x000E0007*/ ERR_SPI_INVALID_LENGTH
} ErrorCode_t;
```

40.4.8 SPI ROM driver variables

40.4.8.1 SPI_HANDLE_T

The handle to the instance of the SPI driver. Each SPI has one handle, so there can be several handles for each SPI block. This handle is created by Init API and used by the transfer functions for the corresponding SPI block.

```
typedef  void     SPI_HANDLE_T   ;    // define TYPE for SPI handle pointer
```

40.4.8.2 SPI_CONFIG_T

```
Typdef  struct {
     uint32_t delay;
     uint32_t divider;
     uint32_t  config;// config register
     uint32_t error_en;//Bit0: OverrunEn, bit1: UnderrunEn,
}SPI_CONFIG_T;
```

The following variables are used in this structure:

`delay`: Configures the delay between SSEL and data transfers and between frames. The value is the content of the SPI DLY register. See Table 362.

`divider`: Clock divider value DIVVAL in the SPI DIV register. See Table 370.

`config`: Enable SPI, configure master/slave, configure signal phase and polarity. The value is the content of the SPI CFG register. See Table 361.

error_en: Enables the receive overrun and transmit underrun error interrupts.

40.4.8.3 SPI_PARAM_T

This structure defines the SPI configuration.

```
typedef struct {     // params passed to SPI driver function
     uint16_t *tx_buffer;// SPI TX buffer, needed in master and slave TX only
                         //mode
     uint16_t *rx_buffer;// SPI RX buffer, needed in master RX only, TX and RX,
                         //and slave RX mode,
     uint32_t size;// total number of SPI frames
     uint32_t fsize_sel;// data lenth of one transfer and SPI SSELx select in TXCTL
     uint32_t tx_rx_flag;
     uint32_t driver_mode;
     SPI_DMA_CFG_T *dma_cfg;// DMA configuration
     SPI_CALLBK_Tcb; // callback function
     SPI_DMA_REQ_T dma_cb; // SPI DMA channel setup callback
} SPI_PARAM_T;
```

The following variables are used in this structure:

size: number of SPI frames. A transfer can consist of several transmits of the TXCTLDAT register and of several frames.

fsize_sel: write the contents of the SPI TXCTL register to select the data length and the slave select lines. In slave mode, you need to only select the data length. See Table 369 “SPI Transmitter Control register (TXCTL, addresses 0x4004 8020 (SPI0), 0x4004 C020 (SPI1)) bit description”.

tx_rx_flag: select receive/transmit mode.

0x0 = SPI transmit only mode

0x1 = SPI receive only mode

0x2 = SPI transmit and receive mode, master mode only, transmit data transfer on MOSI and receive data transfer on MISO.

driver_mode: select the driver mode.

0x0 = Polling mode. Function is blocked until transfer has finished.

0x1 = Interrupt mode. Function exits immediately and a call back function is invoked when the transfer has finished.

0x2 = DMA mode. Data transferred by SPI is processed by DMA. The DMA_req function is called for SPI DMA channel set up. The callback function indicates when the transfer is complete.

40.4.8.4 SPI_DMA_CFG_T

This structure configures the channels used for DMA transfer between the SPI and memory. See Table 167 “DMA requests”, for the DMA channels connected to the SPI receive and transmit request lines.

To perform a DMA transfer for receive data, also enable the SPI transmit DMA channel, so that a SPI clock is generated.

Configure the DMA transfer using the DMA API. The handle to the DMA instance returned by the DMA API is used by the SPI DMA for the SPI transfer.

```
typedef struct {
     uint32_t        dma_txd_num; // SPI TX DMA channel number.
     uint32_t        dma_rxd_num;// SPI RX DMA channel number. In order to do a SPI RX
                                 // DMA, a SPI TX DMA is also needed to generated SPI
                                 // clock.
     DMA_HANDLE_T hDMA;// DMA handle
} SPI_DMA_CFG_T;
```

40.4.8.5 CALLBK_T

The callback is either invoked in the SPI interrupt handler or the DMA interrupt handler depending on the driver mode.

```
typedef void  (*CALLBK_T) (ErrorCode_t error_code, uint32_t num_transfer );
```

`error_code`: SPI error code

`num_transfer`: In interrupt mode, this parameter indicates the number of SPI frames. In DMA mode, this parameter is always zero.

In interrupt mode, error code and number of transfers will be passed to the callback.

In DMA mode, the callback indicates the completion of the DMA transfer.

In master mode, the `frame_size` parameter must be set so that the EOT (End of the transfer) bit in the SPICTL register is set to assert the SPI slave select line.

40.4.8.6 SPI_DMA_REQ_T

When the driver mode is DMA, the following DMA set-up callback is invoked:

```
typedef ErrorCode_t  (*SPI_DMA_REQ_T) (SPI_HANDLE_T handle, SPI_DMA_CFG_T *dma_cfg);
```

To set up the DMA channel, the source address, destination address, DMA transfer length, DMA request information must be retrieved from the driver structure which has been originally passed from the SPI_PARAM_T structure.

40.5 Functional description

40.5.1 Example (no DMA)

Send and receive characters in interrupt mode. Use the SPI API as follows:

1. Assign the SCK, MOSI, MISO, and SSEL0 functions to pins in the switch matrix and set up the system clock.
2. Global defines:

```
#define rom_drivers_ptr (* (ROM **) 0x03000200)
SPID_API_T *  pSpiApi ;  //define pointer to type API function addr table
SPI_HANDLE_T* spi_handle; //handle to SPI API

SPI_PARAM_T     param;
```

```
#define RAMBLOCK_H  60
uint32_t  start_of_ram_block0[ RAMBLOCK_H ] ;

#define BUFFER_SIZE   100
uint32_t spi_buffer[BUFFER_SIZE];
```

3. Define configuration structure and initialize pointer to the API:

```
SPI_CONFIG_T    spi_set;
 pSpiApi = (SPID_API_T *) (rom_drivers_ptr->pSPID);
```

4. Initialize SPI:

```
size_in_bytes = pSpiApi->spi_get_mem_size() ; //size_in_bytes/4 must be
 // < RAMBLOCK_H
spi_handle = pSpiApi->spi_setup(LPC_SPI0_BASE, (uint8_ *)start_of_ram_block0);
```

5. Set up the SPI0 in master mode by defining the SPI_CONFIG_T structure:

```
spi_set.delay =
DLY_PREDELAY(0x0)|DLY_POSTDELAY(0x0)|DLY_FRAMEDELAY(0x0)|DLY_INTERDELAY(0x0);
spi_set.divider = 0xFFFF; // divide system clock by 0xFFFF for SCK
spi_set.config = CFG_MASTER; //master mode
spi_set.error_en = STAT_RXOVERRUN | STAT_TXUNDERRUN;
```

6. Initialize SPI transfer:

```
pSpiApi->spi_init(spi_handle, &spi_set);
```

7. Enable the SPI interrupt in the NVIC.

8. Set up the SPI parameter structure SPI_PARAM_T:

```
param.driver_mode = 0x01; //interrupt mode.
param.tx_rx_flag  = 0x02; // TX AND RX
param.fsize_sel = TXDATCTL_SSELN(SLAVE0) | TXDATCTL_FSIZE(MASTER_FRAME_SIZE);
param.tx_buffer = (uint16_t *)tx_buffer;
param.rx_buffer = (uint16_t *)rx_buffer;
param.eof_flag  = 0;                    // no framedelay used
param.size = 10;  //10 transfers of TXDATCTL_FSIZE size data.
// If receive callback is invoked, transmit follows automatically. Only one
// callback is needed.
param.callback_func_pt = receive_callback;
```

9. Define the receive callback function invoked when the transmit has finished:

```
void  receive_callback(uint32_t err_code, uint32_t n ) {
error_code = (ErrorCode_t)err_code;
     if (err_code != LPC_OK)
     while(1);
     receive_tag = 1;
}
```

10. Master transmits data. Slave sends some data back.

```
receive_tag = 0;
pSpiApi->spi_master_transfer(spi_handle, &param);
while(!receivetag); //wait until receivetag is set
```

40.5.2 Example (using DMA)

The DMA sends and receives characters via the SPI in master mode. Use the SPI API and DMA API as follows:

1. Assign the SCK, MOSI, MISO, and SSEL0 functions to pins in the switch matrix.
2. Global defines:

```
SPID_API_T *  pSpiApi ;  //define pointer to type API function addr table
SPI_HANDLE_T* spi_handle; //handle to SPI API

SPI_PARAM_T     param;
SPI_CONFIG_T    spi_cfg;
SPI_DMA_CFG_T  dmc_cfg;

#define RAMBLOCK_H  60
uint32_t  start_of_ram_block0[ RAMBLOCK_H ] ;

#define BUFFER_SIZE   100
uint32_t adc_buffer[BUFFER_SIZE];
```

3. Initialize pointer to the SPI API:

```
 pSpiApi = (SPID_API_T *) (rom_drivers_ptr->pSPID);
```

4. Initialize SPI:

```
size_in_bytes = pSpiApi->spi_get_mem_size() ; //size_in_bytes/4 must be
 // < RAMBLOCK_H
spi_handle = pSpiApi->spi_setup(LPC_SPI0_BASE, (uint8_ *)start_of_ram_block0);
```

5. Set up the DMA:

```
size_in_bytes =  pDmaApi->dma_get_mem_size(); //size_in_bytes/4 must be
     // < RAMBLOCK_H

pDmaApi = (DMAD_API_T *) (rom_drivers_ptr->pDMAD);
align_location = (uint32_t *)0x02008000;
dma_handle = pDmaApi->dma_setup(LPC_DMA_BASE, (uint8_t *)align_location);
```

6. Set up the SPI0 in DMA mode by defining the SPI_CONFIG_T structure:

```
spi_set.delay =
DLY_PREDELAY(0x0)|DLY_POSTDELAY(0x0)|DLY_FRAMEDELAY(0x0)|DLY_INTERDELAY(0x0);
spi_set.divider = 0xFFFF; // divide system clock by 0xFFFF for SCK
spi_set.config = CFG_MASTER; //master mode
spi_set.error_en = STAT_RXOVERRUN | STAT_TXUNDERRUN; //disable
     // error interrupts - these are only needed in slave mode
```

7. Set-up the DMA_CONFIG_T structure for this DMA transfer (see Section 40.4.8.4):

```
dma_cfg.dma_txd_num = DMAREQ_SPI0_TX;
dma_cfg.dma_rxd_num = DMAREQ_SPI0_RX;
dma_cfg.dma_handle  = (uint32_t)dma_handle;
```

8. Initialize SPI transfer:

```
pSpiApi->spi_init(spi_handle, &spi_set);
```

9. Set up the SPI parameter structure SPI_PARAM_T for the SPI:

```
param.driver_mode = 0x02; //DMA mode.
param.tx_rx_flag  = 0x02; // TX AND RX
param.fsize_sel = TXDATCTL_SSELN(SLAVE0) | TXDATCTL_FSIZE(MASTER_FRAME_SIZE);
param.tx_buffer = (uint16_t *)tx_buffer;
param.rx_buffer = (uint16_t *)rx_buffer;
param.eof_flag  = 1; // after data transfer introduce frame delay.
param.size = 10;  //10 transfers of TXDATCTL_FSIZE size data.
// If receive callback is invoked, transmit follows automatically. Only one
// callback is needed.
param.callback_func_pt = receive_callback;

// DMA set-up

 param.dma_cfg       = &dma_cfg;
 param.dma_req_func_pt = dma_transfer_callback;
```

10. Define the DMA receive callback function invoked when the transmit has finished:

```
ErrorCode_t dma_transfer_callback( SPI_HANDLE_T handle, SPI_DMA_CFG_T *dma_cfg )
{
    DMA_CHANNEL_T chn;
    DMA_TASK_T tsk;
    ErrorCode_t error_code;

    LSPI_REGS_T *lspi = ((LSPI_REGS_T *)((SPI_DRIVER_TypeDef*)handle)->base_addr);
     SPI_DRIVER_TypeDef *driver = (SPI_DRIVER_TypeDef *)handle;

    chn.event = DMA_PER_REQ; //enable peripheral request
    chn.hd_trigger = 0;
    chn.priority = 0;
    tsk.config = DMA_SW_ON | DMA_INT_A; //enable trigger for peripheral request and
           // DMA interrupt
    tsk.data_length = driver->buffer_size - 1;
```

```
    // Enable DMA for SPI RX first. For RX only or TX_RX, only callback_rxd is
     // needed.
    if (driver->callback_rxd != NULL) {
      chn.callback_func_pt = driver->callback_rxd; //set callback function
    }

   tsk.ch_num = dma_cfg->dma_rxd_num;
    tsk.data_type = DMA_8_BIT | DMA_DST_INC_1;
// peripheral to memory
    tsk.src = (uint32_t)&lspi->RXDAT;
    tsk.dst = (uint32_t)driver->buffer_rxd + tsk.data_length;
    tsk.task_addr = NULL; // for task head, no momery is needed.
    error_code = pDmaApi->dma_init((DMA_HANDLE_T*)dma_cfg->dma_handle, &chn,
&tsk);
    if ( error_code != LPC_OK )
      return ( error_code );

    tsk.ch_num = dma_cfg->dma_txd_num;
    tsk.data_type = DMA_8_BIT | DMA_SRC_INC_1;
// memory to peripheral
    tsk.src = (uint32_t)driver->buffer_txd + tsk.data_length;
    tsk.dst = (uint32_t)&lspi->TXDAT;
    tsk.task_addr = NULL; // for task head, no momery is needed.
    error_code = pDmaApi->dma_init((DMA_HANDLE_T*)dma_cfg->dma_handle, &chn,
&tsk);
    return( error_code ); //init spi dma channel
}
```

11. Define the receive callback function. This function is called when the DMA transfer has finished and sets the EOT bit to one in the SP TXDATCTL register (see Table 369) to set the SSEL signal to HIGH.

```
void  receive_callback( uint32_t err_code, uint32_t n ) {
  LSPI_REGS_T *lspi = ((LSPI_REGS_T
     *)((SPI_DRIVER_TypeDef*)spi_handle)->base_addr);
  if (err_code != LPC_OK)
    while(1);
  // set the EOT flag in the TXCTL register
  if ( lspi->CFG & CFG_MASTER )
    lspi->TXCTL |= TXDATCTL_EOT;
    receive_tag = 1;
}
```

12. DMA transmits data. Slave sends some data back to the DMA.

```
receive_tag = 0;
pSpiApi->spi_master_transfer(spi_handle, &param);
while(!receivetag); //wait for receive tag to be set
```

41.1 How to read this chapter

The C_CAN ROM driver routines are available on all parts.

41.2 Features

The on-chip drivers are stored in boot ROM and offer CAN and CANopen initialization and communication features to user applications via a defined API. The following functions are included in the API:

- CAN set-up and initialization
- CAN send and receive messages
- CAN status
- CANopen Object Dictionary
- CANopen SDO expedited communication
- CANopen SDO segmented communication primitives
- CANopen SDO fall-back handler

41.3 General description

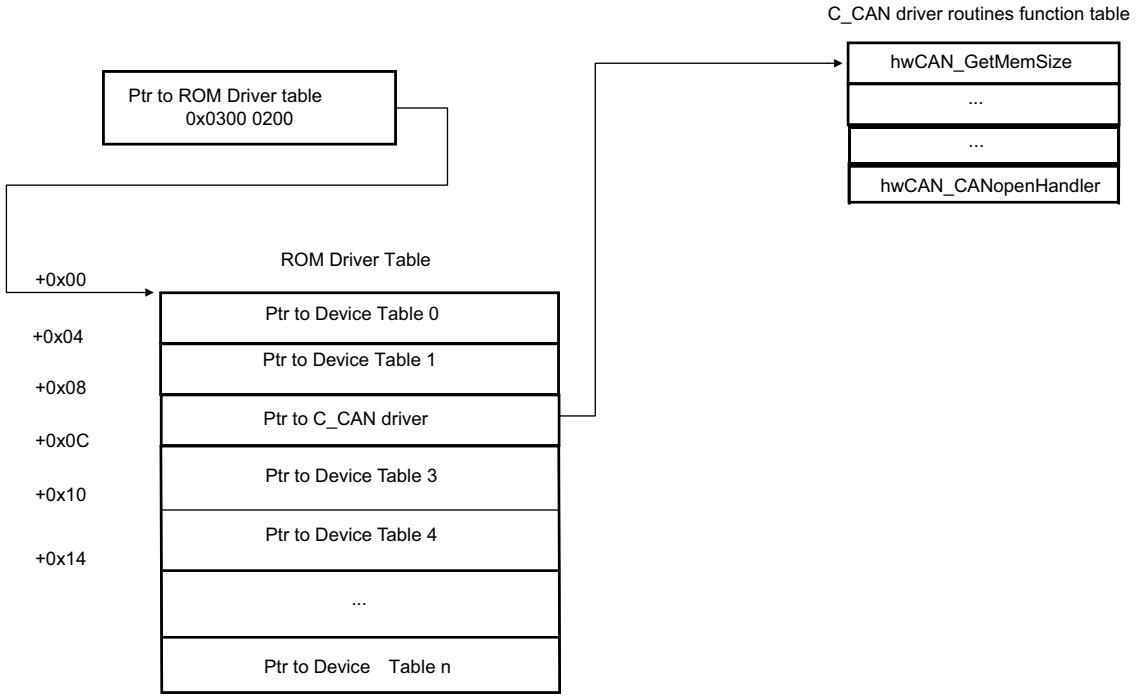


Fig 103. C_CAN driver routines pointer structure

41.4 API description

The C_CAN API contains functions to operate the C_CAN interface.

Table 583. C_CAN API calls

API call	Description	Reference
`uint32_t hwCAN_GetMemSize(CAN_API_INIT_PARAM_T* param);`	Get memory size of one instance of the C_CAN controller	Table 584
`ErrorCode_t hwCAN_Init(CAN_HANDLE_T* phCan, CAN_API_INIT_PARAM_T* param);`	Initialize C_CAN controller, set bit rate, set clock divider	Table 585
`void hwCAN_Isr(CAN_HANDLE_T hCan);`	Interrupt service routine	Table 586
`void hwCAN_ConfigRxmsgobj(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`	Configure RX message object	Table 587
`uint8_t hwCAN_MsgReceive(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`	Receive message on CAN bus	Table 588
`void hwCAN_MsgTransmit(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`	Transmit message on CAN bus	Table 589
`void hwCAN_CANopenHandler(CAN_HANDLE_T hCan);`	CANopen configuration	Table 590

The following structure has to be defined to use the C_CAN API:

```
typedefstruct _CAND_API_T {
     uint32_t (*hwCAN_GetMemSize)(CAN_API_INIT_PARAM_T* param);
     ErrorCode_t (*hwCAN_Init)(CAN_HANDLE_T* phCan, CAN_API_INIT_PARAM_T* param);
     void (*hwCAN_Isr)(CAN_HANDLE_T hCan);
     void (*hwCAN_ConfigRxmsgobj)(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);
     uint8_t (*hwCAN_MsgReceive)(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);
     void (*hwCAN_MsgTransmit)(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);
     void (*hwCAN_CANopenHandler)(CAN_HANDLE_T hCan);
}  CAND_API_T;
```

41.4.1 C_CAN get memory size

This function uses the CAN_API_INIT_PARAM structure to configure the memory location for the message objects and provide access to the callback functions. See Section 41.4.9.2.

Table 584. hwCAN_GetMemSize

Routine	hwCAN_GetMemSize
Prototype	`uint32_t hwCAN_GetMemSize(CAN_API_INIT_PARAM_T* param);`
Input parameter	param: pointer to the C_CAN init parameter object
Return	Memory size in bytes. The result is not used by the C_CAN API itself. The user code can check this value against available SRAM size.
Description	Get memory size for one C_CAN object.

41.4.2 C_CAN initialize

This function uses the CAN_API_INIT_PARAM structure to configure the memory location for the message objects, configure the C_CAN bitrate, and provide access to the callback functions and the handle to the C_CAN object - where does the handle come from????. See Section 41.4.9.2.

Table 585. hwCAN_Init

Routine	hwCAN_Init
Prototype	`ErrorCode_t hwCAN_Init(CAN_HANDLE_T* phCan, CAN_API_INIT_PARAM_T* param);`
Input parameter	phCan: The handle to the C_CAN instance. param: pointer to the C_CAN init parameter object for configuration.
Return	Error code.
Description	Set up operation mode for C_CAN, then enable C_CAN.

41.4.3 C_CAN interrupt service

Table 586. hwCAN_Isr

Routine	hwCAN_Isr
Prototype	`void hwCAN_Isr(CAN_HANDLE_T hCan);`
Input parameter	hCan: The handle to the C_CAN instance.
Return	None
Description	C_CAN interrupt service routine.

41.4.4 C_CAN configure receive message object

Table 587. hwCAN_ConfigRxmsgobj

Routine	hwCAN_ConfigRxmsgobj
Prototype	`void hwCAN_ConfigRxmsgobj(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`
Input parameter	hCan: The handle to the C_CAN instance. msg_obj: pointer to the C_CAN message object structure.
Return	None
Description	Configures the receive message object.

41.4.5 C_CAN receive message object

Table 588. hwCAN_MsgReceive

Routine	hwCAN_MsgReceive
Prototype	`uint8_t hwCAN_MsgReceive(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`
Input parameter	hCan: The handle to the C_CAN instance. msg_obj: pointer to the C_CAN message object structure.
Return	0 = None. 1 = Received valid message object.
Description	Contains the received message object with data

41.4.6 C_CAN transmit message object

Table 589. hwCAN_MsgTransmit

Routine	hwCAN_MsgTransmit
Prototype	`void hwCAN_MsgTransmit(CAN_HANDLE_T hCan, CAN_MSG_OBJ * msg_obj);`
Input parameter	hCan: The handle to the C_CAN instance. msg_obj: pointer to the C_CAN message object structure.
Return	None
Description	Transmits the message object with data.

41.4.7 C_CAN CANopen handler

Table 590. hwCAN_CANopenHandler

Routine	hwCAN_CANopenHandler
Prototype	`void hwCAN_CANopenHandler(CAN_HANDLE_T hCan);`
Input parameter	hCan: The handle to the C_CAN instance.
Return	None
Description	-

41.4.8 Error codes

Table 591. Error codes

Return code	Error Code	Description
0x0009 0001	ERR_CAN_BAD_MEM_BUF	-
0x0009 0002	ERR_CAN_INIT_FAIL	-
0x0009 0003	ERR_CANOPEN_INIT_FAIL	-

```
typedef enum
 ERR_CAN_BASE = 0x00090000,
  /*0x00090001*/ ERR_CAN_BAD_MEM_BUF = ERR_CAN_BASE+1,
  /*0x00090002*/ ERR_CAN_INIT_FAIL,
  /*0x00090003*/ ERR_CANOPEN_INIT_FAIL
} ErrorCode_t;
```

41.4.9 C_CAN ROM driver variables

41.4.9.1 CAN_HANDLE_T

The handle to the instance of the C_CAN driver. This handle is created by Init API and used by the transfer functions for the corresponding C_CAN block.

```
typedef  void CAN_HANDLE_T   ;    // define TYPE for CAN handle pointer_
```

41.4.9.2 _CAN_API_INIT_PARAM_T

This structure defines the address for the can memory and provides access to the configuration structure and the CAN and CANopen callback functions.

```
typedef struct _CAN_API_INIT_PARAM_T {
     uint32_t mem_base;      // Address of user-space memory area to use
     uint32_t can_reg_base;  // Address of start of CAN controller register area
     CAN_CFG  *can_cfg;
     CAN_CALLBACKS  *callbacks;
     CAN_CANOPENCFG *canopen_cfg;
     CANOPEN_CALLBACKS  *co_callbacks;
} CAN_API_INIT_PARAM_T;
```

41.4.9.3 _CAN_CFG

This structure configures the C_CAN clock divider and the bit rate.

```
typedef struct _CAN_CFG {
     uint32_t  clkdiv;
     uint32_t  btr;
     uint32_t  isr_ena;
} CAN_CFG;
```

This structure uses the following variables:

`clkdiv`: This is the clock divider for the C_CAN clock. See Table 423.

`btr`: The contents of the BT register determining the C_CAN bitrate. See Table 397.

`isr_ena`: 1 = IE and EIE interrupts are enabled. 0 = both interrupts disabled. See Table 394.

41.4.9.4 _CAN_MSG_OBJ

```
typedef struct _CAN_MSG_OBJ {
     uint32_t  mode_id;
     uint32_t  mask;
     uint8_t   data[8];
```

```
    uint8_t   dlc;
    uint8_t   msgobj;
}  CAN_MSG_OBJ;
```

The message object structure uses the following variables:

`mode_id`: C_CAN operation mode

```
#define CAN_MSGOBJ_STD  0x00000000UL   // CAN 2.0a 11-bit ID
#define CAN_MSGOBJ_EXT  0x20000000UL   // CAN 2.0b 29-bit ID
#define CAN_MSGOBJ_DAT  0x00000000UL   // data frame
#define CAN_MSGOBJ_RTR  0x40000000UL   // rtr frame
```

`mask`: Sets the mask for received message object numbers. See Table 406.

`data[8]`: Data bytes. See Table 411.

`dlc`: Data length code. See Table 410.

`msgobj`: Message object number.

41.4.9.5 CANopen configuration

The CAN API supports an Object Dictionary interface and the SDO protocol. In order to activate it, the CANopen configuration function has to be called with a pointer to a structure with the CANopen Node ID (1...127), the message object numbers to use for receive and transmit SDOs, a flag to decide whether the CANopen SDO handling should happen in the interrupt serving function automatically or via the dedicated API function, and two pointers to Object Dictionary configuration tables and their sizes.

The _CAN_ODCONSTENTRY table contains all read-only, constant entries of four bytes or less:

```
typedef struct _CAN_ODCONSTENTRY {
  uint16_t index;
  uint8_t  subindex;
  uint8_t  len;
  uint32_t val;
}  CAN_ODCONSTENTRY;
```

The CAN_ODENTRY table contains all variable and writable entries as well as SDO segmented entries:

```
typedef struct _CAN_ODENTRY {
  uint16_t index;
  uint8_t subindex;
  uint8_t entrytype_len;
  uint8_t *val;
} CAN_ODENTRY;
```

The CAN_ODENTRY structure contains the following variable:

`entrytype_len`: Upper-nibble values for OD entry length.

```
#define OD_NONE 0x00 // Object Dictionary entry doesn't exist
#define OD_EXP_RO 0x10 // Object Dictionary entry expedited, read-only
#define OD_EXP_WO 0x20 // Object Dictionary entry expedited, write-only
```

```
#define OD_EXP_RW 0x30 // Object Dictionary entry expedited, read-write
#define OD_SEG_RO 0x40 // Object Dictionary entry segmented, read-only
#define OD_SEG_WO 0x50 // Object Dictionary entry segmented, write-only
#define OD_SEG_RW 0x60 // Object Dictionary entry segmented, read-write
```

The CANopen configuration function sets up the CANopen structure:

```
typedef struct _CAN_CANOPENCFG {
  uint8_t node_id;
  uint8_t msgobj_rx;
  uint8_t msgobj_tx;
  uint8_t isr_handled;
  uint32_t od_const_num;
  CAN_ODCONSTENTRY *od_const_table;
  uint32_t od_num;
  CAN_ODENTRY *od_table;
} CAN_CANOPENCFG;
```

The CAN_OPENCFG structure contains the following variables:

`node_id`: The CANopen Node ID (1...127).

`msgobj_rx`, `msgobj_tx`: Message object numbers for receive and transmit SDOs.

`isr_handled`: 1 = CANopen SDO handling happens in the interrupt serving function automatically. 0 = CANopen SDO handling happens via the dedicated API function.

`od_const_num`: Size of the OD_CONSTENTRY table.

`CAN_ODCONSTDENTRY`: Pointer to the _CAN_ODCONSTENTRY table.

`od_num`: Size of the OD_ENTRY table.

`CAN_ODENTRY`: Pointer to the _CAN_ODENTRY table.

41.4.9.6 CANopen example OD tables and CANopen configuration structure

```
// List of fixed, read-only Object Dictionary (OD) entries
// Expedited SDO only, length=1/2/4 bytes
const CAN_ODCONSTENTRY myConstOD [] = {
// index subindex length value
  { 0x1000, 0x00, 4, 0x54534554UL }, // "TEST"
  { 0x1018, 0x00, 1, 0x00000003UL },
  { 0x1018, 0x01, 4, 0x00000003UL },
  { 0x2000, 0x00, 1, (uint32_t)'M' },
};

// List of variable OD entries
// Expedited SDO with length=1/2/4 bytes
// Segmented SDO application-handled with length and value_pointer don't care
const CAN_ODENTRY myOD [] = {
// index subindex access_type|length value_pointer
  { 0x1001, 0x00, OD_EXP_RO | 1, (uint8_t *)&error_register },
  { 0x1018, 0x02, OD_EXP_RO | 4, (uint8_t *)&device_id },
  { 0x1018, 0x03, OD_EXP_RO | 4, (uint8_t *)&fw_ver },
  { 0x2001, 0x00, OD_EXP_RW | 2, (uint8_t *)&param },
  { 0x2200, 0x00, OD_SEG_RW, (uint8_t *)NULL },
```

```
};

// CANopen configuration structure
const CAN_CANOPENCFG myCANopen = {
  20, // node_id
  5, // msgobj_rx
  6, // msgobj_tx
  TRUE, // isr_handled
  sizeof(myConstOD)/sizeof(myConstOD[0]), // od_const_num
  (CAN_ODCONSTENTRY *)myConstOD, // od_const_table
  sizeof(myOD)/sizeof(myOD[0]), // od_num
  (CAN_ODENTRY *)myOD, // od_table
};
```

Example call:

```
// Initialize CANopen handler
(*rom)->pCAND->config_canopen((CAN_CANOPENCFG *)&myCANopen[0]);
```

41.4.10 CAN callback functions

```
typedef struct _CAN_CALLBACKS {
     void (*CAN_rx)(uint8_t msg_obj);
     void (*CAN_tx)(uint8_t msg_obj);
     void (*CAN_error)(uint32_t error_info);
}  CAN_CALLBACKS;
```

41.4.10.1 CAN message received callback

The CAN message received callback function is called on the interrupt level by the CAN interrupt handler.

Example call:

```
void CAN_rx(uint8_t msg_obj_num){

  /* Determine which CAN message has been received */
  msg_obj.msgobj = msg_obj_num;

  /* Now load up the msg_obj structure with the CAN message */
  pCANDApi->hwCAN_MsgReceive(pCanHandle, &msg_obj);

  if (msg_obj_num == 1)
  {
    /* Simply transmit CAN frame (echo) with with ID +0x100 via buffer 2 */
    msg_obj.msgobj = 2;
    msg_obj.mode_id += 0x100;
    pCANDApi->hwCAN_MsgTransmit(pCanHandle, &msg_obj);
  }
  return;
}
```

Remark: The callback is not called if the user CANopen handler is activated for the message object that is used for SDO receive.

41.4.10.2 CAN message transmit callback

Called on the interrupt level by the CAN interrupt handler after a message has been

successfully transmitted on the bus.

Example call:

```
// CAN transmit handler
void CAN_tx(uint8_t msgobj_num)
{
  // Reset flag used by application to wait for transmission finished
  if (wait_for_tx_finished == msgobj_num)
    wait_for_tx_finished = 0;
  return;
}
```

Remark: The callback is not called after the user CANopen handler has used a message object to transmit an SDO response.

41.4.10.3 CAN error callback

The CAN error callback function is called on the interrupt level by the CAN interrupt handler.

```
// error status bits
#define CAN_ERROR_NONE 0x00000000UL
#define CAN_ERROR_PASS 0x00000001UL
#define CAN_ERROR_WARN 0x00000002UL
#define CAN_ERROR_BOFF 0x00000004UL
#define CAN_ERROR_STUF 0x00000008UL
#define CAN_ERROR_FORM 0x00000010UL
#define CAN_ERROR_ACK 0x00000020UL
#define CAN_ERROR_BIT1 0x00000040UL
#define CAN_ERROR_BIT0 0x00000080UL
#define CAN_ERROR_CRC 0x00000100UL
```

Example call:

```
// CAN error handler
void CAN_error(uint32_t error_info)
  // If we went into bus off state, tell the application to
  // re-initialize the CAN controller
  if (error_info & CAN_ERROR_BOFF)
    reset_can = TRUE;
  return;
}
```

41.4.11 CANopen callback functions

```
typedefstruct _CANOPEN_CALLBACKS {
     uint32_t (*CANOPEN_sdo_read)(uint16_t index, uint8_t subindex);
```

```
    uint32_t (*CANOPEN_sdo_write)(uint16_t index, uint8_t subindex, uint8_t
        *dat_ptr);
    uint32_t (*CANOPEN_sdo_seg_read)(uint16_t index, uint8_t subindex, uint8_t
    openclose, uint8_t *length, uint8_t *data, uint8_t *last);
    uint32_t (*CANOPEN_sdo_seg_write)(uint16_t index, uint8_t subindex, uint8_t
        openclose, uint8_t length, uint8_t *data, uint8_t *fast_resp);
    uint8_t (*CANOPEN_sdo_req)(uint8_t length_req, uint8_t *req_ptr, uint8_t
        *length_resp, uint8_t *resp_ptr);
}  CANOPEN_CALLBACKS;
```

41.4.11.1 CANopen SDO expedited read callback

The CANopen SDO expedited read callback function is called by the CANopen handler. The callback function is called before the SDO response is generated, allowing to modify or update the data.

Example call:

```
// CANopen callback for expedited read accesses
uint32_t CANOPEN_sdo_exp_read(uint16_t index, uint8_t subindex)
{
  // Every read of [2001h,0] increases param by one
  if ((index == 0x2001) && (subindex==0))
    param++;

  return 0;
}
```

Remark: If the flag isr_handled was set to TRUE when initializing CANopen, this callback function will be called by the CAN API interrupt handler and therefore will execute on the interrupt level.

41.4.11.2 CANopen SDO expedited write callback

The CANopen SDO expedited write callback function is called by the CANopen handler. The callback passes on the new data and is called before the new data has been written, allowing to reject or condition the data.

Example call:

```
// CANopen callback for expedited write accesses
uint32_t CANOPEN_sdo_exp_write(uint16_t index, uint8_t subindex, uint8_t
*dat_ptr)
{
  // Writing 0xAA55 to entry [2001h,0] unlocks writing the config table
  if ((index == 0x2001) && (subindex == 0))
  if (*(uint16_t *)dat_ptr == 0xAA55)
    {
      write_config_ena = TRUE;
     return(TRUE);
    }
    else
      return(FALSE); // Reject any other value
```

```
}
```

Remark: If the flag isr_handled was set TRUE when initializing CANopen, this callback function will be called by the CAN API interrupt handler and therefore will execute on the interrupt level.

41.4.11.3 CANopen SDO segmented read callback

The CANopen SDO segmented read callback function is called by the CANopen handler. The callback function allows the following actions:

- inform about the opening of the read channel.
- provide data segments of up to seven bytes to the reading host.
- close the channel when all data has been read.
- abort the transmission at any time.

```
// Values for CANOPEN_sdo_seg_read/write() callback 'openclose' parameter
#define CAN_SDOSEG_SEGMENT 0 // segment read/write
#define CAN_SDOSEG_OPEN 1 // channel is opened
#define CAN_SDOSEG_CLOSE 2 // channel is closed
```

Example call (reading a buffer):

```
uint8_t read_buffer[0x123];

// CANopen callback for segmented read accesses
uint32_t CANOPEN_sdo_seg_read(
  uint16_t index, uint8_t subindex, uint8_t openclose,
  uint8_t *length, uint8_t *data, uint8_t *last)
{
  static uint16_t read_ofs;
  uint16_t i;

  if ((index == 0x2200) && (subindex==0))
  {
    if (openclose == CAN_SDOSEG_OPEN)
    {
      // Initialize the read buffer with "something"
      for (i=0; i<sizeof(read_buffer); i++)
      {
        read_buffer[i] = (i+5) + (i<<2);
      }
      read_ofs = 0;
    }
    else if (openclose == CAN_SDOSEG_SEGMENT)
    {
      i = 7;
      while (i && (read_ofs < sizeof(read_buffer)))
      {
        *data++ = read_buffer[read_ofs++];
        i--;
      }
```

```
      *length = 7-i;
      if (read_ofs == sizeof(read_buffer)) // The whole buffer read:
                                           // this is last segment
      {
        *last = TRUE;
      }
    }
    return 0;
  }
  else
  {
    return SDO_ABORT_NOT_EXISTS;
  }
}
```

Remark: If the flag isr_handled was set TRUE when initializing CANopen, this callback function will be called by the CAN API interrupt handler and therefore will execute on the interrupt level.

41.4.11.4 CANopen SDO segmented write callback

The CANopen SDO segmented write callback function is called by the CANopen handler. The callback function allows the following actions:

- inform about the opening and closing of the write channel.
- pass on data segments of up to seven bytes from the writing host.
- abort the transmission at any time, for example when there is a buffer overflow.

Responses can be selected to be 8-byte (CANopen standard compliant) or 1-byte (faster but not supported by all SDO clients).

```
// Values for CANOPEN_sdo_seg_read/write() callback 'openclose' parameter
#define CAN_SDOSEG_SEGMENT 0 // segment read/write
#define CAN_SDOSEG_OPEN 1 // channel is opened
#define CAN_SDOSEG_CLOSE 2 // channel is closed
```

Example call (writing a buffer):

```
uint8_t write_buffer[0x321];

// CANopen callback for segmented write accesses
uint32_t CANOPEN_sdo_seg_write(
  uint16_t index, uint8_t subindex, uint8_t openclose,
  uint8_t length, uint8_t *data, uint8_t *fast_resp)
{
  static uint16_t write_ofs;
  uint16_t i;

  if ((index == 0x2200) && (subindex==0))
  {
    if (openclose == CAN_SDOSEG_OPEN)
    {
      // Initialize the write buffer
```

```
      for (i=0; i<sizeof(write_buffer); i++)
      {
        write_buffer[i] = 0;
      }
      write_ofs = 0;
    }
    else if (openclose == CAN_SDOSEG_SEGMENT)
    {
      *fast_resp = TRUE;  // Use fast 1-byte segment write response
      i = length;
      while (i && (write_ofs < sizeof(write_buffer)))
      {
        write_buffer[write_ofs++] = *data++;
        i--;
      }
      if (i && (write_ofs >= sizeof(write_buffer))) // Too much data to write
      {
        return SDO_ABORT_TRANSFER;  // Data could not be written
      }
    }
    else if (openclose == CAN_SDOSEG_CLOSE)
    {
      // Write has successfully finished: mark the buffer valid etc.
    }
    return 0;
  }
  else
  {
    return SDO_ABORT_NOT_EXISTS;
  }
}
```

Remark: If the flag isr_handled was set TRUE when initializing CANopen, this callback function will be called by the CAN API interrupt handler and therefore will execute on the interrupt level.

41.4.11.5 CANopen fall-back SDO handler callback

The CANopen fall-back SDO handler callback function is called by the CANopen handler. This function is called whenever an SDO request could not be processed or would end in an SDO abort response. It is called with the full data buffer of the request and allows to generate any type of SDO response. This can be used to implement custom SDO handlers, for example to implement the SDO block transfer method.

```
// Return values for CANOPEN_sdo_req() callback
#define CAN_SDOREQ_NOTHANDLED 0 // process regularly, no impact
#define CAN_SDOREQ_HANDLED_SEND 1 // processed in callback, auto-send
                                  // returned msg
#define CAN_SDOREQ_HANDLED_NOSEND 2 // processed in callback, don't send
                                  // response
```

Example call (not implementing custom processing):

```
// CANopen callback for custom SDO request handler
uint8_t CANOPEN_sdo_req (
  uint8_t length, uint8_t *req_ptr, uint8_t *length_resp, uint8_t *resp_ptr)
{
  return CAN_SDOREQ_NOTHANDLED;
}
```

Remark: If the flag isr_handled was set TRUE when initializing CANopen, this callback function will be called by the CAN API interrupt handler and therefore will execute on the interrupt level.

41.5 Functional description

41.5.1 Example

To send and receive a simple one-time CAN message, use the API as follows:

1. Reserve memory for C_CAN API:

```
uint32_t gCANapiMem[MAX_CAN_PARAM_SIZE];
```

2. Set up CAN_API_INIT_PARAM_T:

```
CAN_API_INIT_PARAM_T myCANConfig = {
     (uint32_t)&gCANapiMem[0],
     LPC_CAN_BASE,
     &gCANConfig,
     (CAN_CALLBACKS *)&callbacks, //need the callback for receive message
     (CAN_CANOPENCFG *)&myCANopen, // not used, but allocate memory for it
     (CANOPEN_CALLBACKS *)&co_callbacks, //not used, but allocate memory for it
};
```

3. Initialize the pointer to the C_CAN API:

```
CAN_HANDLE_T pCanHandle; //CAN handle
CAND_API_T*  pCANDApi;  //define pointer to type API function addr table
CAN_MSG_OBJ msg_obj; //define message object

#define rom_drivers_ptr (* (ROM **) 0x03000200)
pCANDApi = (CAND_API_T*) (rom_drivers_ptr->pCAND);
```

4. Initialize the C_CAN controller (system clock = 72 MHz, CLKDIV = 5 and the C_CAN clock is 72 MHZ/(CLKDIV + 1) = 12 MHz, bit clock is 500 kHz):

```
CAN_CFG gCANConfig; //CAN initialization values
gCANConfig.clkdiv = 5;
gCANConfig.btr = BITRATE500K12MHZ;
gCANConfig.isr_ena = TRUE;

if ( pCANDApi->hwCAN_Init(&pCanHandle, (CAN_API_INIT_PARAM_T *)&myCANConfig)
 != LPC_OK )
{
    while ( 1 );
  }
```

5. Enable the C_CAN interrupt in the NVIC.

6. Set up the message object:

```
msg_obj.msgobj  = 0;
msg_obj.mode_id = 0x700+0x20;
msg_obj.mask    = 0x0;
msg_obj.dlc     = 4;
msg_obj.data[0] = 'T';//0x54
msg_obj.data[1] = 'E';//0x45
msg_obj.data[2] = 'S';//0x53
msg_obj.data[3] = 'T';//0x54
```

7. Define the callback function used on a CAN message receive interrupt:

```
void CAN_rx(uint8_t msg_obj_num){
 // Determine which CAN message has been received
  msg_obj.msgobj = msg_obj_num;
  // Now load up the msg_obj structure with the CAN message
  pCANDApi->hwCAN_MsgReceive(pCanHandle, &msg_obj);
  if (msg_obj_num == 1)
  {
    // Simply transmit CAN frame (echo) with with ID +0x100 via buffer 2
    msg_obj.msgobj = 2;
    msg_obj.mode_id += 0x100;
    pCANDApi->hwCAN_MsgTransmit(pCanHandle, &msg_obj);
  }
  return;
}
```

8. Send the CAN message:

```
pCANDApi->hwCAN_MsgTransmit(pCanHandle, &msg_obj);
```

9. Configure message object 1 to receive all 11-bit messages 0x400-0x4FF:

```
msg_obj.msgobj = 1;
msg_obj.mode_id = 0x400;
msg_obj.mask = 0x700;
```

10. Receive CAN message:

```
pCANDApi->hwCAN_ConfigRxmsgobj(pCanHandle, &msg_obj);
```

42.1 How to read this chapter

The USB ROM driver routines are available on all parts.

42.2 Features

- ROM-base USB drivers
- Communication Device Class (CDC) device class
- Human Interface Device (HID) device class
- Mass storage device class

42.3 General description

The boot ROM contains a USB driver to simplify the USB application development. The USB driver implements the Communication Device Class (CDC), the Human Interface Device (HID), and the Mass Storage Device (MSC) device class. The USB on-chip drivers support composite device.

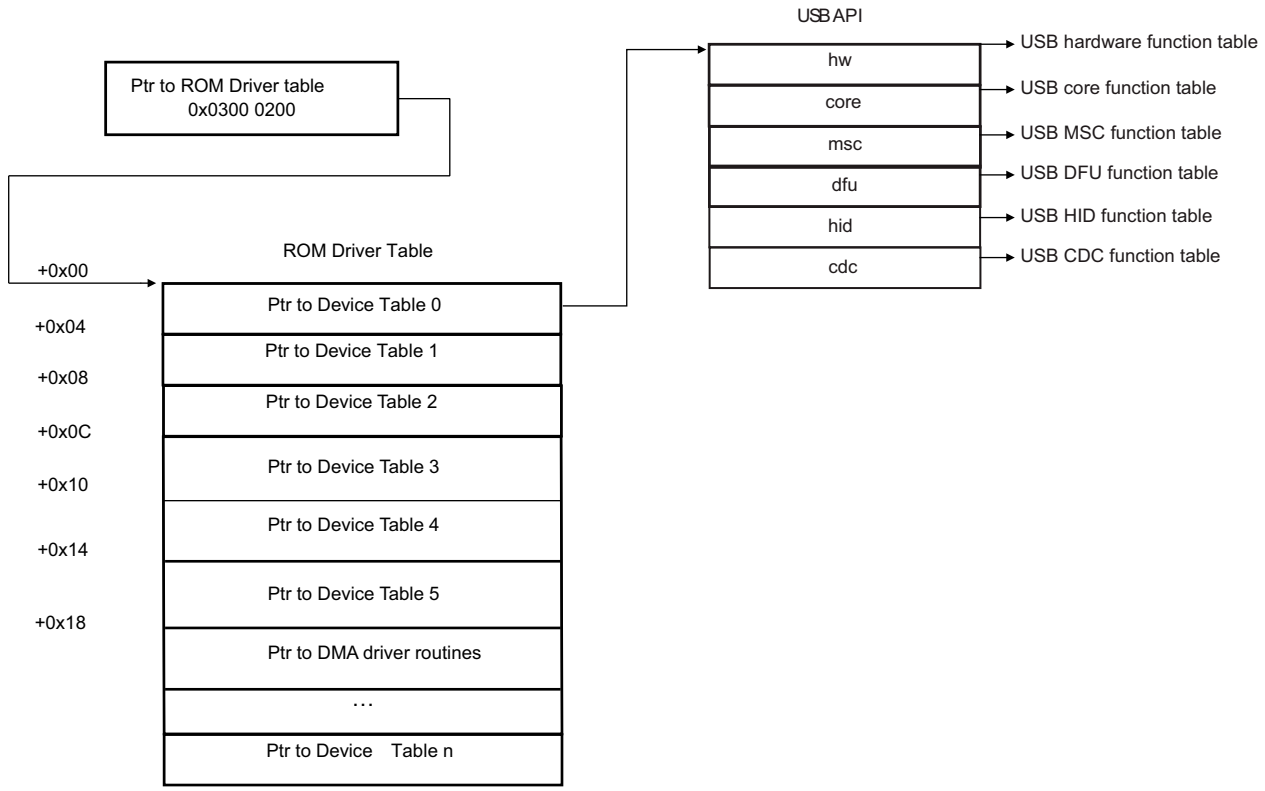


Fig 104. USB driver routines pointer structure

42.3.1 USB driver functions

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

- Communication Device Class (CDC) function driver. This module contains an internal implementation of the USB CDC Class. User applications can use this class driver instead of implementing the CDC-ACM class manually via the low-level USBD_HW and USBD_Core APIs. This module is designed to simplify the user code by exposing only the required interface needed to interface with Devices using the USB CDC-ACM Class.
  - Communication Device Class function driver initialization parameter data structure (Table 619 “USBD_CDC_INIT_PARAM class structure”).
  - CDC class API functions structure. This module exposes functions which interact directly with USB device controller hardware (Table 618 “USBD_CDC_API class structure”).
- USB core layer
  - struct (Table 615 “_WB_T class structure”)
  - union (Table 592 “__WORD_BYTE class structure”)
  - struct (Table 593 “_BM_T class structure”)
  - struct (Table 606 “_REQUEST_TYPE class structure”)

- struct (Table 613 “_USB_SETUP_PACKET class structure”)
- struct (Table 609 “_USB_DEVICE_QUALIFIER_DESCRIPTOR class structure”)
- struct USB device descriptor
- struct (Table 609 “_USB_DEVICE_QUALIFIER_DESCRIPTOR class structure”)
- struct USB configuration descriptor
- struct (Table 611 “_USB_INTERFACE_DESCRIPTOR class structure”)
- struct USB endpoint descriptor
- struct (Table 614 “_USB_STRING_DESCRIPTOR class structure”)
- struct (Table 607 “_USB_COMMON_DESCRIPTOR class structure”)
- struct (Table 612 “_USB_OTHER_SPEED_CONFIGURATION class structure”)
- USB descriptors data structure (Table 608 “_USB_CORE_DESCS_T class structure”)
- USB device stack initialization parameter data structure (Table 617 “USBD_API_INIT_PARAM class structure”).
- USB device stack core API functions structure (Table 620 “USBD_CORE_API class structure”).

- Device Firmware Upgrade (DFU) class function driver
  - DFU descriptors data structure (Table 622 “USBD_DFU_INIT_PARAM class structure”).
  - DFU class API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 621 “USBD_DFU_API class structure”).
- HID class function driver
  - struct (Table 601 “_HID_DESCRIPTOR class structure”).
  - struct (Table 603 “_HID_REPORT_T class structure”).
  - USB descriptors data structure (Table 624 “USBD_HID_INIT_PARAM class structure”).
  - HID class API functions structure. This structure contains pointers to all the functions exposed by the HID function driver module (Table 625 “USBD_HW_API class structure”).
- USB device controller driver
  - Hardware API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 625 “USBD_HW_API class structure”).
- Mass Storage Class (MSC) function driver
  - Mass Storage Class function driver initialization parameter data structure (Table 627).
  - MSC class API functions structure. This module exposes functions which interact directly with the USB device controller hardware (Table 626).

42.3.2 Calling the USB device driver

A fixed location in ROM contains a pointer to the ROM driver table. The ROM driver table contains a pointer to the USB driver table. Pointers to the various USB driver functions are stored in this table. USB driver functions can be called by using a C structure. Figure 104 illustrates the pointer mechanism used to access the on-chip USB driver.

```
typedef struct USBD_API

{

const USBD_HW_API_T* hw;

const USBD_CORE_API_T* core;

const USBD_MSC_API_T* msc;

const USBD_DFU_API_T* dfu;

const USBD_HID_API_T* hid;

const USBD_CDC_API_T* cdc;

const uint32_t* reserved6;

const uint32_t version;

} USBD_API_T;
```

42.4 USB API

42.4.1 __WORD_BYTE

Table 592. __WORD_BYTE class structure

Member	Description
W	uint16_t __WORD_BYTE::W data member to do 16 bit access
WB	WB_TWB_T __WORD_BYTE::WB data member to do 8 bit access

42.4.2 _BM_T

Table 593. _BM_T class structure

Member	Description
Recipient	uint8_t _BM_T::Recipient Recipient type.
Type	uint8_t _BM_T::Type Request type.
Dir	uint8_t _BM_T::Dir Direction type.

42.4.3 _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR

Table 594. _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bDescriptorSubtype
bmCapabilities	uint8_t _CDC_ABSTRACT_CONTROL_MANAGEMENT_DESCRIPTOR::bmCapabilities

42.4.4 _CDC_CALL_MANAGEMENT_DESCRIPTOR

Table 595. _CDC_CALL_MANAGEMENT_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDescriptorSubtype
bmCapabilities	uint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bmCapabilities
bDataInterface	uint8_t _CDC_CALL_MANAGEMENT_DESCRIPTOR::bDataInterface

42.4.5 _CDC_HEADER_DESCRIPTOR

Table 596. _CDC_HEADER_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_t _CDC_HEADER_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_t _CDC_HEADER_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_t _CDC_HEADER_DESCRIPTOR::bDescriptorSubtype
bcdCDC	uint16_t _CDC_HEADER_DESCRIPTOR::bcdCDC

42.4.6 _CDC_LINE_CODING

Table 597. _CDC_LINE_CODING class structure

Member	Description
dwDTERate	uint32_t _CDC_LINE_CODING::dwDTERate
bCharFormat	uint8_t _CDC_LINE_CODING::bCharFormat
bParityType	uint8_t _CDC_LINE_CODING::bParityType
bDataBits	uint8_t _CDC_LINE_CODING::bDataBits

42.4.7 _CDC_UNION_1SLAVE_DESCRIPTOR

Table 598. _CDC_UNION_1SLAVE_DESCRIPTOR class structure

Member	Description
sUnion	CDC_UNION_DESCRIPTORCDC_UNION_DESCRIPTOR _CDC_UNION_1SLAVE_DESCRIPTOR::sUnion
bSlaveInterfaces	uint8_t _CDC_UNION_1SLAVE_DESCRIPTOR::bSlaveInterfaces[1][1]

42.4.8 _CDC_UNION_DESCRIPTOR

Table 599. _CDC_UNION_DESCRIPTOR class structure

Member	Description
bFunctionLength	uint8_t _CDC_UNION_DESCRIPTOR::bFunctionLength
bDescriptorType	uint8_t _CDC_UNION_DESCRIPTOR::bDescriptorType
bDescriptorSubtype	uint8_t _CDC_UNION_DESCRIPTOR::bDescriptorSubtype
bMasterInterface	uint8_t _CDC_UNION_DESCRIPTOR::bMasterInterface

42.4.9 _DFU_STATUS

Table 600. _DFU_STATUS class structure

Member	Description
bStatus	uint8_t _DFU_STATUS::bStatus
bwPollTimeout	uint8_t _DFU_STATUS::bwPollTimeout[3][3]
bState	uint8_t _DFU_STATUS::bState
iString	uint8_t _DFU_STATUS::iString

42.4.10 _HID_DESCRIPTOR

HID class-specific HID Descriptor.

Table 601. _HID_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _HID_DESCRIPTOR::bLength Size of the descriptor, in bytes.
bDescriptorType	uint8_t _HID_DESCRIPTOR::bDescriptorType Type of HID descriptor.
bcdHID	uint16_t _HID_DESCRIPTOR::bcdHID BCD encoded version that the HID descriptor and device complies to.
bCountryCode	uint8_t _HID_DESCRIPTOR::bCountryCode Country code of the localized device, or zero if universal.
bNumDescriptors	uint8_t _HID_DESCRIPTOR::bNumDescriptors Total number of HID report descriptors for the interface.
DescriptorList	PRE_PACK struct POST_PACK _HID_DESCRIPTOR::_HID_DESCRIPTOR_LISTPRE_PACK struct POST_PACK _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST _HID_DESCRIPTOR::DescriptorList[1][1] Array of one or more descriptors

42.4.11 _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST

Table 602. _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST class structure

Member	Description
bDescriptorType	uint8_t _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST::bDescriptorType Type of HID report.
wDescriptorLength	uint16_t _HID_DESCRIPTOR::_HID_DESCRIPTOR_LIST::wDescriptorLength Length of the associated HID report descriptor, in bytes.

42.4.12 _HID_REPORT_T

HID report descriptor data structure.

Table 603. _HID_REPORT_T class structure

Member	Description
len	uint16_t _HID_REPORT_T::len Size of the report descriptor in bytes.
idle_time	uint8_t _HID_REPORT_T::idle_time This value is used by stack to respond to Set_Idle & GET_Idle requests for the specified report ID. The value of this field specified the rate at which duplicate reports are generated for the specified Report ID. For example, a device with two input reports could specify an idle rate of 20 milliseconds for report ID 1 and 500 milliseconds for report ID 2.
__pad	uint8_t _HID_REPORT_T::__pad Padding space.
desc	uint8_t * _HID_REPORT_T::desc Report descriptor.

42.4.13 _MSC_CBW

Table 604. _MSC_CBW class structure

Member	Description
dSignature	uint32_t _MSC_CBW::dSignature
dTag	uint32_t _MSC_CBW::dTag
dDataLength	uint32_t _MSC_CBW::dDataLength
bmFlags	uint8_t _MSC_CBW::bmFlags
bLUN	uint8_t _MSC_CBW::bLUN
bCBLength	uint8_t _MSC_CBW::bCBLength
CB	uint8_t _MSC_CBW::CB[16][16]

42.4.14 _MSC_CSW

Table 605. _MSC_CSW class structure

Member	Description
dSignature	uint32_t _MSC_CSW::dSignature
dTag	uint32_t _MSC_CSW::dTag
dDataResidue	uint32_t _MSC_CSW::dDataResidue
bStatus	uint8_t _MSC_CSW::bStatus

42.4.15 _REQUEST_TYPE

Table 606. _REQUEST_TYPE class structure

Member	Description
B	uint8_t _REQUEST_TYPE::B byte wide access member
BM	BM_TBM_T _REQUEST_TYPE::BM bitfield structure access member

42.4.16 _USB_COMMON_DESCRIPTOR

Table 607. _USB_COMMON_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _USB_COMMON_DESCRIPTOR::bLength Size of this descriptor in bytes
bDescriptorType	uint8_t _USB_COMMON_DESCRIPTOR::bDescriptorType Descriptor Type

42.4.17 _USB_CORE_DESCS_T

USB descriptors data structure.

Table 608. _USB_CORE_DESCS_T class structure

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

42.4.18 _USB_DEVICE_QUALIFIER_DESCRIPTOR

Table 609. _USB_DEVICE_QUALIFIER_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bLength Size of descriptor
bDescriptorType	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDescriptorType Device Qualifier Type
bcdUSB	uint16_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bcdUSB USB specification version number (e.g., 0200H for V2.00)
bDeviceClass	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceClass Class Code
bDeviceSubClass	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceSubClass SubClass Code
bDeviceProtocol	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bDeviceProtocol Protocol Code
bMaxPacketSize0	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bMaxPacketSize0 Maximum packet size for other speed
bNumConfigurations	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bNumConfigurations Number of Other-speed Configurations
bReserved	uint8_t _USB_DEVICE_QUALIFIER_DESCRIPTOR::bReserved Reserved for future use, must be zero

42.4.19 _USB_DFU_FUNC_DESCRIPTOR

Table 610. _USB_DFU_FUNC_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _USB_DFU_FUNC_DESCRIPTOR::bLength
bDescriptorType	uint8_t _USB_DFU_FUNC_DESCRIPTOR::bDescriptorType
bmAttributes	uint8_t _USB_DFU_FUNC_DESCRIPTOR::bmAttributes
wDetachTimeOut	uint16_t _USB_DFU_FUNC_DESCRIPTOR::wDetachTimeOut
wTransferSize	uint16_t _USB_DFU_FUNC_DESCRIPTOR::wTransferSize
bcdDFUVersion	uint16_t _USB_DFU_FUNC_DESCRIPTOR::bcdDFUVersion

42.4.20 _USB_INTERFACE_DESCRIPTOR

Table 611. _USB_INTERFACE_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _USB_INTERFACE_DESCRIPTOR::bLength Size of this descriptor in bytes
bDescriptorType	uint8_t _USB_INTERFACE_DESCRIPTOR::bDescriptorType INTERFACE Descriptor Type
bInterfaceNumber	uint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceNumber Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
bAlternateSetting	uint8_t _USB_INTERFACE_DESCRIPTOR::bAlternateSetting Value used to select this alternate setting for the interface identified in the prior field
bNumEndpoints	uint8_t _USB_INTERFACE_DESCRIPTOR::bNumEndpoints Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the Default Control Pipe.
bInterfaceClass	uint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceClass Class code (assigned by the USB-IF).
bInterfaceSubClass	uint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceSubClass Subclass code (assigned by the USB-IF).
bInterfaceProtocol	uint8_t _USB_INTERFACE_DESCRIPTOR::bInterfaceProtocol Protocol code (assigned by the USB).
iInterface	uint8_t _USB_INTERFACE_DESCRIPTOR::iInterface Index of string descriptor describing this interface

42.4.21 _USB_OTHER_SPEED_CONFIGURATION

Table 612. _USB_OTHER_SPEED_CONFIGURATION class structure

Member	Description
bLength	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bLength Size of descriptor
bDescriptorType	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bDescriptorType Other_speed_Configuration Type
wTotalLength	uint16_t _USB_OTHER_SPEED_CONFIGURATION::wTotalLength Total length of data returned
bNumInterfaces	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bNumInterfaces Number of interfaces supported by this speed configuration
bConfigurationValue	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bConfigurationValue Value to use to select configuration
IConfiguration	uint8_t _USB_OTHER_SPEED_CONFIGURATION::IConfiguration Index of string descriptor
bmAttributes	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bmAttributes Same as Configuration descriptor
bMaxPower	uint8_t _USB_OTHER_SPEED_CONFIGURATION::bMaxPower Same as Configuration descriptor

42.4.22 _USB_SETUP_PACKET

Table 613. _USB_SETUP_PACKET class structure

Member	Description
bmRequestType	REQUEST_TYPE _USB_SETUP_PACKET::bmRequestType This bit-mapped field identifies the characteristics of the specific request. _BM_T.
bRequest	uint8_t _USB_SETUP_PACKET::bRequest This field specifies the particular request. The Type bits in the bmRequestType field modify the meaning of this field. USBD_REQUEST.
wValue	WORD_BYTE _USB_SETUP_PACKET::wValue Used to pass a parameter to the device, specific to the request.
wIndex	WORD_BYTE _USB_SETUP_PACKET::wIndex Used to pass a parameter to the device, specific to the request. The wIndex field is often used in requests to specify an endpoint or an interface.
wLength	uint16_t _USB_SETUP_PACKET::wLength This field specifies the length of the data transferred during the second phase of the control transfer.

42.4.23 _USB_STRING_DESCRIPTOR

Table 614. _USB_STRING_DESCRIPTOR class structure

Member	Description
bLength	uint8_t _USB_STRING_DESCRIPTOR::bLength Size of this descriptor in bytes
bDescriptorType	uint8_t _USB_STRING_DESCRIPTOR::bDescriptorType STRING Descriptor Type
bString	uint16_t _USB_STRING_DESCRIPTOR::bString UNICODE encoded string

42.4.24 _WB_T

Table 615. _WB_T class structure

Member	Description
L	uint8_t _WB_T::L lower byte
H	uint8_t _WB_T::H upper byte

42.4.25 USBD_API

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

Table 616. USBD_API class structure

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

Table 616. USBD_API class structure

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

42.4.26 USBD_API_INIT_PARAM

USB device stack initialization parameter data structure.

Table 617. USBD_API_INIT_PARAM class structure

Member	Description
usb_reg_base	uint32_t USBD_API_INIT_PARAM::usb_reg_base USB device controller's base register address.
mem_base	uint32_t USBD_API_INIT_PARAM::mem_base Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 2048 byte boundary.
mem_size	uint32_t USBD_API_INIT_PARAM::mem_size The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_HW_API::GetMemSize() routine.
max_num_ep	uint8_t USBD_API_INIT_PARAM::max_num_ep max number of endpoints supported by the USB device controller instance (specified by
pad0	uint8_t USBD_API_INIT_PARAM::pad0[3][3]
USB_Reset_Event	USB_CB_T USBD_API_INIT_PARAM::USB_Reset_Event Event for USB interface reset. This event fires when the USB host requests that the device reset its interface. This event fires after the control endpoint has been automatically configured by the library. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will prevent the device from enumerating correctly or operate properly.
USB_Suspend_Event	USB_CB_T USBD_API_INIT_PARAM::USB_Suspend_Event Event for USB suspend. This event fires when the USB host suspends the device by halting its transmission of Start Of Frame pulses to the device. This is generally hooked in order to move the device over to a low power state until the host wakes up the device. **Remark:** This event is called from USB_ISR context and hence is time-critical. Having delays in this callback will cause other system issues.

Table 617. USBD_API_INIT_PARAM class structure

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

Table 617. USBD_API_INIT_PARAM class structure

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

42.4.27 USBD_CDC_API

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

Table 618. USBD_CDC_API class structure

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

Table 618. USBD_CDC_API class structure

Member	Description
init	ErrorCode_t(*ErrorCode_t USBD_CDC_API::init)(USBD_HANDLE_T hUsb, USBD_CDC_INIT_PARAM_T *param, USBD_HANDLE_T *phCDC) Function to initialize CDC function driver module. This function is called by application layer to initialize CDC function driver module. hUsbHandle to the USB device stack. paramStructure containing CDC function driver module initialization parameters. Parameters: 1. hUsb = Handle to the USB device stack. 2. param = Structure containing CDC function driver module initialization parameters. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_BAD_MEM_BUF = Memory buffer passed is not 4-byte aligned or smaller than required. 3. ERR_API_INVALID_PARAM2 = Either CDC_Write() or CDC_Read() or CDC_Verify() callbacks are not defined. 4. ERR_USBD_BAD_INTF_DESC = Wrong interface descriptor is passed. 5. ERR_USBD_BAD_EP_DESC = Wrong endpoint descriptor is passed.
SendNotification	ErrorCode_t(*ErrorCode_t USBD_CDC_API::SendNotification)(USBD_HANDLE_T hCdc, uint8_t bNotification, uint16_t data) Function to send CDC class notifications to host. This function is called by application layer to send CDC class notifications to host. See usbcdc11.pdf, section 6.3, Table 67 for various notification types the CDC device can send. **Remark:** The current version of the driver only supports following notifications allowed by ACM subclass: CDC_NOTIFICATION_NETWORK_CONNECTION, CDC_RESPONSE_AVAILABLE, CDC_NOTIFICATION_SERIAL_STATE. For all other notifications application should construct the notification buffer appropriately and call hw->USB_WriteEP() for interrupt endpoint associated with the interface. Parameters: 1. hCdc = Handle to CDC function driver. 2. bNotification = Notification type allowed by ACM subclass. Should be CDC_NOTIFICATION_NETWORK_CONNECTION, CDC_RESPONSE_AVAILABLE or CDC_NOTIFICATION_SERIAL_STATE. For all other types ERR_API_INVALID_PARAM2 is returned. See usbcdc11.pdf, section 3.6.2.1, table 5. 3. data = Data associated with notification. For CDC_NOTIFICATION_NETWORK_CONNECTION a non-zero data value is interpreted as connected state. For CDC_RESPONSE_AVAILABLE this parameter is ignored. For CDC_NOTIFICATION_SERIAL_STATE the data should use bitmap values defined in usbcdc11.pdf, section 6.3.5, Table 69. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_API_INVALID_PARAM2 = If unsupported notification type is passed.

42.4.28 USBD_CDC_INIT_PARAM

Communication Device Class function driver initialization parameter data structure.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
mem_base	uint32_t USBD_CDC_INIT_PARAM::mem_base Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	uint32_t USBD_CDC_INIT_PARAM::mem_size The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_CDC_API::GetMemSize() routine.
cif_intf_desc	uint8_t * USBD_CDC_INIT_PARAM::cif_intf_desc Pointer to the control interface descriptor within the descriptor array
dif_intf_desc	uint8_t * USBD_CDC_INIT_PARAM::dif_intf_desc Pointer to the data interface descriptor within the descriptor array
CIC_GetRequest	ErrorCode_t(* USBD_CDC_INIT_PARAM::CIC_GetRequest)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length) Communication Interface Class specific get request call-back function. This function is provided by the application software. This function gets called when host sends CIC management element get requests. **Remark:** Applications implementing Abstract Control Model subclass can set this param to NULL. As the default driver parses ACM requests and calls the individual ACM call-back routines defined in this structure. For all other subclasses this routine should be provided by the application. The setup packet data (pSetup) is passed to the call-back so that application can extract the CIC request type and other associated data. By default the stack will assign pBuffer pointer to EP0Buff allocated at init. The application code can directly write data into this buffer as long as data is less than 64 byte. If more data has to be sent then application code should update pBuffer pointer and length accordingly. Parameters: 1. hCdc = Handle to CDC function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See USBD_ZeroCopy for more details on zero-copy concept. 4. length = Amount of data to be sent back to host. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
CIC_SetRequest	ErrorCode_t(* USBD_CDC_INIT_PARAM::CIC_SetRequest)(USBD_HANDLE_T hCdc, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length) Communication Interface Class specific set request call-back function. This function is provided by the application software. This function gets called when host sends a CIC management element requests. **Remark:** Applications implementing Abstract Control Model subclass can set this param to NULL. As the default driver parses ACM requests and calls the individual ACM call-back routines defined in this structure. For all other subclasses this routine should be provided by the application. The setup packet data (pSetup) is passed to the call-back so that application can extract the CIC request type and other associated data. If a set request has data associated, then this call-back is called twice. (1) First when setup request is received, at this time application code could update pBuffer pointer to point to the intended destination. The length param is set to 0 so that application code knows this is first time. By default the stack will assign pBuffer pointer to EP0Buff allocated at init. Note, if data length is greater than 64 bytes and application code doesn't update pBuffer pointer the stack will send STALL condition to host. (2) Second when the data is received from the host. This time the length param is set with number of data bytes received. Parameters: 1. hCdc = Handle to CDC function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing request data. Pointer-to-pointer is used to implement zero-copy buffers. See USBD_ZeroCopy for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
CDC_BulkIN_Hdlr	ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_BulkIN_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event) Communication Device Class specific BULK IN endpoint handler. The application software should provide the BULK IN endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
CDC_BulkOUT_Hdlr	ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_BulkOUT_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event))(USBD_HANDLE_T hUsb, void *data, uint32_t event) Communication Device Class specific BULK OUT endpoint handler. The application software should provide the BULK OUT endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. **Remark:** Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
SendEncpsCmd	ErrorCode_t(* USBD_CDC_INIT_PARAM::SendEncpsCmd)(USBD_HANDLE_T hCDC, uint8_t *buffer, uint16_t len) Abstract control model(ACM) subclass specific SEND_ENCAPSULATED_COMMAND request call-back function. This function is provided by the application software. This function gets called when host sends a SEND_ENCAPSULATED_COMMAND set request. Parameters: 1. hCdc = Handle to CDC function driver. 2. buffer = Pointer to the command buffer. 3. len = Length of the command buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
GetEncpsResp	ErrorCode_t(* USBD_CDC_INIT_PARAM::GetEncpsResp)(USBD_HANDLE_T hCDC, uint8_t **buffer, uint16_t *len) Abstract control model(ACM) subclass specific GET_ENCAPSULATED_RESPONSE request call-back function. This function is provided by the application software. This function gets called when host sends a GET_ENCAPSULATED_RESPONSE request. Parameters: 1. hCdc = Handle to CDC function driver. 2. buffer = Pointer to a pointer of data buffer containing response data. Pointer-to-pointer is used to implement zero-copy buffers. See USBD_ZeroCopy for more details on zero-copy concept. 3. len = Amount of data to be sent back to host. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
SetCommFeature	ErrorCode_t(* USBD_CDC_INIT_PARAM::SetCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t *buffer, uint16_t len) Abstract control model(ACM) subclass specific SET_COMM_FEATURE request call-back function. This function is provided by the application software. This function gets called when host sends a SET_COMM_FEATURE set request. Parameters: 1. hCdc = Handle to CDC function driver. 2. feature = Communication feature type. 3. buffer = Pointer to the settings buffer for the specified communication feature. 4. len = Length of the request buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
GetCommFeature	ErrorCode_t(* USBD_CDC_INIT_PARAM::GetCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature, uint8_t **pBuffer, uint16_t *len) Abstract control model(ACM) subclass specific GET_COMM_FEATURE request call-back function. This function is provided by the application software. This function gets called when host sends a GET_ENCAPSULATED_RESPONSE request. Parameters: 1. hCdc = Handle to CDC function driver. 2. feature = Communication feature type. 3. buffer = Pointer to a pointer of data buffer containing current settings for the communication feature. Pointer-to-pointer is used to implement zero-copy buffers. 4. len = Amount of data to be sent back to host. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
ClrCommFeature	ErrorCode_t(* USBD_CDC_INIT_PARAM::ClrCommFeature)(USBD_HANDLE_T hCDC, uint16_t feature) Abstract control model(ACM) subclass specific CLEAR_COMM_FEATURE request call-back function. This function is provided by the application software. This function gets called when host sends a CLEAR_COMM_FEATURE request. In the call-back the application should Clears the settings for a particular communication feature. Parameters: 1. hCdc = Handle to CDC function driver. 2. feature = Communication feature type. See usbcdc11.pdf, section 6.2.4, Table 47. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
SetCtrlLineState	ErrorCode_t(* USBD_CDC_INIT_PARAM::SetCtrlLineState)(USBD_HANDLE_T hCDC, uint16_t state) Abstract control model(ACM) subclass specific SET_CONTROL_LINE_STATE request call-back function. This function is provided by the application software. This function gets called when host sends a SET_CONTROL_LINE_STATE request. RS-232 signal used to tell the DCE device the DTE device is now present Parameters: 1. hCdc = Handle to CDC function driver. 2. state = The state value uses bitmap values defined the *USB CDC class specification document* published by usb.org. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
SendBreak	ErrorCode_t(* USBD_CDC_INIT_PARAM::SendBreak)(USBD_HANDLE_T hCDC, uint16_t mstime) Abstract control model(ACM) subclass specific SEND_BREAK request call-back function. This function is provided by the application software. This function gets called when host sends a SEND_BREAK request. Parameters: 1. hCdc = Handle to CDC function driver. 2. mstime = Duration of Break signal in milliseconds. If mstime is FFFFh, then the application should send break until another SendBreak request is received with the wValue of 0000h. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

Member	Description
SetLineCode	ErrorCode_t(* USBD_CDC_INIT_PARAM::SetLineCode)(USBD_HANDLE_T hCDC, CDC_LINE_CODING *line_coding) Abstract control model(ACM) subclass specific SET_LINE_CODING request call-back function. This function is provided by the application software. This function gets called when host sends a SET_LINE_CODING request. The application should configure the device per DTE rate, stop-bits, parity, and number-of-character bits settings provided in command buffer. Parameters: 1. hCdc = Handle to CDC function driver. 2. line_coding = Pointer to the CDC_LINE_CODING command buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
CDC_InterruptEP_Hdlr	ErrorCode_t(* USBD_CDC_INIT_PARAM::CDC_InterruptEP_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event) Optional Communication Device Class specific INTERRUPT IN endpoint handler. The application software should provide the INT IN endpoint handler. Applications should transfer data depending on the communication protocol type set in descriptors. Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Pointer to the data which will be passed when callback function is called by the stack. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 619. USBD_CDC_INIT_PARAM class structure

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

42.4.29 USBD_CORE_API

USBD stack Core API functions structure.

Table 620. USBD_CORE_API class structure

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

Table 620. USBD_CORE_API class structure

Member	Description
RegisterEpHandler	ErrorCode_t(*ErrorCode_t USBD_CORE_API::RegisterEpHandler)(USBD_HANDLE_T hUsb, uint32_t ep_index, USB_EP_HANDLER_T pfn, void *data) Function to register interrupt/event handler for the requested endpoint with USB device stack. The application layer uses this function to register the custom class's EP0 handler. The stack calls all the registered class handlers on any EP0 event before going through default handling of the event. This gives the class handlers to implement class specific request handlers and also to override the default stack handling for a particular event targeted to the interface. Check USB_EP_HANDLER_T for more details on how the callback function should be implemented. Parameters: 1. hUsb = Handle to the USB device stack. 2. ep_index = Class specific EP0 handler function. 3. pfn = Class specific EP0 handler function. 4. data = Pointer to the data which will be passed when callback function is called by the stack. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success 2. ERR_USBD_TOO_MANY_CLASS_HDLR(0x0004000c) = Too many endpoint handlers.
SetupStage	void(*void USBD_CORE_API::SetupStage)(USBD_HANDLE_T hUsb) Function to set EP0 state machine in setup state. This function is called by USB stack and the application layer to set the EP0 state machine in setup state. This function will read the setup packet received from USB host into stack's buffer. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.
DataInStage	void(*void USBD_CORE_API::DataInStage)(USBD_HANDLE_T hUsb) Function to set EP0 state machine in data_in state. This function is called by USB stack and the application layer to set the EP0 state machine in data_in state. This function will write the data present in EP0Data buffer to EP0 FIFO for transmission to host. **Remark:** This interface is provided to users to invoke this function in other scenarios which are not handle by current stack. In most user applications this function is not called directly.Also this function can be used by users who are selectively modifying the USB device stack's standard handlers through callback interface exposed by the stack. Parameters: 1. hUsb = Handle to the USB device stack. Returns: Nothing.

Table 620. USBD_CORE_API class structure

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

Table 620. USBD_CORE_API class structure

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

42.4.30 USBD_DFU_API

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

Table 621. USBD_DFU_API class structure

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

42.4.31 USBD_DFU_INIT_PARAM

USB descriptors data structure.

Table 622. USBD_DFU_INIT_PARAM class structure

Member	Description
mem_base	uint32_t USBD_DFU_INIT_PARAM::mem_base Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	uint32_t USBD_DFU_INIT_PARAM::mem_size The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_DFU_API::GetMemSize() routine.

Table 622. USBD_DFU_INIT_PARAM class structure

Member	Description
wTransferSize	uint16_t USBD_DFU_INIT_PARAM::wTransferSize DFU transfer block size in number of bytes. This value should match the value set in DFU descriptor provided as part of the descriptor array (
pad	uint16_t USBD_DFU_INIT_PARAM::pad
intf_desc	uint8_t * USBD_DFU_INIT_PARAM::intf_desc Pointer to the DFU interface descriptor within the descriptor array (
DFU_Write	uint8_t(*uint8_t(* USBD_DFU_INIT_PARAM::DFU_Write)(uint32_t block_num, uint8_t **src, uint32_t length, uint8_t *bwPollTimeout))(uint32_t block_num, uint8_t **src, uint32_t length, uint8_t *bwPollTimeout) DFU Write callback function. This function is provided by the application software. This function gets called when host sends a write command. For application using zero-copy buffer scheme this function is called for the first time with Parameters: 1. block_num = Destination start address. 2. src = Pointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. bwPollTimeout = Pointer to a 3 byte buffer which the callback implementer should fill with the amount of minimum time, in milliseconds, that the host should wait before sending a subsequent DFU_GETSTATUS request. 4. length = Number of bytes to be written. Returns: Returns DFU_STATUS_ values defined in mw_usbd_dfu.h.
DFU_Read	uint32_t(*uint32_t(* USBD_DFU_INIT_PARAM::DFU_Read)(uint32_t block_num, uint8_t **dst, uint32_t length))(uint32_t block_num, uint8_t **dst, uint32_t length) DFU Read callback function. This function is provided by the application software. This function gets called when host sends a read command. Parameters: 1. block_num = Destination start address. 2. dst = Pointer to a pointer to the source of data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 3. length = Amount of data copied to destination buffer. Returns: Returns DFU_STATUS_ values defined in mw_usbd_dfu.h.
DFU_Done	void(*USBD_DFU_INIT_PARAM::DFU_Done)(void) DFU done callback function. This function is provided by the application software. This function gets called after download is finished. Nothing. Returns: Nothing.

Table 622. USBD_DFU_INIT_PARAM class structure

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

42.4.32 USBD_HID_API

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

Table 623. USBD_HID_API class structure

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

42.4.33 USBD_HID_INIT_PARAM

USB descriptors data structure.

Table 624. USBD_HID_INIT_PARAM class structure

Member	Description
mem_base	uint32_t USBD_HID_INIT_PARAM::mem_base Base memory location from where the stack can allocate data and buffers. **Remark:** The memory address set in this field should be accessible by USB DMA controller. Also this value should be aligned on 4 byte boundary.
mem_size	uint32_t USBD_HID_INIT_PARAM::mem_size The size of memory buffer which stack can use. **Remark:** The mem_size should be greater than the size returned by USBD_HID_API::GetMemSize() routine.
max_reports	uint8_t USBD_HID_INIT_PARAM::max_reports Number of HID reports supported by this instance of HID class driver.
pad	uint8_t USBD_HID_INIT_PARAM::pad[3][3]
intf_desc	uint8_t * USBD_HID_INIT_PARAM::intf_desc Pointer to the HID interface descriptor within the descriptor array (
report_data	USB_HID_REPORT_T *USB_HID_REPORT_T* USBD_HID_INIT_PARAM::report_data Pointer to an array of HID report descriptor data structure (**Remark:** This array should be of global scope.
HID_GetReport	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_GetReport)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t *length) HID get report callback function. This function is provided by the application software. This function gets called when host sends a HID_REQUEST_GET_REPORT request. The setup packet data (**Remark:** HID reports are sent via interrupt IN endpoint also. This function is called only when report request is received on control endpoint. Application should implement HID_EpIn_Hdlr to send reports to host via interrupt IN endpoint. Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing report data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 624. USBD_HID_INIT_PARAM class structure

Member	Description
HID_SetReport	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetReport)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuffer, uint16_t length) HID set report callback function. This function is provided by the application software. This function gets called when host sends a HID_REQUEST_SET_REPORT request. The setup packet data (Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuffer = Pointer to a pointer of data buffer containing report data. Pointer-to-pointer is used to implement zero-copy buffers. See Zero-Copy Data Transfer model for more details on zero-copy concept. 4. length = Amount of data copied to destination buffer. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
HID_GetPhysDesc	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_GetPhysDesc)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t **pBuf, uint16_t *length) Optional callback function to handle HID_GetPhysDesc request. The application software could provide this callback HID_GetPhysDesc handler to handle get physical descriptor requests sent by the host. When host requests Physical Descriptor set 0, application should return a special descriptor identifying the number of descriptor sets and their sizes. A Get_Descriptor request with the Physical Index equal to 1 should return the first Physical Descriptor set. A device could possibly have alternate uses for its items. These can be enumerated by issuing subsequent Get_Descriptor requests while incrementing the Descriptor Index. A device should return the last descriptor set to requests with an index greater than the last number defined in the HID descriptor. **Remark:** Applications which don't have physical descriptor should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet received from host. 3. pBuf = Pointer to a pointer of data buffer containing physical descriptor data. If the physical descriptor is in USB accessible memory area application could just update the pointer or else it should copy the descriptor to the address pointed by this pointer. 4. length = Amount of data copied to destination buffer or descriptor length. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 624. USBD_HID_INIT_PARAM class structure

Member	Description
HID_SetIdle	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetIdle)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t idleTime) Optional callback function to handle HID_REQUEST_SET_IDLE request. The application software could provide this callback to handle HID_REQUEST_SET_IDLE requests sent by the host. This callback is provided to applications to adjust timers associated with various reports, which are sent to host over interrupt endpoint. The setup packet data (**Remark:** Applications which don't send reports on Interrupt endpoint or don't have idle time between reports should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet recived from host. 3. idleTime = Idle time to be set for the specified report. Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.
HID_SetProtocol	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_SetProtocol)(USBD_HANDLE_T hHid, USB_SETUP_PACKET *pSetup, uint8_t protocol) Optional callback function to handle HID_REQUEST_SET_PROTOCOL request. The application software could provide this callback to handle HID_REQUEST_SET_PROTOCOL requests sent by the host. This callback is provided to applications to adjust modes of their code between boot mode and report mode. **Remark:** Applications which don't support protocol modes should set this data member to zero before calling the USBD_HID_API::Init(). Parameters: 1. hHid = Handle to HID function driver. 2. pSetup = Pointer to setup packet recived from host. 3. protocol = Protocol mode. 0 = Boot Protocol 1 = Report Protocol Returns: The call back should returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 624. USBD_HID_INIT_PARAM class structure

Member	Description
HID_EpIn_Hdlr	ErrorCode_t(* USBD_HID_INIT_PARAM::HID_EpIn_Hdlr)(USBD_HANDLE_T hUsb, void *data, uint32_t event) Optional Interrupt IN endpoint event handler. The application software could provide Interrupt IN endpoint event handler. Application which send reports to host on interrupt endpoint should provide an endpoint event handler through this data member. This data member is ignored if the interface descriptor Parameters: 1. hUsb = Handle to the USB device stack. 2. data = Handle to HID function driver. 3. event = Type of endpoint event. See USBD_EVENT_T for more details. Returns: The call back should return ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK = On success. 2. ERR_USBD_UNHANDLED = Event is not handled hence pass the event to next in line. 3. ERR_USBD_xxx = For other error conditions.

Table 624. USBD_HID_INIT_PARAM class structure

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

Table 624. USBD_HID_INIT_PARAM class structure

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

42.4.34 USBD_HW_API

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

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

Member	Description
SetStallEP	void(*void USBD_HW_API::SetStallEP)(USBD_HANDLE_T hUsb, uint32_t EPNum) Function to STALL selected USB endpoint. Generates STALL signalling for requested endpoint. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.
ClrStallEP	void(*void USBD_HW_API::ClrStallEP)(USBD_HANDLE_T hUsb, uint32_t EPNum) Function to clear STALL state for the requested endpoint. This function clears STALL state for the requested endpoint. Parameters: 1. hUsb = Handle to the USB device stack. 2. EPNum = Endpoint number as per USB specification. ie. An EP1_IN is represented by 0x81 number. Returns: Nothing.
SetTestMode	ErrorCode_t(*ErrorCode_t USBD_HW_API::SetTestMode)(USBD_HANDLE_T hUsb, uint8_t mode) Function to set high speed USB device controller in requested test mode. USB-IF requires the high speed device to be put in various test modes for electrical testing. This USB device stack calls this function whenever it receives USB_REQUEST_CLEAR_FEATURE request for USB_FEATURE_TEST_MODE. Users can put the device in test mode by directly calling this function. Returns ERR_USBD_INVALID_REQ when device controller is full-speed only. Parameters: 1. hUsb = Handle to the USB device stack. 2. mode = Test mode defined in USB 2.0 electrical testing specification. Returns: Returns ErrorCode_t type to indicate success or error condition. Return values: 1. LPC_OK(0) = - On success 2. ERR_USBD_INVALID_REQ(0x00040001) = - Invalid test mode or Device controller is full-speed only.

Table 625. USBD_HW_API class structure

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

Table 625. USBD_HW_API class structure

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

42.4.35 USBD_MSC_API

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

Table 626. USBD_MSC_API class structure

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

42.4.36 USBD_MSC_INIT_PARAM

Mass Storage class function driver initialization parameter data structure.

Table 627. USBD_MSC_INIT_PARAM class structure

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

Table 627. USBD_MSC_INIT_PARAM class structure

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

Table 627. USBD_MSC_INIT_PARAM class structure

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

Table 627. USBD_MSC_INIT_PARAM class structure

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

43.1 How to read this chapter

The debug functionality is identical for all parts.

43.2 Features

- Supports ARM Serial Wire Debug mode.
- Trace port provides CPU instruction trace capability. Output via a Serial Wire Viewer.
- Direct debug access to all memories, registers, and peripherals.
- No target resources are required for the debugging session.
- Four breakpoints. Four instruction breakpoints that can also be used to remap instruction addresses for code patches. Two data comparators that can be used to remap addresses for patches to literal values.
- Two data watchpoints that can also be used as triggers.
- Supports JTAG boundary scan.
- Instrumentation Trace Macrocell allows additional software controlled trace.

43.3 Basic configuration

The serial wire debug pins are enabled by default. The JTAG pins for boundary scan are selected by hardware after a reset.

43.4 Pin description

The tables below indicate the various pin functions related to debug. Some of these functions share pins with other functions which therefore may not be used at the same time. Trace using the Serial Wire Output has limited bandwidth.

Table 628. Serial Wire Debug pin description

Function	Direction	Connect to	Use register	Reference	Description
SWCLK	Input	PIO0_19	PINENABLE1	Table 124	**Serial Wire Clock.** This pin is the clock for SWD debug logic when in the Serial Wire Debug mode (SWD). This pin is pulled up internally.
SWDIO	Input / Output	PIO0_20	PINENABLE1	Table 124	**Serial wire debug data input/output.** The SWDIO pin is used by an external debug tool to communicate with and control the part. This pin is pulled up internally.
SWO	Output	any pin	PINASSIGN15	Table 122	**Serial Wire Output.** The SWO pin optionally provides data from the ITM for an external debug tool to evaluate.

The JTAG boundary pin functions are selected by hardware at reset. See Section 43.6.3.

Table 629. JTAG boundary scan pin description

Function	Direction	Description
TCK	Input	**JTAG Test Clock.** This pin is the clock for JTAG boundary scan when the RESET pin is LOW.
TMS	Input	**JTAG Test Mode Select.** The TMS pin selects the next state in the TAP state machine. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.
TDI	Input	**JTAG Test Data In.** This is the serial data input for the shift register. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.
TDO	Output	**JTAG Test Data Output.** This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal. This pin is used for JTAG boundary scan when the RESET pin is LOW.
TRST	Input	**JTAG Test Reset.** The TRST pin can be used to reset the test logic within the debug logic. This pin includes an internal pull-up and is used for JTAG boundary scan when the RESET pin is LOW.

43.5 General description

Debug functions are integrated into the ARM Cortex-M3. Serial wire debug functions are supported. The ARM Cortex-M3 is configured to support up to four breakpoints and two watchpoints.

Debugging uses the Serial Wire Debug mode. Support for boundary scan is available.

Trace is supported via the Serial Wire Output.

43.6 Functional description

43.6.1 Debug limitations

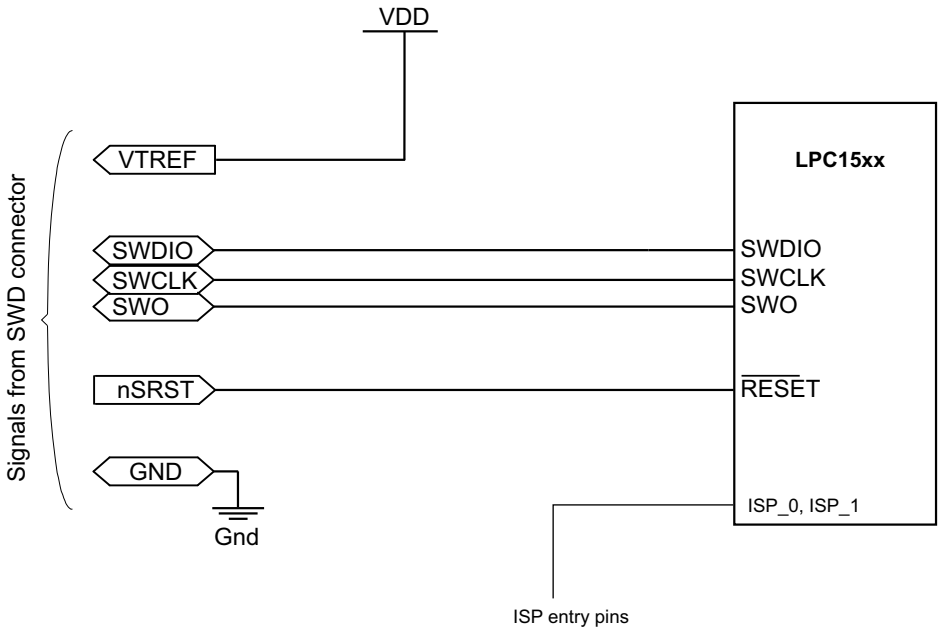
Important: Due to limitations of the ARM Cortex-M3 integration, the part cannot wake up in the usual manner from Deep-sleep mode. Do not use the debugger in Deep-sleep or Power-down modes.

The debug mode changes the way in which reduced power modes work internal to the ARM Cortex-M3 CPU. Therefore power measurements should not be made while debugging, and power consumption is higher than during normal operation.

During a debugging session, the System Tick Timer is automatically stopped whenever the CPU is stopped. Other peripherals are not affected.

43.6.2 Debug connections for SWD

For debugging purposes, it is useful to provide access to the ISP entry pins (see Table 90 “ISP pin assignments”). The ISP entry pins can be used to recover the part from configurations which would disable the SWD port such as improper PLL configuration, assigning another function to the SWD pins through the switch matrix, entry into Deep power-down mode out of reset, etc. The ISP entry pins can be used for other functions such as GPIO but 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 105. Connecting the SWD pins to a standard SWD connector

43.6.3 Boundary scan

The $\overline{\text{RESET}}$ pin selects between the test TAP controller for JTAG boundary scan ($\overline{\text{RESET}}$ = LOW) and the ARM SWD debug port TAP controller ($\overline{\text{RESET}}$ = HIGH). The ARM SWD debug port is disabled while the part is in reset. A LOW on the $\overline{\text{TRST}}$ pin resets the test TAP controller.

Remark: Boundary scan operations should not be started until 250 μs after POR. The test TAP must be reset after the boundary scan and left in either TLR or RTO state. Boundary scan is not affected by Code Read Protection.

Remark: POR, BOD reset, or a LOW on the $\overline{\text{TRST}}$ pin puts the test TAP controller in the Test-Logic Reset state. The first TCK clock while $\overline{\text{RESET}}$ = HIGH places the test TAP in Run-Test Idle mode.

44.1 How to read this chapter

This chapter contains code examples to help understand how to use the registers of various peripheral blocks when writing software drivers. For a comprehensive description of each peripheral and register interface, see the respective chapter.

Remark: The code listings are for illustrative purposes only and are not intended to be application-ready functions.

44.2 Code examples I2C

44.2.1 Definitions

Table 630. I2C Code example

I2C defines

```
#define I2C_CFG_MSTEN (0x1)
#define I2C_CFG_SLVEN (0x2)
#define I2C_STAT_MSTPENDING (0x1)
#define I2C_STAT_MSTSTATE (0xe)
#define I2C_STAT_MSTST_IDLE (0x0)
#define I2C_STAT_MSTST_RX (0x2)
#define I2C_STAT_MSTST_TX (0x4)
#define I2C_STAT_MSTST_NACK_ADDR (0x6)
#define I2C_STAT_MSTST_NACK_TX (0x8)
#define I2C_STAT_SLVPENDING (0x100)
#define I2C_STAT_SLVSTATE (0x600)
#define I2C_STAT_SLVST_ADDR (0x000)
#define I2C_STAT_SLVST_RX (0x200)
#define I2C_STAT_SLVST_TX (0x400)
#define I2C_MSTCTL_MSTCONTINUE (0x1)
#define I2C_MSTCTL_MSTSTART (0x2)
#define I2C_MSTCTL_MSTSTOP (0x4)
#define I2C_SLVCTL_SLVCONTINUE (0x1)
#define I2C_SLVCTL_SLVNACK (0x2)
```

44.2.2 Interrupt handler

Table 631. I2C Code example

Interrupt handler

```
void I2c_IRQHandler() {
uint32_t intstat = LPC_I2C->INTSTAT;
      uint32_t stat = LPC_I2C->STAT;
      if(intstat & I2C_STAT_MSTPENDING) {
            uint32_t mst_state = stat & I2C_STAT_MSTSTATE;
            if(mst_state == I2C_STAT_MSTST_IDLE) {
                  LPC_I2C->MSTDAT = (0x23 << 1) | 1; // address and 1 for RWn bit in
      order
// to read data
                  LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
            }
            if(mst_state == I2C_STAT_MSTST_RX) {
                  uint8_t data;
                  data = LPC_I2C->MSTDAT; // receive data
                  if(data != 0xdd) abort();
                  LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 1 for RWn bit in
      order
// to read data
                  LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // repeated start (nack
      implied)
            }
            if(mst_state == I2C_STAT_MSTST_TX) {
                  LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // stop transaction
                  LPC_I2C->INTENCLR = I2C_STAT_MSTPENDING;
            }
      }
}
```

44.2.3 Master write one byte to slave

Table 632. I2C Code example

Master write one byte to slave. Address 0x23, Data 0xdd. Polling mode.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 0 for RWn bit
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = 0xdd; // send data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // send stop
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.4 Master read one byte from slave

Table 633. I2C Code example

Master read one byte from slave. Address 0x23. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 1; // address and 1 for RWn bit
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_RX) abort();
data = LPC_I2C->MSTDAT; // read data
if(data != 0xdd) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // send stop
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.5 Master write one byte to subaddress on slave

Table 634. I2C Code example

Master write one byte to subaddress on slave. Address 0x23, subaddress 0xaa, Data 0xdd. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 0 for RWn bit in order to write
// subaddress
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = 0xaa; // send subaddress
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = 0xdd; // send data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // send stop
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.6 Master read one byte from subaddress on slave

Table 635. I2C Code example

Master read one byte from subaddress on slave. Address 0x23, subaddress 0xaa. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 0 for RWn bit in order to write
// subaddress
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = 0xaa; // send subaddress
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 1; // address and 1 for RWn bit in order to write
// subaddress
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send repeated start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_RX) abort();
data = LPC_I2C->MSTDAT; // read data
if(data != 0xdd) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // send stop
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.7 Master receiving nack on address

Table 636. I2C Code example

Master receive nack on address. Address 0x23. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 0 for RWn bit
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_NACK_ADDR) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // stop transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.8 Master receiving nack on data

Table 637. I2C Code example

Master receive nack on data. Address 0x23, data 0xdd. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 0 for RWn bit in order to write data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_TX) abort();
LPC_I2C->MSTDAT = 0xdd; // send data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_NACK_TX) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // stop transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.9 Master sending nack and stop on data

Table 638. I2C Code example

Master sending nack and stop on data. Address 0x23. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 1; // address and 1 for RWn bit in order to read data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_RX) abort();
data = LPC_I2C->MSTDAT; // receive data
if(data != 0xdd) abort();
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // stop transaction (nack implied)
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.10 Master sending nack and repeated start on data

Table 639. I2C Code example

Master sending nack and repeated start on data. Address 0x23. Polling mode. No error checking.

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 1; // address and 1 for RWn bit in order to read data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // send start
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_RX) abort();
data = LPC_I2C->MSTDAT; // receive data
if(data != 0xdd) abort();
LPC_I2C->MSTDAT = (0x23 << 1) | 0; // address and 1 for RWn bit in order to read data
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTART; // repeated start (nack implied)
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
LPC_I2C->MSTCTL = I2C_MSTCTL_MSTSTOP; // stop transaction
while(!(LPC_I2C->STAT & I2C_STAT_MSTPENDING));
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
```

44.2.11 Master sending nack and repeated start on data. Interrupt mode

Table 640. I2C Code example

Master sending nack and repeated start on data. Address 0x23. No error checking. Interrupt mode

```
LPC_I2C->CFG = I2C_CFG_MSTEN;
LPC_I2C->INTENSET = I2C_STAT_MSTPENDING;
NVIC_EnableIRQ(I2c_IRQn);
while(LPC_I2C->INTENSET & I2C_STAT_MSTPENDING);
if((LPC_I2C->STAT & I2C_STAT_MSTSTATE) != I2C_STAT_MSTST_IDLE) abort();
NVIC_DisableIRQ(I2c_IRQn);
```

44.2.12 Slave read one byte from master

Table 641. I2C Code example

Slave read one byte from master. Address 0x23. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_RX) abort();
data = LPC_I2C->SLVDAT; // read data
if(data != 0xdd) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack data
```

44.2.13 Slave write one byte to master

Table 642. I2C Code example

Slave write one byte to master. Address 0x23, Data 0xdd. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_TX) abort();
LPC_I2C->SLVDAT = 0xdd; // write data
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // continue transaction
```

44.2.14 Slave read one byte from master into subaddress

Table 643. I2C Code example

Slave read one byte from master into subaddress. Address 0x23. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_RX) abort();
subaddress = LPC_I2C->SLVDAT; // read subaddress
if(subaddress != 0xaa) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack data
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_RX) abort();
data[subaddress] = LPC_I2C->SLVDAT; // read data into subaddress
if(data[subaddress] != 0xdd) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack data
```

44.2.15 Slave write one byte to master from subaddress

Table 644. I2C Code example

Slave write one byte to master from subaddress. Address 0x23. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_RX) abort();
subaddress = LPC_I2C->SLVDAT; // read subaddress
if(subaddress != 0xaa) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // continue transaction
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_TX) abort();
LPC_I2C->SLVDAT = data[subaddress]; // write data from subaddress
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // continue transaction
```

44.2.16 Slave nack matched address from master

Table 645. I2C Code example

Slave nack matched address from master. Address 0x23. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVNACK; // nack address
```

44.2.17 Slave nack data from master

Table 646. I2C Code example

Slave nack data from master. Address 0x23. Polling mode.

```
LPC_I2C->SLVADR0 = 0x23 << 1; // put address in address 0 register
LPC_I2C->CFG = I2C_CFG_SLVEN;
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_ADDR) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVCONTINUE; // ack address
while(!(LPC_I2C->STAT & I2C_STAT_SLVPENDING));
if((LPC_I2C->STAT & I2C_STAT_SLVSTATE) != I2C_STAT_SLVST_RX) abort();
data = LPC_I2C->SLVDAT; // read data
if(data != 0xdd) abort();
LPC_I2C->SLVCTL = I2C_SLVCTL_SLVNACK; // nack data
```

44.3 Code examples SPI

44.3.1 Definitions

Table 647. Code example

SPI defines

```
#define SPI_CFG_ENABLE (0x1)
#define SPI_CFG_MASTER (0x4)
#define SPI_STAT_RXRDY (0x1)
#define SPI_STAT_TXRDY (0x2)
#define SPI_STAT_SSD (0x20)
#define SPI_STAT_MSTIDLE (0x100)
#define SPI_TXDATCTL_SSEL_N(s) ((s) << 16)
#define SPI_TXDATCTL_EOT (1 << 20)
#define SPI_TXDATCTL_EOF (1 << 21)
#define SPI_TXDATCTL_RXIGNORE (1 << 22)
#define SPI_TXDATCTL_FLEN(l) ((l) << 24)
```

44.3.2 Interrupt handler

Table 648. Code example

Interrupt handler

```
void Spi_IRQHandler() {
      uint16_t data;
      uint32_t intstat = LPC_SPI->INTSTAT;
      if(intstat & SPI_STAT_TXRDY) {
            if(tx_state == 0) {
                  LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(15) |
      SPI_TXDATCTL_SSEL_N(0xe) | 0xdddd;
                  tx_state++;
            }
            if(tx_state == 1) {
                  LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_SSEL_N(0xe)
      | 0xdd;
                  LPC_SPI->INTENCLR = SPI_STAT_TXRDY;
            }
      }
      if(intstat & SPI_STAT_RXRDY) {
            if(rx_state == 0) {
                  data = LPC_SPI->RXDAT;
                  if(data != 0xdddd) abort();
                  rx_state++;
            }
            if(rx_state == 1) {
                  data = LPC_SPI->RXDAT;
                  if(data != 0xdd) abort();
                  LPC_SPI->INTENCLR = SPI_STAT_RXRDY;
            }
      }
}
```

44.3.3 Transmit one byte to slave 0

Table 649. SPI Code example

Transmit one byte to slave 0.

```
LPC_SPI->CFG = SPI_CFG_MASTER | SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_RXIGNORE | SPI_TXDATCTL_EOT
| SPI_TXDATCTL_SSEL_N(0xe) | 0xdd;
while(~LPC_SPI->STAT & SPI_STAT_MSTIDLE);
```

44.3.4 Receive one byte from slave 0

Table 650. SPI Code example

Receive one byte from slave 0.

```
LPC_SPI->CFG = SPI_CFG_MASTER | SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_EOT |
SPI_TXDATCTL_SSEL_N(0xe);
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdd) abort();
while(~LPC_SPI->STAT & SPI_STAT_MSTIDLE);
```

44.3.5 Transmit and receive a byte to/from slave 0

Table 651. SPI Code example

Transmit and receive a byte to/from slave 0.

```
LPC_SPI->CFG = SPI_CFG_MASTER | SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_EOT |
 SPI_TXDATCTL_SSEL_N(0xe) | 0xdd;
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdd) abort();
while(~LPC_SPI->STAT & SPI_STAT_MSTIDLE);
```

44.3.6 Transmit and receive 24 bits to/from slave 0

Table 652. SPI Code example

Transmit and receive 24 bits to/from slave 0.

```
LPC_SPI->CFG = SPI_CFG_MASTER | SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(15) | SPI_TXDATCTL_SSEL_N(0xe) | 0xdddd;
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdddd) abort();
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_EOT |
 SPI_TXDATCTL_SSEL_N(0xe) | 0xdd;
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdd) abort();
while(~LPC_SPI->STAT & SPI_STAT_MSTIDLE);
```

44.3.7 Transmit and receive 24 bits to/from slave 0, interrupt mode

Table 653. SPI Code example

Transmit and receive 24 bits to/from slave 0, interrupt mode.

```
LPC_SPI->CFG = SPI_CFG_MASTER | SPI_CFG_ENABLE;
LPC_SPI->INTENSET = SPI_STAT_TXRDY | SPI_STAT_RXRDY;
while(LPC_SPI->INTENSET & (SPI_STAT_TXRDY | SPI_STAT_RXRDY));
NVIC_DisableIRQ(Spi_IRQn);
```

44.3.8 Transmit 8 bits to master

Table 654. SPI Code example

Transmit 8 bits to master.

```
LPC_SPI->CFG = SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | SPI_TXDATCTL_RXIGNORE | 0xdd;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->STAT = SPI_STAT_SSD;
while(~LPC_SPI->STAT & SPI_STAT_SSD);
LPC_SPI->STAT = SPI_STAT_SSD;
```

44.3.9 Receive 8 bits to master

Table 655. SPI Code example

Receive 8 bits to master.

```
LPC_SPI->CFG = SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7);
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdd) abort();
```

44.3.10 Transmit and receive 24 bits to master

Table 656. SPI Code example

Transmit and receive 24 bits to master.

```
LPC_SPI->CFG = SPI_CFG_ENABLE;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(15) | 0xdddd;
while(~LPC_SPI->STAT & SPI_STAT_TXRDY);
LPC_SPI->TXDATCTL = SPI_TXDATCTL_FLEN(7) | 0xdd;
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdddd) abort();
while(~LPC_SPI->STAT & SPI_STAT_RXRDY);
data = LPC_SPI->RXDAT;
if(data != 0xdd) abort();
```

44.4 Code examples UART

44.4.1 Definitions

Table 657. UART Code example

UART defines

```
#define UART_CFG_ENABLE (0x1 << 0)
#define UART_CFG_DATALEN(d) (((d) - 7) << 2)
#define UART_STAT_RXRDY (0x1 << 0)
#define UART_STAT_TXRDY (0x1 << 2)
#define UART_STAT_TXIDLE (0x1 << 3)
```

44.4.2 Interrupt handler

Table 658. UART Code example

Interrupt handler

```
void Usart_IRQHandler() {
      uint32_t intstat = LPC_USART->INTSTAT;
      if(intstat & UART_STAT_RXRDY) {
            if(!tx_rdy_flag) abort();
            tx_rdy_flag = 0;
            LPC_USART->TXDAT = LPC_USART->RXDAT;
            LPC_USART->INTENSET = UART_STAT_TXRDY;
            tx_counter++;
      }
      if(intstat & UART_STAT_TXRDY) {
            if(tx_rdy_flag) abort();
            tx_rdy_flag = 1;
            LPC_USART->INTENCLR = UART_STAT_TXRDY;
      }
}
```

44.4.3 Transmit one byte of data

Table 659. UART Code example

Transmit one byte of data.

```
LPC_USART->CFG = UART_CFG_DATALEN(8) | UART_CFG_ENABLE;
while(~LPC_USART->STAT & UART_STAT_TXRDY);
LPC_USART->TXDAT = 0xdd;
while(~LPC_USART->STAT & UART_STAT_TXIDLE);
```

44.4.4 Receive one byte of data

Table 660. UART Code example

Receive one byte of data.

```
LPC_USART->CFG = UART_CFG_DATALEN(8) | UART_CFG_ENABLE;
while(~LPC_USART->STAT & UART_STAT_RXRDY);
data = LPC_USART->RXDAT;
if(data != 0xdd) abort();
```

44.4.5 Transmit and receive one byte of data

Table 661. UART Code example

Transmit and receive one byte of data.

```
LPC_USART->CFG = UART_CFG_DATALEN(8) | UART_CFG_ENABLE;
while(~LPC_USART->STAT & UART_STAT_TXRDY);
LPC_USART->TXDAT = 0xdd;
while(~LPC_USART->STAT & UART_STAT_RXRDY);
data = LPC_USART->RXDAT;
if(data != 0xdd) abort();
```

44.4.6 Loop back 10 bytes of data

Table 662. UART Code example

Loop back 10 bytes of data.

```
LPC_USART->CFG = UART_CFG_DATALEN(8) | UART_CFG_ENABLE;
for(i = 0; i < 10; i++) {
     while(~LPC_USART->STAT & (UART_STAT_TXRDY | UART_STAT_RXRDY));
     LPC_USART->TXDAT = LPC_USART->RXDAT;
}
while(~LPC_USART->STAT & UART_STAT_TXIDLE);
```

44.4.7 Loop back 10 bytes of data using interrupts

Table 663. UART Code example

Loop back 10 bytes of data using interrupts.

```
LPC_USART->CFG = UART_CFG_DATALEN(8) | UART_CFG_ENABLE;
LPC_USART->INTENSET = UART_STAT_TXRDY | UART_STAT_RXRDY;
while(tx_counter < 10);
LPC_USART->INTENCLR = UART_STAT_TXRDY | UART_STAT_RXRDY;
while(~LPC_USART->STAT & UART_STAT_TXIDLE);
NVIC_DisableIRQ(Usart_IRQn);
```

44.5 Code examples DMA

44.5.1 Definitions

Table 664. DMA Code example

DMA defines

```
#define DMA_CTL_ENABLE (0x1)
#define DMA_CFG_PERIPHREQEN (0x1 << 0)
#define DMA_CFG_HWTRIGEN (0x1 << 1)
#define DMA_CFG_BURSTPOWER(p) ((p) << 8)
#define DMA_CFG_SRCBURSTWRAP (0x1 << 14)
#define DMA_CFG_DSTBURSTWRAP (0x1 << 15)
#define DMA_CTLSTAT_TRIG (0x1 << 2)
#define DMA_XFERCFG_CFGVALID (0x1 << 0)
#define DMA_XFERCFG_RELOAD (0x1 << 1)
#define DMA_XFERCFG_SWTRIG (0x1 << 2)
#define DMA_XFERCFG_CLRTRIG (0x1 << 3)
#define DMA_XFERCFG_SETINTA (0x1 << 4)
#define DMA_XFERCFG_SETINTB (0x1 << 5)
#define DMA_XFERCFG_BYTE (0x0 << 8)
#define DMA_XFERCFG_SHORT (0x1 << 8)
#define DMA_XFERCFG_WORD (0x2 << 8)
#define DMA_XFERCFG_SRCNOINC (0x0 << 12)
#define DMA_XFERCFG_SRCINC (0x1 << 12)
#define DMA_XFERCFG_DSTNOINC (0x0 << 14)
#define DMA_XFERCFG_DSTINC (0x1 << 14)
#define DMA_XFERCFG_XFERCOUNT(c) ((c) << 16)
```

44.5.2 Transmit one byte of data

Table 665. DMA Code example

DMA 16 bytes from source to destination

```
//DMA 16 bytes from source to destination.
volatile uint8_t src[16];
volatile uint8_t dst[16];
uint32_t i;
for(i = 0; i < 16; i++)
     src[i] = 1 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[15];
dma_sram[2] = (uint32_t)&dst[15];
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
 DMA_XFERCFG_SRCINC | DMA_XFERCFG_BYTE | DMA_XFERCFG_SWTRIG | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < 16; i++)
     if(dst[i] != src[i])
          abort();
```

44.5.3 DMA 4 bytes of source into 16 bytes of destination

Table 666. DMA Code example

DMA 4 bytes of source into 16 bytes of destination. 0x1 0x2 0x3 0x4 -> 0x1 0x2 0x3 0x4 0x1 0x2 0x3 0x4 0x1 0x2 0x3 0x4 0x1 0x2 0x3 0x4.

```
for(i = 0; i < 4; i++)
     src[i] = 2 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[3];
dma_sram[2] = (uint32_t)&dst[15];
LPC_DMA_CHANNEL->CFG = DMA_CFG_SRCBURSTWRAP | DMA_CFG_BURSTPOWER(2);
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
 DMA_XFERCFG_SRCINC | DMA_XFERCFG_BYTE | DMA_XFERCFG_SWTRIG | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < 16; i++)
     if(dst[i] != src[i % 4])
          abort();
LPC_DMA_CHANNEL->CFG = 0;
```

44.5.4 DMA one word of source into four words of destination

Table 667. DMA Code example

DMA 1 word of source into 4 words of destination. 0x04030201 -> 0x04030201 0x04030201 0x04030201 0x04030201

```
for(i = 0; i < 1; i++)
     src[i] = 3 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[0];
dma_sram[2] = (uint32_t)&dst[3];
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
DMA_XFERCFG_WORD | DMA_XFERCFG_SWTRIG | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < 4; i++)
     if(dst[i] != src[0])
          abort();
```

44.5.5 DMA 16 bytes of source to destination with inta and valid mechanism

Table 668. DMA Code example

DMA 16 bytes of source to destination. Erase source. DMA 16 bytes of destination back to source. With inta and valid mechanism.

```
for(i = 0; i < 16; i++)
     src[i] = 4 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[15];
dma_sram[2] = (uint32_t)&dst[15];
dma_sram[3] = (uint32_t)reload;
reload[0] = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC | DMA_XFERCFG_SRCINC
 | DMA_XFERCFG_BYTE; // don't set config valid
reload[1] = (uint32_t)&dst[15];
reload[2] = (uint32_t)&src[15];
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
 DMA_XFERCFG_SRCINC | DMA_XFERCFG_BYTE | DMA_XFERCFG_SETINTA | DMA_XFERCFG_SWTRIG |
 DMA_XFERCFG_RELOAD | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(~LPC_DMA_COMMON->INTA0 & (1 << 0));
LPC_DMA_COMMON->INTA0 = (1 << 0);
for(i = 0; i < 16; i++) {
     if(dst[i] != src[i])
          abort();
     src[i] = 0;
}
LPC_DMA_COMMON->SETVALID0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < 16; i++)
     if(src[i] != dst[i])
          abort();
```

44.5.6 DMA 16 bytes of source to destination with inta and trigger mechanism

Table 669. DMA Code example

DMA 16 bytes of source to destination. Erase source. DMA 16 bytes of destination data back to source. With inta and trigger mechanism.

```
for(i = 0; i < 16; i++)
     src[i] = 5 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[15];
dma_sram[2] = (uint32_t)&dst[15];
dma_sram[3] = (uint32_t)reload;
reload[0] = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC | DMA_XFERCFG_SRCINC
| DMA_XFERCFG_BYTE | DMA_XFERCFG_CFGVALID;
reload[1] = (uint32_t)&dst[15];
reload[2] = (uint32_t)&src[15];
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
DMA_XFERCFG_SRCINC | DMA_XFERCFG_BYTE | DMA_XFERCFG_SETINTA | DMA_XFERCFG_CLRTRIG |
DMA_XFERCFG_SWTRIG | DMA_XFERCFG_RELOAD | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(~LPC_DMA_COMMON->INTA0 & (1 << 0));
LPC_DMA_COMMON->INTA0 = (1 << 0);
for(i = 0; i < 16; i++) {
     if(dst[i] != src[i])
          abort();
     src[i] = 0;
}
LPC_DMA_COMMON->SETTRIG0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < 16; i++)
     if(src[i] != dst[i])
          abort();
```

44.5.7 DMA 16 bytes of source to destination. Modify source.

Table 670. DMA Code example

DMA 16 bytes of source to destination. Stop in the middle. Modify source. Continue DMA.

```
for(i = 0; i < 16; i++)
     src[i] = 6 + i;
LPC_DMA_COMMON->SRAMBASE = (uint32_t)dma_sram;
LPC_DMA_COMMON->CTL = DMA_CTL_ENABLE;
dma_sram[1] = (uint32_t)&src[15];
dma_sram[2] = (uint32_t)&dst[15];
LPC_DMA_CHANNEL->XFERCFG = DMA_XFERCFG_XFERCOUNT(15) | DMA_XFERCFG_DSTINC |
 DMA_XFERCFG_SRCINC | DMA_XFERCFG_BYTE | DMA_XFERCFG_SWTRIG | DMA_XFERCFG_CFGVALID;
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(~LPC_DMA_COMMON->BUSY0 & (1 << 0)); // wait for it to start
LPC_DMA_COMMON->ENABLECLR0 = (1 << 0); // disable channel
while(LPC_DMA_COMMON->BUSY0 & (1 << 0)); // wait for it to idle
count = 15 - (LPC_DMA_CHANNEL->XFERCFG >> 16);
for(i = 0; i < 16; i++)
     src[i] = ~src[i];
LPC_DMA_COMMON->ENABLESET0 = (1 << 0);
while(LPC_DMA_COMMON->ACTIVE0 & (1 << 0));
for(i = 0; i < count; i++)
     if(src[i] != (uint8_t)~dst[i])
          abort();
for(; i < 16; i++)
     if(src[i] != dst[i])
          abort();
```

45.1 Abbreviations

Table 671. Abbreviations

Acronym	Description
A/D	Analog-to-Digital
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
BOD	BrownOut Detection
GPIO	General Purpose Input/Output
JTAG	Joint Test Action Group
PLL	Phase-Locked Loop
RC	Resistor-Capacitor
SPI	Serial Peripheral Interface
SSI	Serial Synchronous Interface
SSP	Synchronous Serial Port
TAP	Test Access Port
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous Asynchronous Receiver/Transmitter

45.2 References

[1] **DDI 0337E —** ARM Cortex-M3 technical reference manual

[2] LPC15xx data sheet:
http://www.nxp.com/documents/data_sheet/LPC15XX.pdf

[3] LPC15xx Errata sheet:
http://www.nxp.com/documents/errata_sheet/ES_LPC15XX.pdf

45.3 Legal information

45.3.1 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.

45.3.2 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.

45.3.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I²C-bus — logo is a trademark of NXP B.V.

45.4 Tables

45.5 Figures

45.6 Contents

Chapter 1: LPC15xx Introductory information

Chapter 1: LPC15xx Memory mapping

Chapter 2: LPC15xx Nested Vectored Interrupt Controller (NVIC)

Chapter 3: LPC15xx System configuration (SYSCON)

Chapter 4: LPC15xx Power Management Unit (PMU)

Chapter 5: LPC15xx Boot process

Chapter 6: LPC15xx Pin description

Chapter 7: LPC15xx I/O pin configuration (IOCON)

Chapter 8: LPC15xx Switch Matrix (SWM)

Chapter 9: LPC15xx Input multiplexing (INPUT MUX)

Chapter 10: LPC15xx General Purpose I/O (GPIO)

Chapter 11: LPC15xx Group GPIO input interrupt (GINT0/1)

Chapter 12: LPC15xx Pin interrupt and pattern match (PINT)

Chapter 13: LPC15xx DMA controller

Chapter 14: LPC15xx SCT Input Processing Unit (SCTIPU)

Chapter 15: Large SCTimer/PWM (SCTimer0/PWM, SCTimer1/PWM)

Chapter 16: Small SCTimer/PWM (SCTimer2/PWM, SCTimer3/PWM)

Chapter 17: LPC15xx Windowed Watchdog Timer (WWDT)

Chapter 18: LPC15xx Real-Time Clock (RTC)

Chapter 19: LPC15xx Multi-Rate Timer (MRT)

Chapter 20: LPC15xx Repetitive Interrupt Timer (RIT)

Chapter 21: LPC15xx ARM Cortex-M3 Systick timer

Chapter 22: LPC15xx Quadrature Encoder Interface (QEI)

Chapter 23: LPC15xx USB

Chapter 24: LPC15xx USART0/1/2

Chapter 25: LPC15xx SPI0/1

Chapter 26: LPC15xx I2C-bus interface

Chapter 27: LPC15xx Controller Area Network C_CAN0

Chapter 28: LPC15xx 12-bit ADC controller ADC0/1

Chapter 29: LPC15xx 12-bit DAC

Chapter 30: LPC15xx Analog comparators 0/1/2/3

Chapter 31: LPC15xx Temperature sensor

Chapter 32: LPC15xx CRC engine

Chapter 33: LPC15xx Flash controller

Chapter 34: LPC15xx Flash/EEPROM API

Chapter 35: LPC15xx Power profiles/Power control API

Chapter 36: LPC15xx USART API ROM driver routines

Chapter 37: LPC15xx DMA API ROM driver routines

Chapter 38: LPC15xx I2C API

Chapter 39: LPC15xx ADC API

Chapter 40: LPC15xx SPI API ROM driver routines

Chapter 41: LPC15xx C_CAN API ROM driver routines

Chapter 42: LPC15xx USB ROM API

Chapter 43: LPC15xx Serial Wire Debug (SWD)

Chapter 44: LPC15xx Code examples

Chapter 45: Supplementary information

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 3 March 2014
Document identifier: UM10736