# UM10398

## LPC111x/LPC11Cxx User manual

**Rev. 4 — 4 March 2011** **User manual**

### Document information

| Info | Content |
|---|---|
| **Keywords** | ARM Cortex-M0, LPC1111, LPC1112, LPC1113, LPC1114, LPC11C12, LPC11C14, LPC1100, LPC1100L, LPC11C00, LPC11C22, LPC11C24 |
| **Abstract** | LPC111x/LPC11Cxx User manual |

### Revision history

| Rev | Date | Description |
|---|---|---|
| 4 | 20110304 | |
| Modifications: | • LPC111FHN33/102 and LPC11FHN33/201 device IDs updated in Table 42 and Table 280.<br>• Section 3.7 “Start-up behavior” added.<br>• Figure 6 “Power profiles pointer structure” updated. | |
| 3 | 20110114 | LPC111x/LPC11C1x/LPC11C2x User manual |
| Modifications: | • Parts LPC11C22 and LPC11C24 added.<br>• Description of on-chip CAN transceiver added.<br>• Description of system tick timer updated. Reference clock added as clock source (Chapter 18) and Table 351.<br>• Editorial updates to Chapter 12.<br>• Single-cycle hardware multiply specified in Table 355. | |
| 2 | 20101102 | LPC111x/LPC11C1x User manual |
| Modifications: | • Parts LPC111x/102/202/302 (LPC1100L series) added (Table 1, Table 280).<br>• PLL output frequency updated (< 100 MHz) in Section 3.10 “System PLL functional description”.<br>• Description of Deep-sleep and Deep-power down modes updated in Section 3.8 “Power management”.<br>• Chapter 5 “LPC111x/LPC11Cxx Power profiles” added.<br>• WDT changed to 24-bit timer in Chapter 17 “LPC111x/LPC11Cxx WatchDog Timer (WDT)”.<br>• Connection to standard debug connector explained in Section 21.6.2 “Debug connections”.<br>• Register SYSTCKCAL moved to location 0x4004 8154 (Table 6 and Table 33).<br>• Flash signature generation added (Section 20.10 “Flash signature generation”).<br>• Baudrates for ISP restricted (230400 bps removed) in Section 20.5 “UART ISP commands”.<br>• Flash write disturbance effect described in Section 20.5.7 “Copy RAM to flash <Flash address> <RAM address> <no of bytes> (UART ISP)”.<br>• Sections clocking and power control removed and content combined with section basic configuration for each chapter. | |
| 1 | 20100721 | LPC111x/LPC11C1x User manual |

## Contact information

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

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

## 1.1 Introduction

The LPC111x/LPC11Cxx are a ARM Cortex-M0 based, low-cost 32-bit MCU family, designed for 8/16-bit microcontroller applications, offering performance, low power, simple instruction set and memory addressing together with reduced code size compared to existing 8/16-bit architectures.

The LPC111x/LPC11Cxx operate at CPU frequencies of up to 50 MHz.

The peripheral complement of the LPC111x/LPC11Cxx includes up to 32 kB of flash memory, up to 8 kB of data memory, one C_CAN controller (LPC11Cxx), one Fast-mode Plus $I^2C$-bus interface, one RS-485/EIA-485 UART, up to two SPI interfaces with SSP features, four general purpose timers, a 10-bit ADC, and up to 42 general purpose I/O pins.

On-chip C_CAN drivers and flash In-System Programming tools via C_CAN are included on the LPC11Cxx. In addition, parts LPC11C2x are equipped with an on-chip CAN transceiver.

**Remark:** This user manual covers the LPC111x/LPC11Cxx series. The series consists of the LPC1100 series (parts LPC111x/101/201/301), the LPC1100L series (parts LPC111x/102/202/302), and the LPC11C00 series (parts LPC11C1x/301 and LPC11C2x/301). The LPC1100L include the power profiles and the LPC11C00 include the C_CAN controller and on-chip CAN drivers.

## 1.2 Features

- System:
  - ARM Cortex-M0 processor, running at frequencies of up to 50 MHz.
  - ARM Cortex-M0 built-in Nested Vectored Interrupt Controller (NVIC).
  - Serial Wire Debug.
  - System tick timer.
- Memory:
  - 32 kB (LPC1114/LPC11C14), 24 kB (LPC1113), 16 kB (LPC1112/LPC11C12), or 8 kB (LPC1111) on-chip flash programming memory.
  - 8 kB, 4 kB, or 2 kB SRAM.
  - In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software.
- Digital peripherals:
  - Up to 42 General Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors. Number of GPIO pins is reduced for smaller packages and LPC11C22/C24.
  - GPIO pins can be used as edge and level sensitive interrupt sources.
  - High-current output driver (20 mA) on one pin.

  - High-current sink drivers (20 mA) on two $I^2C$-bus pins in Fast-mode Plus.
  - Four general purpose timers/counters with a total of four capture inputs and up to 13 match outputs.
  - Programmable WatchDog Timer (WDT).
- Analog peripherals:
  - 10-bit ADC with input multiplexing among 8 pins.
- Serial interfaces:
  - UART with fractional baud rate generation, internal FIFO, and RS-485 support.
  - Two SPI controllers with SSP features and with FIFO and multi-protocol capabilities (second SPI on LQFP48 and PLCC44 packages only).
  - $I^2C$-bus interface supporting full $I^2C$-bus specification and Fast-mode Plus with a data rate of 1 Mbit/s with multiple address recognition and monitor mode.
  - C_CAN controller (LPC11Cxx only). On-chip CAN and CANopen drivers included.
  - On-chip, high-speed CAN transceiver (parts LPC11C22/C24 only).
- Clock generation:
  - 12 MHz internal RC oscillator trimmed to 1% accuracy that can optionally be used as a system clock.
  - Crystal oscillator with an operating range of 1 MHz to 25 MHz.
  - Programmable watchdog oscillator with a frequency range of 7.8 kHz to 1.8 MHz.
  - 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.
  - Clock output function with divider that can reflect the system oscillator clock, IRC clock, CPU clock, and the Watchdog clock.
- Power control:
  - Integrated PMU (Power Management Unit) to minimize power consumption during Sleep, Deep-sleep, and Deep power-down modes.
  - Power profiles residing in boot ROM allowing to optimize performance and minimize power consumption for any given application through one simple function call. (On LPC111x/102/202/302 only.)
  - Three reduced power modes: Sleep, Deep-sleep, and Deep power-down.
  - Processor wake-up from Deep-sleep mode via a dedicated start logic using up to 13 of the functional pins.
  - Power-On Reset (POR).
  - Brownout detect with four separate thresholds for interrupt and forced reset.
- Unique device serial number for identification.
- Single 3.3 V power supply (1.8 V to 3.6 V).
- Available as 48-pin LQFP package, 33-pin HVQFN package, and 44-pin PLCC package.

## 1.3 Ordering information

**Table 1. Ordering information**

| Type number | Package | | |
|---|---|---|---|
| | Name | Description | Version |
| LPC1111FHN33/101 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1111FHN33/102 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1111FHN33/201 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1111FHN33/202 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1112FHN33/101 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1112FHN33/102 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1112FHN33/201 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1112FHN33/202 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1113FHN33/201 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1113FHN33/202 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1113FHN33/301 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1113FHN33/302 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1114FHN33/201 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1114FHN33/202 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1114FHN33/301 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1114FHN33/302 | HVQFN33 | HVQFN: plastic thermal enhanced very thin quad flat package; no leads; 33 terminals; body 7 × 7 × 0.85 mm | n/a |
| LPC1113FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC1113FBD48/302 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC1114FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC1114FBD48/302 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC1114FA44/301 | PLCC44 | PLCC44; plastic leaded chip carrier; 44 leads | sot187-2 |
| LPC1114FA44/302 | PLCC44 | PLCC44; plastic leaded chip carrier; 44 leads | sot187-2 |

**Table 1. Ordering information** *…continued*

| Type number | Package | | |
|---|---|---|---|
| | Name | Description | Version |
| LPC11C12FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC11C14FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC11C22FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |
| LPC11C24FBD48/301 | LQFP48 | LQFP48: plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm | sot313-2 |

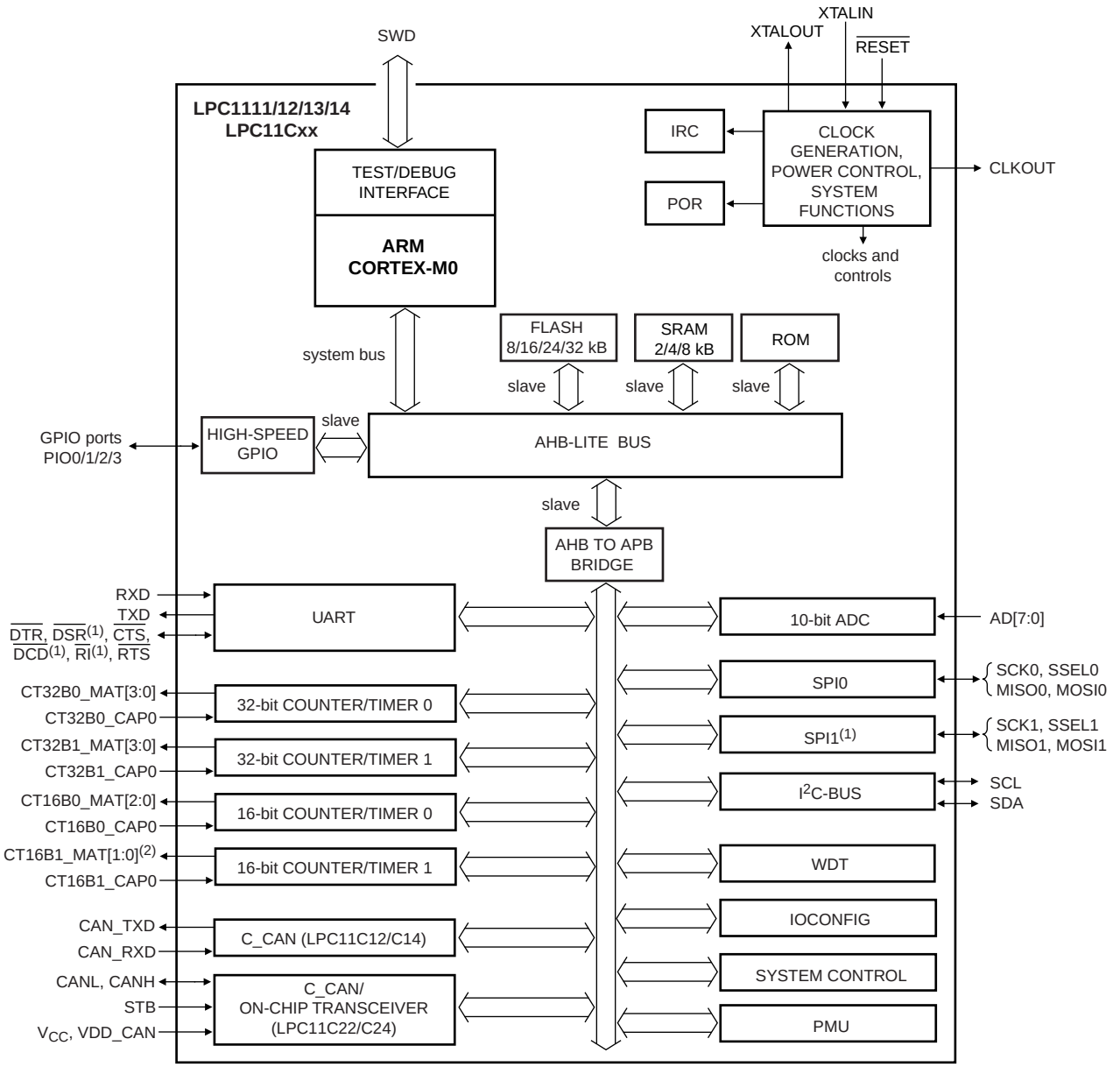
**Table 2. Ordering options**

| Type number | Series | Flash | Total SRAM | UART RS-485 | $I^2C$/ Fm+ | SPI | C_CAN | Power profiles | ADC channels | Package |
|---|---|---|---|---|---|---|---|---|---|---|
| **LPC1111** | | | | | | | | | | |
| LPC1111FHN33/101 | LPC1100 | 8 kB | 2 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1111FHN33/102 | LPC1100L | 8 kB | 2 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1111FHN33/201 | LPC1100 | 8 kB | 4 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1111FHN33/202 | LPC1100L | 8 kB | 4 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| **LPC1112** | | | | | | | | | | |
| LPC1112FHN33/101 | LPC1100 | 16 kB | 2 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1112FHN33/102 | LPC1100L | 16 kB | 2 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1112FHN33/201 | LPC1100 | 16 kB | 4 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1112FHN33/202 | LPC1100L | 16 kB | 4 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| **LPC1113** | | | | | | | | | | |
| LPC1113FHN33/201 | LPC1100 | 24 kB | 4 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1113FHN33/202 | LPC1100L | 24 kB | 4 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1113FHN33/301 | LPC1100 | 24 kB | 8 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1113FHN33/302 | LPC1100L | 24 kB | 8 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1113FBD48/301 | LPC1100 | 24 kB | 8 kB | 1 | 1 | 2 | - | no | 8 | LQFP48 |
| LPC1113FBD48/302 | LPC1100L | 24 kB | 8 kB | 1 | 1 | 2 | - | yes | 8 | LQFP48 |
| **LPC1114** | | | | | | | | | | |
| LPC1114FHN33/201 | LPC1100 | 32 kB | 4 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1114FHN33/202 | LPC1100L | 32 kB | 4 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1114FHN33/301 | LPC1100 | 32 kB | 8 kB | 1 | 1 | 1 | - | no | 8 | HVQFN33 |
| LPC1114FHN33/302 | LPC1100L | 32 kB | 8 kB | 1 | 1 | 1 | - | yes | 8 | HVQFN33 |
| LPC1114FBD48/301 | LPC1100 | 32 kB | 8 kB | 1 | 1 | 2 | - | no | 8 | LQFP48 |
| LPC1114FBD48/302 | LPC1100L | 32 kB | 8 kB | 1 | 1 | 2 | - | yes | 8 | LQFP48 |
| LPC1114FA44/301 | LPC1100 | 32 kB | 8 kB | 1 | 1 | 2 | - | no | 8 | PLCC44 |
| LPC1114FA44/302 | LPC1100L | 32 kB | 8 kB | 1 | 1 | 2 | - | yes | 8 | PLCC44 |
| **LPC11C12/LPC11C14** | | | | | | | | | | |
| LPC11C12FBD48/301 | LPC11C00 | 16 kB | 8 kB | 1 | 1 | 2 | 1 | no | 8 | LQFP48 |
| LPC11C14FBD48/301 | LPC11C00 | 32 kB | 8 kB | 1 | 1 | 2 | 1 | no | 8 | LQFP48 |

**Table 2. Ordering options** *…continued*

| Type number | Series | Flash | Total SRAM | UART RS-485 | $I^2C$/ Fm+ | SPI | C_CAN | Power profiles | ADC channels | Package |
|---|---|---|---|---|---|---|---|---|---|---|
| **LPC11C22/LPC11C24 with on-chip, high-speed CAN transceiver** | | | | | | | | | | |
| LPC11C22FBD48/301 | LPC11C00 | 16 kB | 8 kB | 1 | 1 | 2 | 1 | no | 8 | LQFP48 |
| LPC11C24FBD48/301 | LPC11C00 | 32 kB | 8 kB | 1 | 1 | 2 | 1 | no | 8 | LQFP48 |

## 1.4 Block diagram




(1) LQFP48/PLCC44 packages only.

(2) Not available on LPC11C22/C24.

**Fig 1. LPC111x/LPC11Cxx block diagram**

## 1.5 ARM Cortex-M0 processor

The ARM Cortex-M0 processor is described in detail in Section 22.2 “About the Cortex-M0 processor and core peripherals”. For the LPC111x/LPC11Cxx, the ARM Cortex-M0 processor core is configured as follows:

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

## 2.1 How to read this chapter

Table 3 and Table 4 show the memory configurations for different LPC111x/LPC11Cxx parts.

**Table 3. LPC111x memory configuration**

| Part | Flash | SRAM | | |
|---|---|---|---|---|
| Suffix | | /101; /102 | /201; /202 | /301; /302 |
| LPC1111 | 8 kB | 2 kB | 4 KB | - |
| LPC1112 | 16 kB | 2 kB | 4 KB | - |
| LPC1113 | 24 kB | - | 4 KB | 8 kB |
| LPC1114 | 32 kB | - | 4 KB | 8 kB |

**Table 4. LPC11Cxx memory configuration**

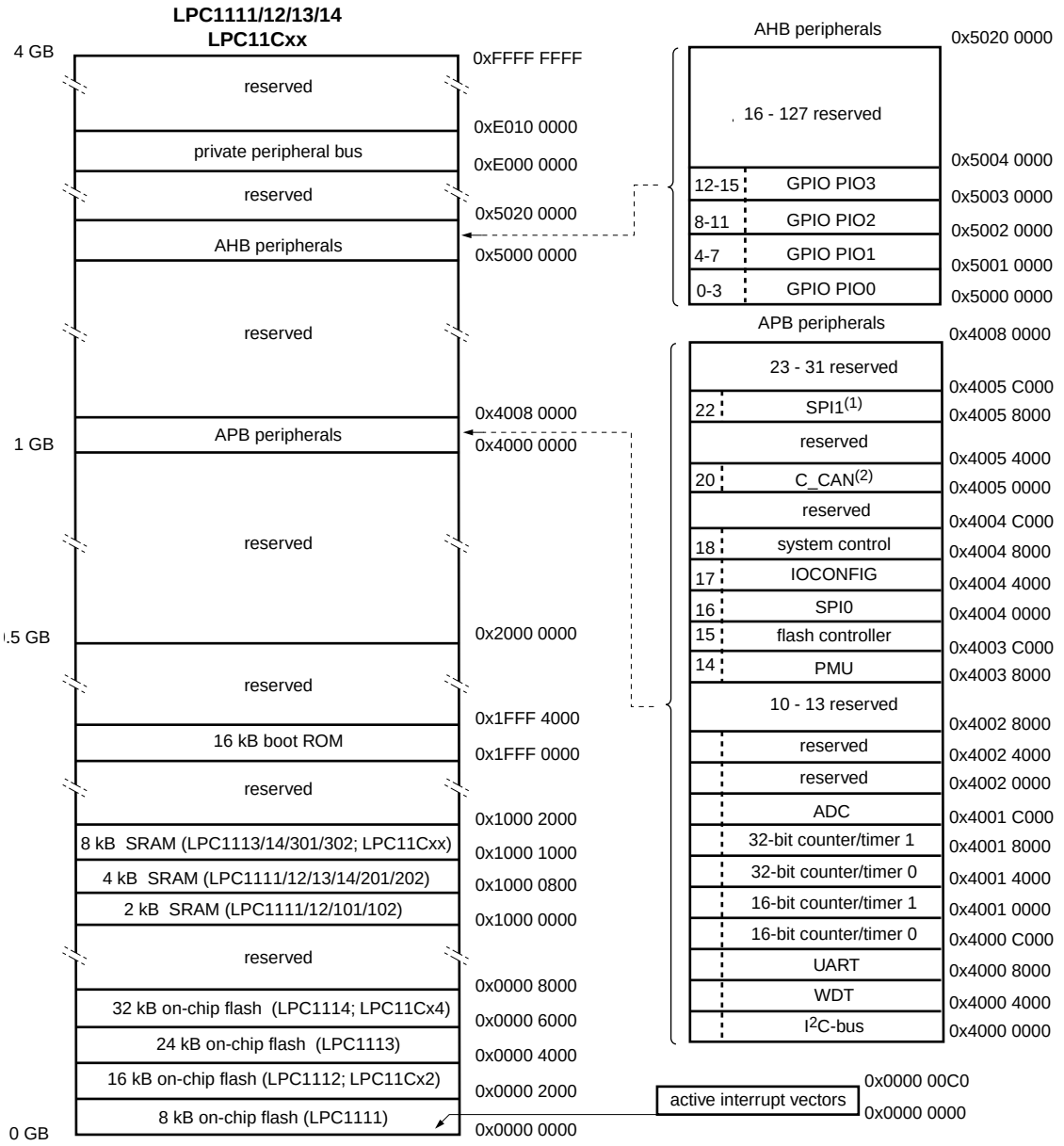
| Part | Flash | SRAM |
|---|---|---|
| LPC11C12/301 | 16 kB | 8 kB |
| LPC11C14/301 | 32 kB | 8 kB |
| LPC11C22/301 | 16 kB | 8 kB |
| LPC11C24/301 | 32 kB | 8 kB |

## 2.2 Memory map

Figure 2 shows the memory and peripheral address space of the LPC111x/LPC11Cxx.

The AHB peripheral area is 2 MB in size and is divided to allow for up to 128 peripherals. On the LPC111x/LPC11Cxx, the GPIO ports are the only AHB peripherals. The APB peripheral area is 512 kB in size and is divided to allow for up to 32 peripherals. Each peripheral of either type is allocated 16 kB of space. This allows simplifying the address decoding for each peripheral.

All peripheral register addresses are 32-bit word aligned regardless of their size. An implication of this is that word and half-word registers must be accessed all at once. For example, it is not possible to read or write the upper byte of a word register separately.




(1) LQFP48/PLCC44 packages only.

(2) LPC11Cxxonly.

**Fig 2. LPC111x/LPC11Cxx memory map**

## 3.1 How to read this chapter

The C_CAN controller is available on parts LPC11Cxx only, and the corresponding clock and reset control bits (bit 17 in the SYSAHBCLKCTRL register and bit 3 in PRESETCTRL register) are reserved for all LPC1111x parts.

**Remark:** For parts LPC111x/102/202/302, also refer to Chapter 5 for setting up power profiles.

## 3.2 Introduction

The system configuration block controls oscillators, start logic, and clock generation of the LPC111x/LPC11Cxx. Also included in this block are registers for setting the priority for AHB access and a register for remapping flash, SRAM, and ROM memory areas.

## 3.3 Pin description

Table 5 shows pins that are associated with system control block functions.

**Table 5. Pin summary**

| Pin name | Pin direction | Pin description |
|---|---|---|
| CLKOUT | O | Clockout pin |
| PIO0_0 to PIO0_11 | I | Start logic wake-up pins port 0 |
| PIO1_0 | I | Start logic wake-up pin port 1 |

## 3.4 Clock generation

See Figure 3 for an overview of the LPC111x/LPC11Cxx Clock Generation Unit (CGU).

The LPC111x/LPC11Cxx include three independent oscillators. These are the system oscillator, the Internal RC oscillator (IRC), and the watchdog oscillator. Each oscillator can be used for more than one purpose as required in a particular application.

Following reset, the LPC111x/LPC11Cxx will operate from the Internal RC oscillator until switched by software. This allows systems to operate without any external crystal and the bootloader code to operate at a known frequency.

The SYSAHBCLKCTRL register gates the system clock to the various peripherals and memories. UART, the WDT, and SPI0/1 have individual clock dividers to derive peripheral clocks from the main clock.

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

For details on power control see Section 3.9.

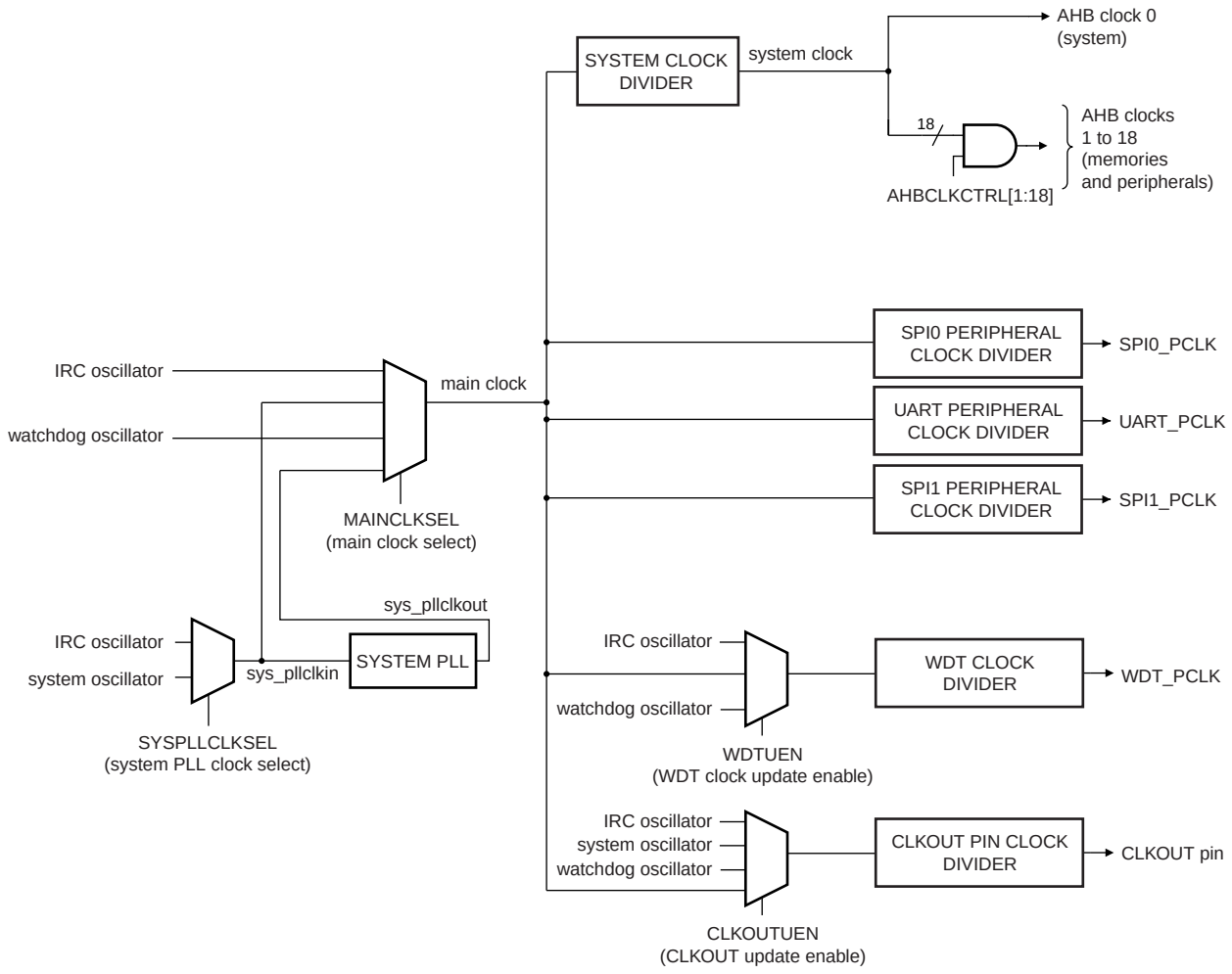



**Fig 3. LPC111x/LPC11Cxx CGU block diagram**

## 3.5 Register description

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

See Section 3.12 for the flash access timing register, which can be re-configured as part the system setup. This register is not part of the system configuration block.

**Table 6. Register overview: system control block (base address 0x4004 8000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SYSMEMREMAP | R/W | 0x000 | System memory remap | 0x002 | Table 7 |
| PRESETCTRL | R/W | 0x004 | Peripheral reset control | 0x000 | Table 8 |
| SYSPLLCTRL | R/W | 0x008 | System PLL control | 0x000 | Table 9 |
| SYSPLLSTAT | R | 0x00C | System PLL status | 0x000 | Table 10 |
| - | - | 0x010 - 0x01C | Reserved | - | - |

**Table 6. Register overview: system control block (base address 0x4004 8000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SYSOSCCTRL | R/W | 0x020 | System oscillator control | 0x000 | Table 11 |
| WDTOSCCTRL | R/W | 0x024 | Watchdog oscillator control | 0x000 | Table 12 |
| IRCCTRL | R/W | 0x028 | IRC control | 0x080 | Table 13 |
| - | - | 0x02C | Reserved | - | - |
| SYSRSTSTAT | R | 0x030 | System reset status register | 0x000 | Table 14 |
| - | - | 0x034 - 0x03C | Reserved | - | - |
| SYSPLLCLKSEL | R/W | 0x040 | System PLL clock source select | 0x000 | Table 15 |
| SYSPLLCLKUEN | R/W | 0x044 | System PLL clock source update enable | 0x000 | Table 16 |
| - | - | 0x048 - 0x06C | Reserved | - | - |
| MAINCLKSEL | R/W | 0x070 | Main clock source select | 0x000 | Table 17 |
| MAINCLKUEN | R/W | 0x074 | Main clock source update enable | 0x000 | Table 18 |
| SYSAHBCLKDIV | R/W | 0x078 | System AHB clock divider | 0x001 | Table 19 |
| - | - | 0x07C | Reserved | - | - |
| SYSAHBCLKCTRL | R/W | 0x080 | System AHB clock control | 0x85F | Table 20 |
| - | - | 0x084 - 0x090 | Reserved | - | - |
| SSP0CLKDIV | R/W | 0x094 | SPI0 clock divder | 0x000 | Table 21 |
| UARTCLKDIV | R/W | 0x098 | UART clock divder | 0x000 | Table 22 |
| SSP1CLKDIV | R/W | 0x09C | SPI1 clock divder | 0x000 | Table 23 |
| - | - | 0x0A0-0x0CC | Reserved | - | - |
| WDTCLKSEL | R/W | 0x0D0 | WDT clock source select | 0x000 | Table 24 |
| WDTCLKUEN | R/W | 0x0D4 | WDT clock source update enable | 0x000 | Table 25 |
| WDTCLKDIV | R/W | 0x0D8 | WDT clock divider | 0x000 | Table 26 |
| - | - | 0x0DC | Reserved | - | - |
| CLKOUTCLKSEL | R/W | 0x0E0 | CLKOUT clock source select | 0x000 | Table 27 |
| CLKOUTUEN | R/W | 0x0E4 | CLKOUT clock source update enable | 0x000 | Table 28 |
| CLKOUTCLKDIV | R/W | 0x0E8 | CLKOUT clock divider | 0x000 | Table 29 |
| - | - | 0x0EC - 0x0FC | Reserved | - | - |
| PIOPORCAP0 | R | 0x100 | POR captured PIO status 0 | user dependent | Table 30 |
| PIOPORCAP1 | R | 0x104 | POR captured PIO status 1 | user dependent | Table 31 |
| - | R | 0x108 - 0x14C | Reserved | - | - |
| BODCTRL | R/W | 0x150 | BOD control | 0x000 | Table 32 |
| SYSTCKCAL | R/W | 0x154 | System tick counter calibration | 0x004 | Table 33 |
| - | - | 0x158 - 0x1FC | Reserved | - | - |
| STARTAPRP0 | R/W | 0x200 | Start logic edge control register 0 | | Table 34 |
| STARTERP0 | R/W | 0x204 | Start logic signal enable register 0 | | Table 35 |
| STARTRSRP0CLR | W | 0x208 | Start logic reset register 0 | n/a | Table 36 |
| STARTSRP0 | R | 0x20C | Start logic status register 0 | n/a | Table 37 |
| - | - | 0x210 - 0x22C | Reserved | - | - |

**Table 6. Register overview: system control block (base address 0x4004 8000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| PDSLEEPCFG | R/W | 0x230 | Power-down states in Deep-sleep mode | 0x0000 0000 | Table 39 |
| PDAWAKECFG | R/W | 0x234 | Power-down states after wake-up from Deep-sleep mode | 0x0000 EDF0 | Table 40 |
| PDRUNCFG | R/W | 0x238 | Power-down configuration register | 0x0000 EDF0 | Table 41 |
| - | - | 0x23C - 0x3F0 | Reserved | - | - |
| DEVICE_ID | R | 0x3F4 | Device ID | part dependent | Table 42 |

### 3.5.1 System memory remap register

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

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

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | MAP | | System memory remap | 10 |
| | | 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 | 0x00 |

### 3.5.2 Peripheral reset control register

This register allows software to reset the SPI and I2C peripherals. Writing a 0 to the SSP0/1_RST_N or I2C_RST_N bits resets the SPI0/1 or I2C peripheral. Writing a 1 de-asserts the reset.

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

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

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | SSP0_RST_N | | SPI0 reset control | 0 |
| | | 0 | Resets the SPI0 peripheral. | |
| | | 1 | SPI0 reset de-asserted. | |
| 1 | I2C_RST_N | | I2C reset control | 0 |
| | | 0 | Resets the I2C peripheral. | |
| | | 1 | I2C reset de-asserted. | |

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

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2 | SSP1_RST_N | | SPI1 reset control | 0 |
| | | 0 | Resets the SPI1 peripheral. | |
| | | 1 | SPI1 reset de-asserted. | |
| 3 | CAN_RST_N | | C_CAN reset control | 0 |
| | | 0 | Resets the C_CAN peripheral. | |
| | | 1 | C_CAN reset de-asserted. | |
| 31:4 | - | - | Reserved | 0x00 |

### 3.5.3 System PLL control register

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

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

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

### 3.5.4 System PLL status register

This register is a Read-only register and supplies the PLL lock status (see Section 3.11.1).

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

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

### 3.5.5 System oscillator control register

This register configures the frequency range for the system oscillator.

**Table 11. System oscillator control register (SYSOSCCTRL, address 0x4004 8020) bit description**

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

### 3.5.6 Watchdog oscillator control register

This register configures the watchdog oscillator. The oscillator consists of an analog and a digital part. The analog part contains the oscillator function and generates an analog clock (Fclkana). With the digital part, the analog output clock (Fclkana) can be divided to the required output clock frequency wdt_osc_clk. The analog output frequency (Fclkana) can be adjusted with the FREQSEL bits between 500 kHz and 3.4 MHz. With the digital part Fclkana will be divided (divider ratios = 2, 4,...,64) to wdt_osc_clk using the DIVSEL bits.

The output clock frequency of the watchdog oscillator can be calculated as wdt_osc_clk = $^{Fclkana}?_{(2 \times (1 + DIVSEL))}$ = 7.8 kHz to 1.7 MHz (nominal values).

**Remark:** Any setting of the FREQSEL bits will yield a Fclkana value within ±40% of the listed frequency value. The watchdog oscillator is the clock source with the lowest power consumption. If accurate timing is required, use the IRC or system oscillator.

**Remark:** The frequency of the watchdog oscillator is undefined after reset. The watchdog oscillator frequency must be programmed by writing to the WDTOSCCTRL register before using the watchdog oscillator.

**Table 12. Watchdog oscillator control register (WDTOSCCTRL, address 0x4004 8024) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:0 | DIVSEL | | Select divider for Fclkana.<br>wdt_osc_clk = Fclkana/ (2 × (1 + DIVSEL))<br>00000: 2 × (1 + DIVSEL) = 2<br>00001: 2 × (1 + DIVSEL) = 4<br>to<br>11111: 2 × (1 + DIVSEL) = 64 | 0 |
| 8:5 | FREQSEL | | Select watchdog oscillator analog output frequency (Fclkana). | 0x00 |
| | | 0x1 | 0.5 MHz | |
| | | 0x2 | 0.8 MHz | |

**Table 12. Watchdog oscillator control register (WDTOSCCTRL, address 0x4004 8024) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| | | 0x3 | 1.1 MHz | |
| | | 0x4 | 1.4 MHz | |
| | | 0x5 | 1.6 MHz | |
| | | 0x6 | 1.8 MHz | |
| | | 0x7 | 2.0 MHz | |
| | | 0x8 | 2.2 MHz | |
| | | 0x9 | 2.4 MHz | |
| | | 0xA | 2.6 MHz | |
| | | 0xB | 2.7 MHz | |
| | | 0xC | 2.9 MHz | |
| | | 0xD | 3.1 MHz | |
| | | 0xE | 3.2 MHz | |
| | | 0xF | 3.4 MHz | |
| 31:9 | - | - | Reserved | 0x00 |

## 3.5.7 Internal resonant crystal 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 13. Internal resonant crystal control register (IRCCTRL, address 0x4004 8028) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | TRIM | Trim value | 0x1000 0000, then flash will reprogram |
| 31:9 | - | Reserved | 0x00 |

### 3.5.8 System reset status register

The SYSRSTSTAT register shows the source of the latest reset event. The bits are cleared by writing a one to any of the bits. The POR event clears all other bits in this register, but if another reset signal - for example EXTRST - remains asserted after the POR signal is negated, then its bit is set to detected.

**Table 14. System reset status register (SYSRSTSTAT, address 0x4004 8030) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | POR | | POR reset status | 0x0 |
| | | 0 | No POR detected. | |
| | | 1 | POR detected. | |
| 1 | EXTRST | | Status of the external $\overline{\text{RESET}}$ pin | 0x0 |
| | | 0 | No $\overline{\text{RESET}}$ event detected. | |
| | | 1 | $\overline{\text{RESET}}$ detected. | |
| 2 | WDT | | Status of the Watchdog reset | 0x0 |
| | | 0 | No WDT reset detected. | |
| | | 1 | WDT reset detected. | |
| 3 | BOD | | Status of the Brown-out detect reset | 0x0 |
| | | 0 | No BOD reset detected. | |
| | | 1 | BOD reset detected. | |
| 4 | SYSRST | | Status of the software system reset | 0x0 |
| | | 0 | No System reset detected. | |
| | | 1 | System reset detected. | |
| 31:5 | - | - | Reserved | 0x0 |

### 3.5.9 System PLL clock source select register

This register selects the clock source for the system PLL. The SYSPLLCLKUEN register (see Section 3.5.10) must be toggled from LOW to HIGH for the update to take effect.

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

**Remark:** When using the C_CAN controller with baudrates above 100 kbit/s, the system oscillator must be selected.

**Table 15. System PLL clock source select register (SYSPLLCLKSEL, address 0x4004 8040) bit description**

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

### 3.5.10 System PLL clock source update enable register

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

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

**Table 16. System PLL clock source update enable register (SYSPLLCLKUEN, address 0x4004 8044) bit description**

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

### 3.5.11 Main clock source select register

This register selects the main system clock which can be either any input to the system PLL, the output from the system PLL (sys_pllclkout), or the watchdog or IRC oscillators directly. The main system clock clocks the core, the peripherals, and the memories.

The MAINCLKUEN register (see Section 3.5.12) must be toggled from LOW to HIGH for the update to take effect.

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

**Remark:** When using the C_CAN controller with baudrates above 100 kbit/s, the system oscillator must be selected.

**Table 17. Main clock source select register (MAINCLKSEL, address 0x4004 8070) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | SEL | | Clock source for main clock | 0x00 |
| | | 0x0 | IRC oscillator | |
| | | 0x1 | Input clock to system PLL | |
| | | 0x2 | WDT oscillator | |
| | | 0x3 | System PLL clock out | |
| 31:2 | - | - | Reserved | 0x00 |

### 3.5.12 Main clock source update enable register

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

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

**Table 18. Main clock source update enable register (MAINCLKUEN, address 0x4004 8074) bit description**

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

## 3.5.13 System AHB clock divider register

This register divides the main clock to provide the system clock to the core, memories, and the peripherals. The system clock can be shut down completely by setting the DIV bits to 0x0.

**Table 19. System AHB clock divider register (SYSAHBCLKDIV, address 0x4004 8078) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DIV | System AHB clock divider values<br>0: System clock disabled.<br>1: Divide by 1.<br>to<br>255: Divide by 255. | 0x01 |
| 31:8 | - | Reserved | 0x00 |

## 3.5.14 System AHB clock control register

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

**Table 20. System AHB clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | SYS | | Enables clock for AHB to APB bridge, to the AHB matrix, to the Cortex-M0 FCLK and HCLK, to the SysCon, and to the PMU. This bit is read only. | 1 |
| | | 0 | Reserved | |
| | | 1 | Enable | |
| 1 | ROM | | Enables clock for ROM. | 1 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 2 | RAM | | Enables clock for RAM. | 1 |
| | | 0 | Disable | |
| | | 1 | Enable | |

**Table 20. System AHB clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 3 | FLASHREG | | Enables clock for flash register interface. | 1 |
| | | 0 | Disabled | |
| | | 1 | Enabled | |
| 4 | FLASHARRAY | | Enables clock for flash array access. | 1 |
| | | 0 | Disabled | |
| | | 1 | Enabled | |
| 5 | I2C | | Enables clock for I2C. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 6 | GPIO | | Enables clock for GPIO. | 1 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 7 | CT16B0 | | Enables clock for 16-bit counter/timer 0. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 8 | CT16B1 | | Enables clock for 16-bit counter/timer 1. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 9 | CT32B0 | | Enables clock for 32-bit counter/timer 0. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 10 | CT32B1 | | Enables clock for 32-bit counter/timer 1. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 11 | SSP0 | | Enables clock for SPI0. | 1 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 12 | UART | | Enables clock for UART. Note that the UART pins must be configured in the IOCON block before the UART clock can be enabled. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 13 | ADC | | Enables clock for ADC. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 14 | - | | Reserved | 0 |
| 15 | WDT | | Enables clock for WDT. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |

Table 20. System AHB clock control register (SYSAHBCLKCTRL, address 0x4004 8080) bit description *...continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 16 | IOCON | | Enables clock for I/O configuration block. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 17 | CAN | | Enables clock for C_CAN | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 18 | SSP1 | | Enables clock for SPI1. | 0 |
| | | 0 | Disable | |
| | | 1 | Enable | |
| 31:19 | - | - | Reserved | 0x00 |

### 3.5.15 SPI0 clock divider register

This register configures the SPI0 peripheral clock SPI0_PCLK. The SPI0_PCLK can be shut down by setting the DIV bits to 0x0.

Table 21. SPI0 clock divider register (SSP0CLKDIV, address 0x4004 8094) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DIV | SPI0_PCLK clock divider values<br>0: Disable SPI0_PCLK.<br>1: Divide by 1.<br>to<br>255: Divide by 255. | 0x00 |
| 31:8 | - | Reserved | 0x00 |

### 3.5.16 UART clock divider register

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

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

Table 22. UART clock divider register (UARTCLKDIV, address 0x4004 8098) bit description

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

### 3.5.17 SPI1 clock divider register

This register configures the SPI1 peripheral clock SPI1_PCLK. The SPI1_PCLK can be shut down by setting the DIV bits to 0x0.

Table 23. SPI1 clock divider register (SSP1CLKDIV, address 0x4004 809C) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DIV | SPI1_PCLK clock divider values<br>0: Disable SPI1_PCLK.<br>1: Divide by 1.<br>to<br>255: Divide by 255. | 0x00 |
| 31:8 | - | Reserved | 0x00 |

### 3.5.18 WDT clock source select register

This register selects the clock source for the watchdog timer. The WDTCLKUEN register (see Section 3.5.19) must be toggled from LOW to HIGH for the update to take effect.

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

Table 24. WDT clock source select register (WDTCLKSEL, address 0x4004 80D0) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | SEL | | WDT clock source | 0x00 |
| | | 0x0 | IRC oscillator | |
| | | 0x1 | Main clock | |
| | | 0x2 | Watchdog oscillator | |
| | | 0x3 | Reserved | |
| 31:2 | - | - | Reserved | 0x00 |

### 3.5.19 WDT clock source update enable register

This register updates the clock source of the watchdog timer with the new input clock after the WDTCLKSEL register has been written to. In order for the update to take effect at the input of the watchdog timer, first write a zero to the WDTCLKUEN register and then write a one to WDTCLKUEN.

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

Table 25. WDT clock source update enable register (WDTCLKUEN, address 0x4004 80D4) bit description

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

### 3.5.20 WDT clock divider register

This register determines the divider values for the watchdog clock wdt_clk.

Table 26. WDT clock divider register (WDTCLKDIV, address 0x4004 80D8) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DIV | WDT clock divider values<br>0: Disable WDT_PCLK.<br>1: Divide by 1.<br>to<br>255: Divide by 255. | 0x00 |
| 31:8 | - | Reserved | 0x00 |

### 3.5.21 CLKOUT clock source select register

This register configures the clkout_clk signal to be output on the CLKOUT pin. All three oscillators and the main clock can be selected for the clkout_clk clock.

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

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

Table 27. CLKOUT clock source select register (CLKOUTCLKSEL, address 0x4004 80E0) bit description

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

### 3.5.22 CLKOUT clock source update enable register

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

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

Table 28. CLKOUT clock source update enable register (CLKOUTUEN, address 0x4004 80E4) bit description

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

### 3.5.23 CLKOUT clock divider register

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

**Table 29. CLKOUT clock divider registers (CLKOUTCLKDIV, address 0x4004 80E8) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7:0 | DIV | | Clock output divider values<br>0: Disable CLKOUT.<br>1: Divide by 1.<br>to<br>255: Divide by 255. | 0x00 |
| 31:8 | - | - | Reserved | 0x00 |

### 3.5.24 POR captured PIO status register 0

The PIOPORCAP0 register captures the state (HIGH or LOW) of the PIO pins of ports 0,1, and 2 (pins PIO2_0 to PIO2_7) at power-on-reset. Each bit represents the reset state of one GPIO pin. This register is a read-only status register.

**Table 30. POR captured PIO status registers 0 (PIOPORCAP0, address 0x4004 8100) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 11:0 | CAPPIO0_n | Raw reset status input PIO0_n: PIO0_11 to PIO0_0 | User implementation dependent |
| 23:12 | CAPPIO1_n | Raw reset status input PIO1_n: PIO1_11 to PIO1_0 | User implementation dependent |
| 31:24 | CAPPIO2_n | Raw reset status input PIO2_n: PIO2_7 to PIO2_0 | User implementation dependent |

### 3.5.25 POR captured PIO status register 1

The PIOPORCAP1 register captures the state (HIGH or LOW) of the PIO pins of port 2 (PIO2_8 to PIO2_11) and port 3 at power-on-reset. Each bit represents the reset state of one PIO pin. This register is a read-only status register.

**Table 31. POR captured PIO status registers 1 (PIOPORCAP1, address 0x4004 8104) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | CAPPIO2_8 | Raw reset status input PIO2_8 | User implementation dependent |
| 1 | CAPPIO2_9 | Raw reset status input PIO2_9 | User implementation dependent |
| 2 | CAPPIO2_10 | Raw reset status input PIO2_10 | User implementation dependent |
| 3 | CAPPIO2_11 | Raw reset status input PIO2_11 | User implementation dependent |
| 4 | CAPPIO3_0 | Raw reset status input PIO3_0 | User implementation dependent |
| 5 | CAPPIO3_1 | Raw reset status input PIO3_1 | User implementation dependent |
| 6 | CAPPIO3_2 | Raw reset status input PIO3_2 | User implementation dependent |
| 7 | CAPPIO3_3 | Raw reset status input PIO3_3 | User implementation dependent |
| 8 | CAPPIO3_4 | Raw reset status input PIO3_4 | User implementation dependent |
| 9 | CAPPIO3_5 | Raw reset status input PIO3_5 | User implementation dependent |
| 31:10 | - | Reserved | - |

### 3.5.26 BOD control register

The BOD control register selects four separate threshold values for sending a BOD interrupt to the NVIC and for forced reset. Reset and interrupt threshold values listed in Table 32 are typical values.

**Table 32. BOD control register (BODCTRL, address 0x4004 8150) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | BODRSTLEV | | BOD reset level | 00 |
| | | 0x0 | Level 0: The reset assertion threshold voltage is 1.46 V; the reset de-assertion threshold voltage is 1.63 V. | |
| | | 0x1 | Level 1: The reset assertion threshold voltage is 2.06 V; the reset de-assertion threshold voltage is 2.15 V. | |
| | | 0x2 | Level 2: The reset assertion threshold voltage is 2.35 V; the reset de-assertion threshold voltage is 2.43 V. | |
| | | 0x3 | Level 3: The reset assertion threshold voltage is 2.63 V; the reset de-assertion threshold voltage is 2.71 V. | |
| 3:2 | BODINTVAL | | BOD interrupt level | 00 |
| | | 0x0 | Level 0: The interrupt assertion threshold voltage is 1.65 V; the interrupt de-assertion threshold voltage is 1.80 V. | |
| | | 0x1 | Level 1:The interrupt assertion threshold voltage is 2.22 V; the interrupt de-assertion threshold voltage is 2.35 V. | |
| | | 0x2 | Level 2: The interrupt assertion threshold voltage is 2.52 V; the interrupt de-assertion threshold voltage is 2.66 V. | |
| | | 0x3 | Level 3: The interrupt assertion threshold voltage is 2.80 V; the interrupt de-assertion threshold voltage is 2.90 V. | |
| 4 | BODRSTENA | | BOD reset enable | 0 |
| | | 0 | Disable reset function. | |
| | | 1 | Enable reset function. | |
| 31:5 | - | - | Reserved | 0x00 |

### 3.5.27 System tick counter calibration register

This register determines the value of the SYST_CALIB register (see Table 255).

**Table 33. System tick timer calibration register (SYSTCKCAL, address 0x4004 8154) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 25:0 | CAL | System tick timer calibration value | 0x04 |
| 31:26 | - | Reserved | 0x00 |

### 3.5.28 Start logic edge control register 0

The STARTAPRP0 register controls the start logic inputs of ports 0 (PIO0_0 to PIO0_11) and 1 (PIO1_0). This register selects a falling or rising edge on the corresponding PIO input to produce a falling or rising clock edge, respectively, for the start logic (see Section 3.10.2).

Every bit in the STARTAPRP0 register controls one port input and is connected to one wake-up interrupt in the NVIC. Bit 0 in the STARTAPRP0 register corresponds to interrupt 0, bit 1 to interrupt 1, etc. (see Table 52), up to a total of 13 interrupts.

**Remark:** Each interrupt connected to a start logic input must be enabled in the NVIC if the corresponding PIO pin is used to wake up the chip from Deep-sleep mode.

**Table 34. Start logic edge control register 0 (STARTAPRP0, address 0x4004 8200) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 11:0 | APRPIO0_n | Edge select for start logic input PIO0_n: PIO0_11 to PIO0_0<br>0 = Falling edge<br>1 = Rising edge | 0x0 |
| 12 | APRPIO1_0 | Edge select for start logic input PIO1_0<br>0 = Falling edge<br>1 = Rising edge | 0x0 |
| 31:13 | - | Reserved | 0x0 |

## 3.5.29 Start logic signal enable register 0

This STARTERP0 register enables or disables the start signal bits in the start logic. The bit assignment is identical to Table 34.

**Table 35. Start logic signal enable register 0 (STARTERP0, address 0x4004 8204) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 11:0 | ERPIO0_n | Enable start signal for start logic input PIO0_n: PIO0_11 to PIO0_0<br>0 = Disabled<br>1 = Enabled | 0x0 |
| 12 | ERPIO1_0 | Enable start signal for start logic input PIO1_0<br>0 = Disabled<br>1 = Enabled | 0x0 |
| 31:13 | - | Reserved | 0x0 |

## 3.5.30 Start logic reset register 0

Writing a one to a bit in the STARTRSRP0CLR register resets the start logic state. The bit assignment is identical to Table 34. The start-up logic uses the input signals to generate a clock edge for registering a start signal. This clock edge (falling or rising) sets the interrupt for waking up from Deep-sleep mode. Therefore, the start-up logic states must be cleared before being used.

**Table 36. Start logic reset register 0 (STARTRSRP0CLR, address 0x4004 8208) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 11:0 | RSRPIO0_n | Start signal reset for start logic input PIO0_n:PIO0_11 to PIO0_0<br>0 = Do nothing.<br>1 = Writing 1 resets the start signal. | n/a |
| 12 | RSRPIO1_0 | Start signal reset for start logic input PIO1_0<br>0 = Do nothing.<br>1 = Writing 1 resets the start signal. | n/a |
| 31:13 | - | Reserved | n/a |

## 3.5.31 Start logic status register 0

This register reflects the status of the enabled start signal bits. The bit assignment is identical to Table 34. Each bit (if enabled) reflects the state of the start logic, i.e. whether or not a wake-up signal has been received for a given pin.

**Table 37. Start logic status register 0 (STARTSRP0, address 0x4004 820C) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 11:0 | SRPIO0_n | Start signal status for start logic input PIO0_n: PIO0_11 to PIO0_0<br>0 = No start signal received.<br>1 = Start signal pending. | n/a |
| 12 | SRPIO1_0 | Start signal status for start logic input PIO1_0<br>0 = No start signal received.<br>1 = Start signal pending. | n/a |
| 31:13 | - | Reserved | n/a |

## 3.5.32 Deep-sleep mode configuration register

This register controls the behavior of the WatchDog (WD) oscillator and the BOD circuit when the device enters Deep-sleep mode.

This register **must be initialized at least once before entering Deep-sleep mode** with one of the four values shown in Table 38:

**Table 38. Allowed values for PDSLEEPCFG register**

| Configuration | **WD oscillator on** | **WD oscillator off** |
|---|---|---|
| **BOD on** | PDSLEEPCFG = 0x0000 18B7 | PDSLEEPCFG = 0x0000 18F7 |
| **BOD off** | PDSLEEPCFG = 0x0000 18BF | PDSLEEPCFG = 0x0000 18FF |

**Remark:** Failure to initialize and program this register correctly may result in undefined behavior of the microcontroller. The values listed in Table 38 are the only values allowed for PDSLEEPCFG register.

To select the appropriate power configuration for Deep-sleep mode, consider the following:

- BOD: Leaving the BOD circuit enabled will protect the part from a low voltage event occurring while the part is in Deep-sleep mode. However, the BOD circuit causes an additional current drain in Deep-sleep mode.
- WD oscillator: The watchdog oscillator can be left running in Deep-sleep mode to provide a clock for the watchdog timer or a general purpose timer if they are needed for timing a wake-up event (see Section 3.10.3 for details). In this case, the watchdog oscillator analog output frequency must be set to its lowest value (bits FREQSEL in the WDTOSCCTRL = 0001, see Table 12) and all peripheral clocks other than the timer clock must be disabled in the SYSAHBCLKCTRL register (see Table 20) before entering Deep-sleep mode.

  The watchdog oscillator, if running, contributes an additional current drain in Deep-sleep mode.

**Remark:** Reserved bits in this register must always be written as indicated. This register must be initialized correctly before entering Deep-sleep mode.

**Table 39. Deep-sleep configuration register (PDSLEEPCFG, address 0x4004 8230) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | NOTUSED | | Reserved. **Always write these bits as 111.** | 0 |
| 3 | BOD_PD | | BOD power-down control in Deep-sleep mode, see Table 38. | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 5:4 | NOTUSED | | Reserved. **Always write these bits as 11.** | 0 |
| 6 | WDTOSC_PD | | Watchdog oscillator power control in Deep-sleep mode, see Table 38. | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 7 | NOTUSED | | Reserved. **Always write this bit as 1.** | 0 |
| 10:8 | NOTUSED | | Reserved. **Always write these bits as 000.** | 0 |
| 12:11 | NOTUSED | | Reserved. **Always write these bits as 11.** | 0 |
| 31:13 | - | 0 | Reserved | 0 |

### 3.5.33 Wake-up configuration register

The bits in this register determine the state the chip enters when it is waking up from Deep-sleep mode.

By default, the IRC and flash memory are powered and running and the BOD circuit is enabled when the chip wakes up from Deep-sleep mode.

**Remark:** Reserved bits must be always written as indicated.

**Table 40. Wake-up configuration register (PDAWAKECFG, address 0x4004 8234) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | IRCOUT_PD | | IRC oscillator output wake-up configuration | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 1 | IRC_PD | | IRC oscillator power-down wake-up configuration | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 2 | FLASH_PD | | Flash wake-up configuration | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 3 | BOD_PD | | BOD wake-up configuration | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 4 | ADC_PD | | ADC wake-up configuration | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 5 | SYSOSC_PD | | System oscillator wake-up configuration | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 6 | WDTOSC_PD | | Watchdog oscillator wake-up configuration | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 7 | SYSPLL_PD | | System PLL wake-up configuration | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 8 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 9 | - | | Reserved. **Always write this bit as 0.** | 0 |
| 10 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 11 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 12 | - | | Reserved. **Always write this bit as 0.** | 0 |
| 15:13 | - | | Reserved. **Always write these bits as 111.** | 111 |
| 31:16 | - | - | Reserved | - |

### 3.5.34 Power-down configuration register

The bits in the PDRUNCFG register control 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.

By default, the IRC and flash memory are powered and running and the BOD circuit is enabled.

**Remark:** Reserved bits must be always written as indicated.

**Table 41. Power-down configuration register (PDRUNCFG, address 0x4004 8238) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | IRCOUT_PD | | IRC oscillator output power-down | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 1 | IRC_PD | | IRC oscillator power-down | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 2 | FLASH_PD | | Flash power-down | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 3 | BOD_PD | | BOD power-down | 0 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 4 | ADC_PD | | ADC power-down | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 5 | SYSOSC_PD | | System oscillator power-down | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 6 | WDTOSC_PD | | Watchdog oscillator power-down | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 7 | SYSPLL_PD | | System PLL power-down | 1 |
| | | 0 | Powered | |
| | | 1 | Powered down | |
| 8 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 9 | - | | Reserved. **Always write this bit as 0.** | 0 |
| 10 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 11 | - | | Reserved. **Always write this bit as 1.** | 1 |
| 12 | - | | Reserved. **Always write this bit as 0.** | 0 |
| 15:13 | - | | Reserved. **Always write these bits as 111.** | 111 |
| 31:16 | - | - | Reserved | - |

### 3.5.35 Device ID register

This device ID register is a read-only register and contains the part ID for each LPC111x/LPC11Cxx part. This register is also read by the ISP/IAP commands (Section 20.5.11).

**Table 42. Device ID register (DEVICE_ID, address 0x4004 83F4) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | DEVICEID | Part ID numbers for LPC111x/LPC11Cxx parts<br>0x041E 502B = LPC1111FHN33/101<br>0x2516 D02B = LPC1111FHN33/102<br>0x0416 502B = LPC1111FHN33/201<br>0x2516 902B = LPC1111FHN33/202<br>0x042D 502B = LPC1112FHN33/101<br>0x2524 D02B = LPC1112FHN33/102<br>0x0425 502B = LPC1112FHN33/201<br>0x2524 902B = LPC1112FHN33/202<br>0x0434 502B = LPC1113FHN33/201<br>0x2532 902B = LPC1113FHN33/202<br>0x0434 102B = LPC1113FHN33/301<br>0x2532 102B = LPC1113FHN33/302<br>0x0434 102B = LPC1113FBD48/301<br>0x2532 102B = LPC1113FBD48/302<br>0x0444 502B = LPC1114FHN33/201<br>0x2540 902B = LPC1114FHN33/202<br>0x0444 102B = LPC1114FHN33/301<br>0x2540 102B = LPC1114FHN33/302<br>0x0444 102B = LPC1114FBD48/301<br>0x2540 102B = LPC1114FBD48/302<br>0x0444 102B = LPC1114FA44/301<br>0x2540 102B = LPC1114FA44/302<br>0x1421 102B = LPC11C12/FBD48/301<br>0x1440 102B = LPC11C14/FBD48/301<br>0x1431 102B = LPC11C22/FBD48/301<br>0X1430 102B = LPC11C24/FBD48/301 | part-dependent |

## 3.6 Reset

Reset has four sources on the LPC111x/LPC11Cxx: the $\overline{\text{RESET}}$ pin, Watchdog Reset, Power-On Reset (POR), and Brown Out Detect (BOD). In addition, there is a software reset.

The $\overline{\text{RESET}}$ pin is a Schmitt trigger input pin. Assertion of chip Reset by any source, once the operating voltage attains a usable level, starts the IRC causing reset to remain asserted until the external Reset is de-asserted, the oscillator is running, and the flash controller has completed its initialization.

On the assertion of a reset source external to the Cortex-M0 CPU (POR, BOD reset, External reset, and Watchdog reset), the following processes are initiated:

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

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

## 3.7 Start-up behavior

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

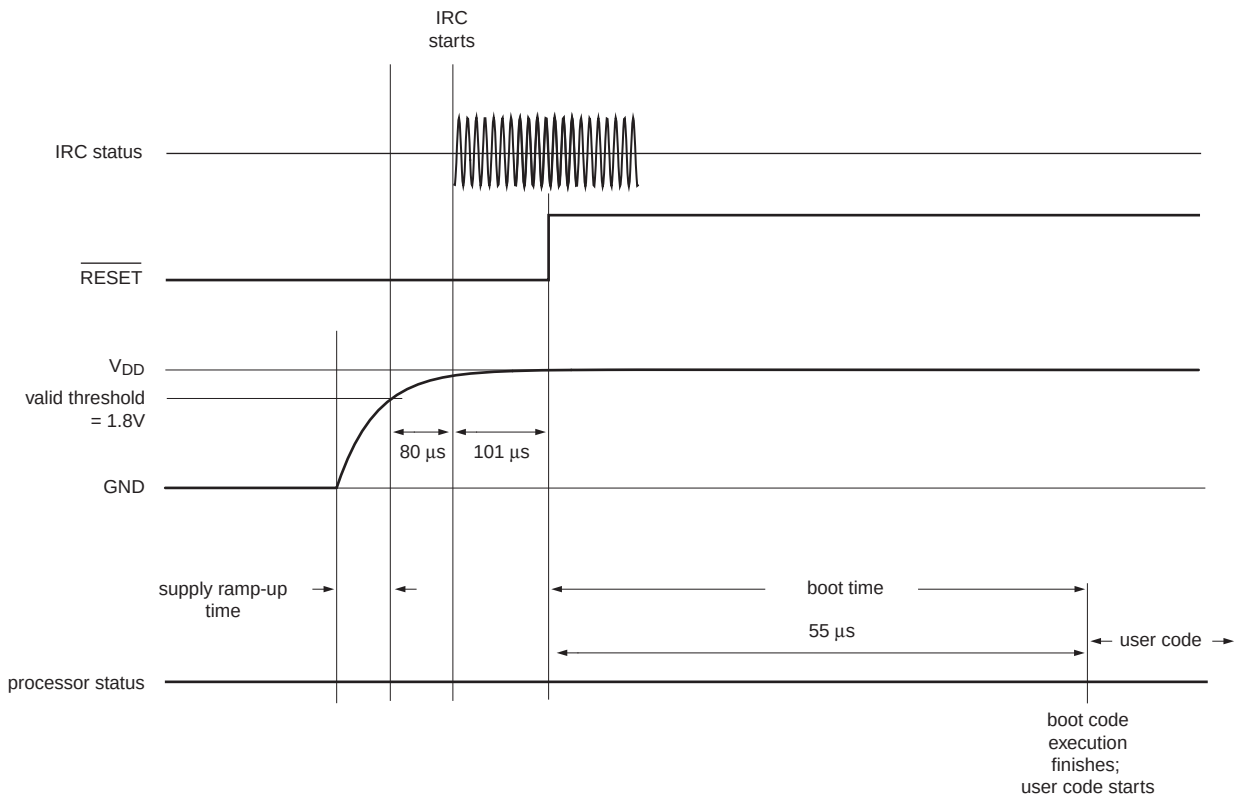



**Fig 4. Start-up timing**

## 3.8 Brown-out detection

The LPC111x/LPC11Cxx includes four levels for monitoring the voltage on the $V_{DD}$ pin. If this voltage falls below one of the four selected levels, the BOD asserts an interrupt signal to the NVIC. This signal can be enabled for interrupt in the Interrupt Enable Register in the NVIC in order to cause a CPU interrupt; if not, software can monitor the signal by reading the NVIC status register (see Table 52). An additional four threshold levels can be selected to cause a forced reset of the chip (see Table 32).

## 3.9 Power management

The LPC111x/LPC11Cxx 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 three special modes of processor power reduction: Sleep mode, Deep-sleep mode, and Deep power-down mode.

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

### 3.9.1 Active mode

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

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

#### 3.9.1.1 Power configuration in Active mode

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

- The SYSAHBCLKCTRL register controls which memories and peripherals are running (Table 20).
- The power to various analog blocks (PLL, oscillators, the ADC, the BOD circuit, and the flash block) can be controlled at any time individually through the PDRUNCFG register (Table 41).
- The clock source for the system clock can be selected from the IRC (default), the system oscillator, or the watchdog oscillator (see Figure 3 and related registers).
- The system clock frequency can be selected by the SYSPLLCTRL (Table 9) and the SYSAHBCLKDIV register (Table 19).
- Selected peripherals (UART, SPI0/1, WDT) use individual peripheral clocks with their own clock dividers. The peripheral clocks can be shut down through the corresponding clock divider registers (Table 21 to Table 23).

### 3.9.2 Sleep mode

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

Peripheral functions, if selected to be clocked in the SYSAHBCLKCTRL register, continue operation during Sleep mode and may generate interrupts to cause the processor to resume execution. Sleep mode eliminates dynamic power used by the processor itself, memory systems and their related controllers, and internal buses. The processor state and registers, peripheral registers, and internal SRAM values are maintained, and the logic levels of the pins remain static.

#### 3.9.2.1 Power configuration in Sleep mode

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

- The clock remains running.

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

#### 3.9.2.2 Programming Sleep mode

The following steps must be performed to enter Sleep mode:

1. The DPDEN bit in the PCON register must be set to zero (Table 47).
2. The SLEEPDEEP bit in the ARM Cortex-M0 SCR register must be set to zero, see (Table 345).
3. Use the ARM Cortex-M0 Wait-For-Interrupt (WFI) instruction.

#### 3.9.2.3 Wake-up from Sleep mode

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

### 3.9.3 Deep-sleep mode

In Deep-sleep mode, the system clock to the processor is disabled as in Sleep mode. All analog blocks are powered down, except for the BOD circuit and the watchdog oscillator, which must be selected or deselected during Deep-sleep mode in the PDSLEEPCFG register.

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

#### 3.9.3.1 Power configuration in Deep-sleep mode

Power consumption in Deep-sleep mode is determined by the Deep-sleep power configuration setting in the PDSLEEPCFG (Table 39) register:

- The only clock source available in Deep-sleep mode is the watchdog oscillator. The watchdog oscillator can be left running in Deep-sleep mode if required for timer-controlled wake-up (see Section 3.10.3). All other clock sources (the IRC and system oscillator) and the system PLL are shut down. The watchdog oscillator analog output frequency must be set to the lowest value of its analog clock output (bits FREQSEL in the WDTOSCCTRL = 0001, see Table 12).
- The BOD circuit can be left running in Deep-sleep mode if required by the application.
- If the watchdog oscillator is running in Deep-sleep mode, only the watchdog timer or one of the general-purpose timers should be enabled in SYSAHBCLKCTRL register to minimize power consumption.

#### 3.9.3.2 Programming Deep-sleep mode

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

1. The DPDEN bit in the PCON register must be set to zero (Table 47).

2. Select the power configuration in Deep-sleep mode in the PDSLEEPCFG (Table 39) register.
   a. If a timer-controlled wake-up is needed, ensure that the watchdog oscillator is powered in the PDRUNCFG register and switch the clock source to WD oscillator in the MAINCLKSEL register (Table 17).
   b. If no timer-controlled wake-up is needed and the watchdog oscillator is shut down, ensure that the IRC is powered in the PDRUNCFG register and switch the clock source to IRC in the MAINCLKSEL register (Table 17). This ensures that the system clock is shut down glitch-free.
3. Select the power configuration after wake-up in the PDAWAKECFG (Table 40) register.
4. If an external pin is used for wake-up, enable and clear the wake-up pin in the start logic registers (Table 34 to Table 37), and enable the start logic interrupt in the NVIC.
5. In the SYSAHBCLKCTRL register (Table 20), disable all peripherals except counter/timer or WDT if needed.
6. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register (Table 345).
7. Use the ARM WFI instruction.

#### 3.9.3.3 Wake-up from Deep-sleep mode

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

- Signal on an external pin. For this purpose, pins PIO0_0 to PIO0_11 and PIO1_0 can be enabled as inputs to the start logic. The start logic does not require any clocks and generates the interrupt if enabled in the NVIC to wake up from Deep-sleep mode.
- Input signal to the start logic created by a match event on one of the general purpose timer external match outputs. The pin holding the timer match function must be enabled as start logic input in the NVIC, the corresponding timer must be enabled in the SYSAHBCLKCTRL register, and the watchdog oscillator must be running in Deep-sleep mode (for details see Section 3.10.3).
- Reset from the BOD circuit. In this case, the BOD circuit must be enabled in the PDSLEEPCFG register, and the BOD reset must be enabled in the BODCTRL register (Table 32).
- Reset from the watchdog timer. In this case, the watchdog oscillator must be running in Deep-sleep mode (see PDSLEEPCFG register), and the WDT must be enabled in the SYSAHBCLKCTRL register.
- A reset signal from the external $\overline{\text{RESET}}$ pin.

**Remark:** If the watchdog oscillator is running in Deep-sleep mode, its frequency determines the wake-up time causing the wake-up time to be longer than waking up with the IRC.

### 3.9.4 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.

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 five 32-bit general purpose registers of the PMU block.

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

#### 3.9.4.1 Power configuration in Deep power-down mode

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

#### 3.9.4.2 Programming Deep power-down mode

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

1. Write one to the DPDEN bit in the PCON register (see Table 47).
2. Store data to be retained in the general purpose registers (Table 48).
3. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register (Table 345).
4. Ensure that the IRC is powered by setting bits IRCOUT_PD and IRC_PD to zero in the PDRUNCFG register before entering Deep power-down mode.
5. Use the ARM WFI instruction.

**Remark:** The WAKEUP pin must be pulled HIGH externally before entering Deep power-down mode.

#### 3.9.4.3 Wake-up from Deep power-down mode

Pulling the WAKEUP pin LOW wakes up the LPC111x/LPC11Cxx from Deep power-down, and the chip goes through the entire reset process (Section 3.6). The minimum pulse width for the HIGH-to-LOW transition on the WAKEUP pin is 50 ns.

Follow these steps to wake up the chip from Deep power-down mode:

1. A wake-up signal is generated when a HIGH-to-LOW transition occurs externally on the WAKEUP pin with a pulse length of at least 50 ns while the part is in Deep power-down mode.
   - The PMU will turn on the on-chip voltage regulator. When the core voltage reaches the power-on-reset (POR) trip point, a system reset will be triggered and the chip re-boots.
   - All registers except the GPREG0 to GPREG4 and PCON will be in their reset state.
2. Once the chip has booted, read the deep power-down flag in the PCON register (Table 47) to verify that the reset was caused by a wake-up event from Deep power-down.
3. Clear the deep power-down flag in the PCON register (Table 47).
4. (Optional) Read the stored data in the general purpose registers (Table 48 and Table 49).
5. Set up the PMU for the next Deep power-down cycle.

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

## 3.10 Deep-sleep mode details

### 3.10.1 IRC oscillator

The IRC is the only oscillator on the LPC111x/LPC11Cxx that can always shut down glitch-free. Therefore it is recommended that the user switches the clock source to IRC before the chip enters Deep-sleep mode.

### 3.10.2 Start logic

The Deep-sleep mode is exited when the start logic indicates an interrupt to the ARM core. The port pins PIO0_0 to PIO0_11 and PIO1_0 are connected to the start logic and serve as wake-up pins. The user must program the start logic registers for each input to set the appropriate edge polarity for the corresponding wake-up event. Furthermore, the interrupts corresponding to each input must be enabled in the NVIC. Interrupts 0 to 12 in the NVIC correspond to 13 PIO pins (see Section 3.5.28).

The start logic does not require a clock to run because it uses the input signals on the enabled pins to generate a clock edge when enabled. Therefore, the start logic signals should be cleared (see Table 36) before use.

The start logic can also be used in Active mode to provide a vectored interrupt using the LPC111x/LPC11Cxx's input pins.

### 3.10.3 Using the general purpose counter/timers to create a self-wake-up event

If enabled in Deep-sleep mode through the SYSAHBCLKCFG register, the counter/timers can count clock cycles of the watchdog oscillator and create a match event when the number of cycles equals a preset match value. The match event causes the corresponding match output pin to go HIGH, LOW, or toggle. The state of the match output pin is also monitored by the start logic and can trigger a wake-up interrupt if that pin is enabled in the NVIC and the start logic trigger is configured accordingly in the start logic edge control register (see Table 34).

The following steps must be performed to configure the counter/timer and create a timed Deep-sleep self-wake-up event:

1. Configure the port pin as match output in the IOCONFIG block. Select from pins PIO0_1 or PIO0_8 to PIO0_11, which are inputs to the start logic and also hold a match output function.
2. In the corresponding counter/timer, set the match value, and configure the match output for the selected pin.
3. Select the watchdog oscillator to run in Deep-sleep mode in the PDSLEEPCFG register.
4. Switch the clock source to the watchdog oscillator in the MAINCLKSEL register (Table 17) and ensure the watchdog oscillator is powered in the PDRUNCFG register.
5. Enable the pin, configure its edge detect function, and reset the start logic in the start logic registers (Table 34 to Table 37), and enable the interrupt in the NVIC.
6. Disable all other peripherals in the SYSAHBCLKCTRL register.
7. Ensure that the DPDEN bit in the PCON register is set to zero (Table 47).

8. Write one to the SLEEPDEEP bit in the ARM Cortex-M0 SCR register (Table 345).
9. Start the counter/timer.
10. Use the ARM WFI instruction to enter Deep-sleep mode.

## 3.11 System PLL functional description

The LPC111x/LPC11Cxx uses the system PLL to create the clocks for the core and peripherals.

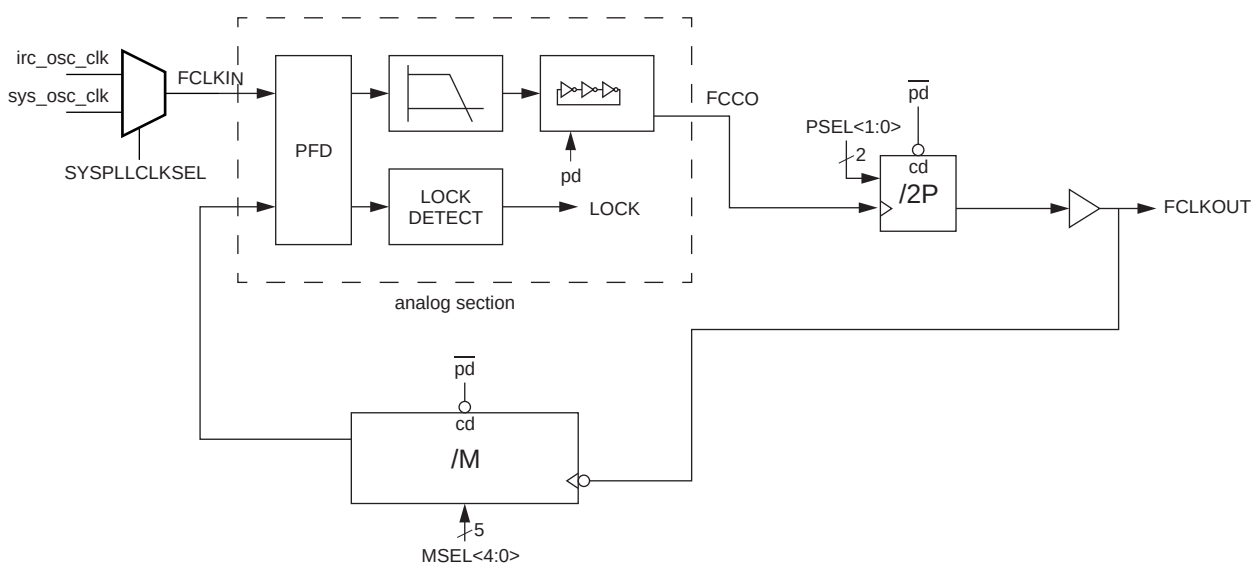



**Fig 5. System PLL block diagram**

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 clock(s), or are sent directly to the output(s). The main output clock is then divided by M by the programmable feedback divider to generate the feedback clock. The output signal of the phase-frequency detector is also monitored by the lock detector, to signal when the PLL has locked on to the input clock.

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

### 3.11.1 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.

### 3.11.2 Power-down control

To reduce the power consumption when the PLL clock is not needed, a Power-down mode has been incorporated. This mode is enabled by setting the SYSPLL_PD bits to one in the Power-down configuration register (Table 41). In this mode, the internal current reference will be turned off, the oscillator and the phase-frequency detector will be stopped and the dividers will enter a reset state. While in Power-down mode, the lock output will be low to indicate that the PLL is not in lock. When the Power-down mode is terminated by setting the SYSPLL_PD bits 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.

### 3.11.3 Divider ratio programming

#### Post divider

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

#### Feedback divider

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

#### 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.

### 3.11.4 Frequency selection

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

**Table 43. PLL frequency parameters**

| Parameter | System PLL |
|---|---|
| FCLKIN | Frequency of sys_pllclkin (input clock to the system PLL) from the SYSPLLCLKSEL multiplexer (see Section 3.5.9). |
| FCCO | Frequency of the Current Controlled Oscillator (CCO); 156 to 320 MHz. |
| FCLKOUT | Frequency of sys_pllclkout. FCLKOUT must be < 100 MHz. |
| P | System PLL post divider ratio; PSEL bits in SYSPLLCTRL (see Section 3.5.3). |
| M | System PLL feedback divider register; MSEL bits in SYSPLLCTRL (see Section 3.5.3). |

#### 3.11.4.1 Normal mode

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

$$FCLKOUT = M \times FCLKIN = (FCCO)/(2 \times P) \quad (1)$$

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

1. Specify the input clock frequency FCLKIN.
2. Calculate M to obtain the desired output frequency FCLKOUT with M = FCLKOUT / FCLKIN.
3. Find a value so that FCCO = 2 × P × FCLKOUT.
4. Verify that all frequencies and divider values conform to the limits specified in Table 9.
5. Ensure that FCLKOUT < 100 MHz.

Table 44 shows how to configure the PLL for a 12 MHz crystal oscillator using the SYSPLLCTRL register (Table 9). The main clock is equivalent to the system clock if the system clock divider SYSAHBCLKDIV is set to one (see Table 19).

**Table 44. PLL configuration examples**

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

#### 3.11.4.2 Power-down mode

In this mode, the internal current reference is turned off, the oscillator and the phase-frequency detector are stopped, and the dividers enter a reset state. While in Power-down mode, the lock output is be LOW to indicate that the PLL is not in lock. When the Power-down mode is terminated by setting the SYSPLL_PD bit to zero in the Power-down configuration register (Table 41), the PLL resumes its normal operation and asserts the lock signal HIGH once it has regained lock on the input clock.

## 3.12 Flash memory access

Depending on the system clock frequency, access to the flash memory can be configured with various access times by writing to the FLASHCFG register at address 0x4003 C010. This register is part of the flash configuration block (see Figure 2).

**Remark:** Improper setting of this register may result in incorrect operation of the LPC111x/LPC11Cxx flash memory.

**Table 45. Flash configuration register (FLASHCFG, address 0x4003 C010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | FLASHTIM | | Flash memory access time. FLASHTIM +1 is equal to the number of system clocks used for flash access. | 10 |
| | | 00 | 1 system clock flash access time (for system clock frequencies of up to 20 MHz). | |
| | | 01 | 2 system clocks flash access time (for system clock frequencies of up to 40 MHz). | |
| | | 10 | 3 system clocks flash access time (for system clock frequencies of up to 50 MHz). | |
| | | 11 | Reserved. | |
| 31:2 | - | - | Reserved. **User software must not change the value of these bits. Bits 31:2 must be written back exactly as read**. | - |

## 4.1 How to read this chapter

**Remark:** For parts LPC111x/102/202/302, also refer to Chapter 5 for power control.

## 4.2 Introduction

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

## 4.3 Register description

**Table 46. Register overview: PMU (base address 0x4003 8000)**

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

### 4.3.1 Power control register

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

**Table 47. Power control register (PCON, address 0x4003 8000) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | - | - | Reserved. Do not write 1 to this bit. | 0x0 |
| 1 | DPDEN | | Deep power-down mode enable | 0 |
| | | 0 | ARM WFI will enter Sleep or Deep-sleep mode (clock to ARM Cortex-M0 core turned off). | |
| | | 1 | ARM WFI will enter Deep-power down mode (ARM Cortex-M0 core powered-down). | |
| 7:2 | - | - | Reserved. Do not write ones to this bit. | 0x0 |

Table 47. Power control register (PCON, address 0x4003 8000) bit description *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 8 | SLEEPFLAG | | Sleep mode flag | 0 |
| | | 0 | Read: No power-down mode entered. LPC111x/LPC11Cxx is in Active mode. Write: No effect. | |
| | | 1 | Read: Sleep/Deep-sleep or Deep power-down mode entered. Write: Writing a 1 clears the SLEEPFLAG bit to 0. | |
| 10:9 | - | - | Reserved. Do not write ones to this bit. | 0x0 |
| 11 | DPDFLAG | | Deep power-down flag | 0x0 |
| | | 0 | Read: Deep power-down mode **not** entered. Write: No effect. | 0x0 |
| | | 1 | Read: Deep power-down mode entered. Write: Clear the Deep power-down flag. | 0x0 |
| 31:12 | - | - | Reserved. Do not write ones to this bit. | 0x0 |

### 4.3.2 General purpose registers 0 to 3

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

Table 48. General purpose registers 0 to 3 (GPREG0 - GPREG3, address 0x4003 8004 to 0x4003 8010) bit description

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

### 4.3.3 General purpose register 4

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

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

Table 49. General purpose register 4 (GPREG4, address 0x4003 8014) bit description

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

Table 49. General purpose register 4 (GPREG4, address 0x4003 8014) bit description

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

## 4.4 Functional description

For details of entering and exiting Deep power-down mode, see Section 3.9.4.

## 5.1 How to read this chapter

The power profiles are available for parts LPC111x/102/202/302 only (LPC1100L series).

## 5.2 Features

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

## 5.3 Description

The API calls to the ROM are performed by executing functions which are pointed by a pointer within the ROM Driver Table. Figure 6 shows the pointer structure used to call the Power Profiles API.

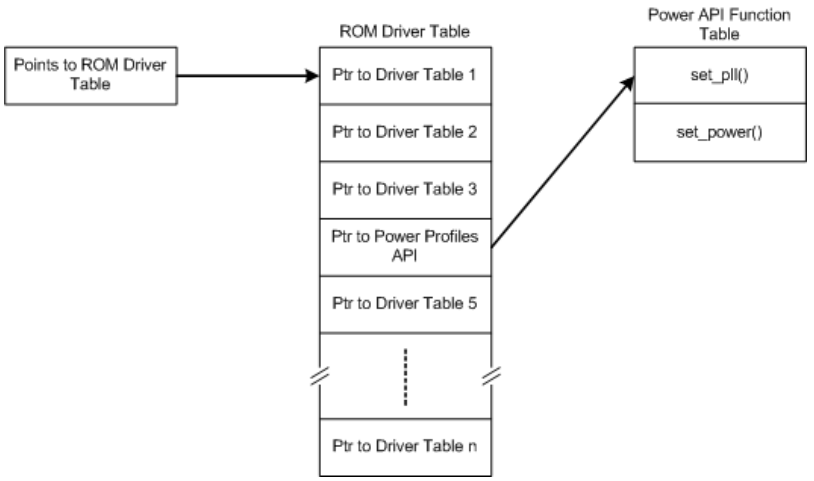



**Fig 6. Power profiles pointer structure**

## 5.4 Definitions

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

```
typedef struct _PWRD {
     void (*set_pll)(unsigned int cmd[], unsigned int resp[]);
     void (*set_power)(unsigned int cmd[], unsigned int resp[]);
} PWRD;
typedef struct _ROM {
     const PWRD * pWRD;
} ROM;
```

```
ROM ** rom = (ROM **) 0x1FFF1FF8;
unsigned int command[4], result[2];
```

## 5.5 Clocking routine

### 5.5.1 set_pll

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

**IMPORTANT:** Before this routine is invoked, the PLL clock source (IRC/system oscillator) must be selected (Table 15), the main clock source must be set to the input clock to the system PLL (Table 17) and the system/AHB clock divider must be set to 1 (Table 19).

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

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

**Table 50. set_pll routine**

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

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

```
/* set_pll mode options */
#define    CPU_FREQ_EQU          0
#define    CPU_FREQ_LTE          1
#define    CPU_FREQ_GTE          2
#define    CPU_FREQ_APPROX       3
/* set_pll result0 options */
#define    PLL_CMD_SUCCESS       0
#define    PLL_INVALID_FREQ      1
#define    PLL_INVALID_MODE      2
#define    PLL_FREQ_NOT_FOUND    3
#define    PLL_NOT_LOCKED        4
```

#### 5.5.1.1 System PLL input frequency and expected system clock

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

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

#### 5.5.1.2 Mode

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

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

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

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

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

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

#### 5.5.1.3 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. If a non-zero value is provided, that is 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*.

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

#### 5.5.1.4 Code examples

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

##### 5.5.1.4.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 12000;
command[1] = 60000;
command[2] = CPU_FREQ_EQU;
```

```
command[3] = 0;
(*rom)->pWRD->set_pll(command, result);
```

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

#### 5.5.1.4.2 Invalid frequency selection (system clock divider restrictions)

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

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

#### 5.5.1.4.3 Exact solution cannot be found (PLL)

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

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

#### 5.5.1.4.4 System clock less than or equal to the expected value

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

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

#### 5.5.1.4.5 System clock greater than or equal to the expected value

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

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

#### 5.5.1.4.6 System clock approximately equal to the expected value

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

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

## 5.6 Power routine

### 5.6.1 set_power

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

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

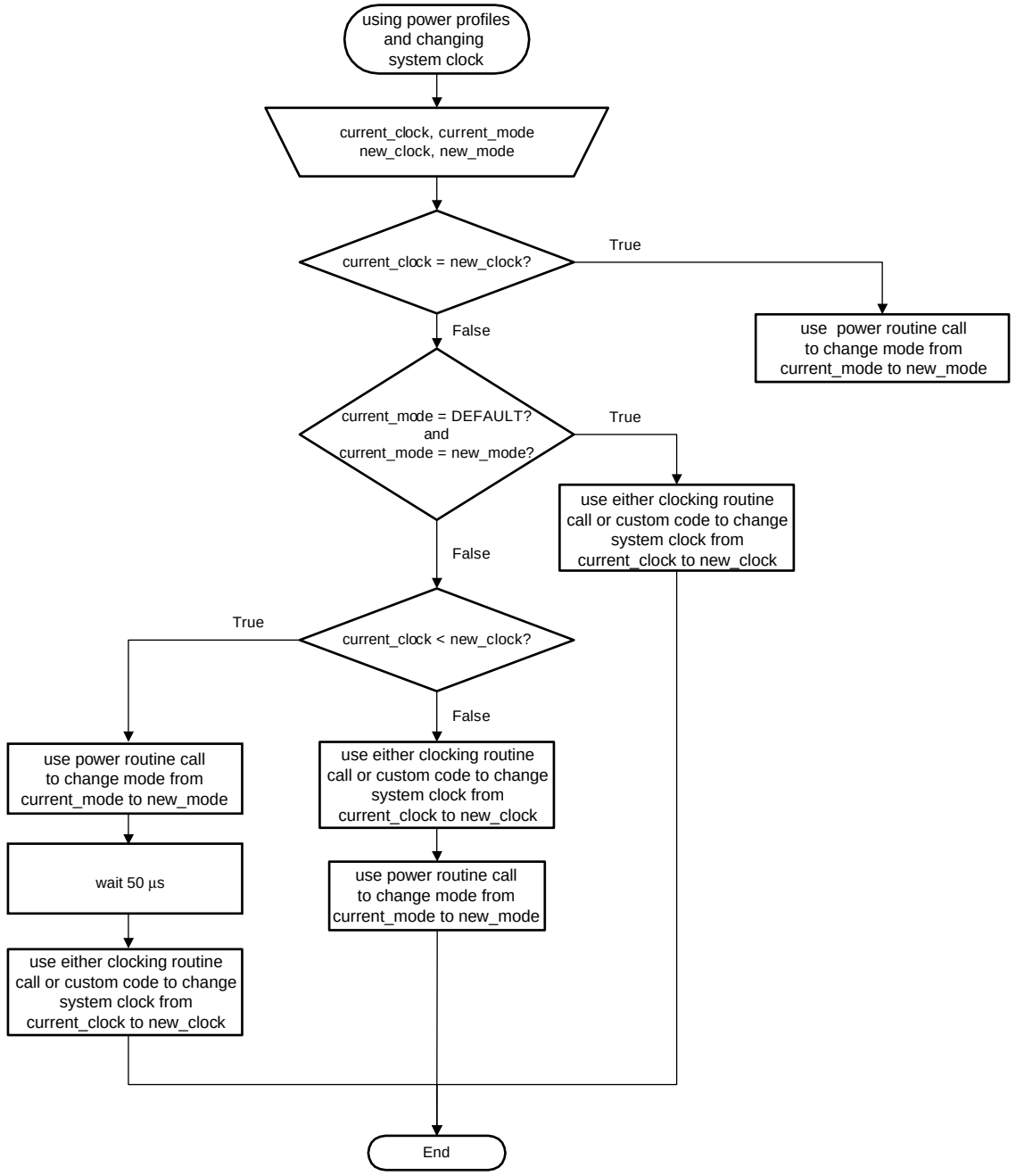



**Fig 7. Power profiles usage**

Table 51. set_power routine

| Routine | set_power |
|---|---|
| Input | **Param0:** new system clock (in MHz)<br>**Param1:** mode (PWR_DEFAULT, PWR_CPU_PERFORMANCE, PWR_EFFICIENCY, PWR_LOW_CURRENT)<br>**Param2:** current system clock (in MHz) |
| Result | **Result0:** PWR_CMD_SUCCESS \| PWR_INVALID_FREQ \| PWR_INVALID_MODE |

The following definitions are needed for set_power routine calls:

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

#### 5.6.1.1 New system clock

The new system clock is the clock rate at which the microcontroller will be running after either a successful execution of a clocking routine call or a similar code provided by the user. This operand must be an integer between 1 to 50 MHz inclusive. If a value out of this range is supplied, *set_power* returns PWR_INVALID_FREQ and does not change the power control system.

#### 5.6.1.2 Mode

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

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

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

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

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

#### 5.6.1.3 Current system clock

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

#### 5.6.1.4 Code examples

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

##### 5.6.1.4.1 Invalid frequency (device maximum clock rate exceeded)

```
command[0] = 55;
command[1] = PWR_CPU_PERFORMANCE;
command[2] = 12;
(*rom)->pWRD->set_power(command, result);
```

The above setup would be used in a system running at 12 MHz attempting to switch to 55 MHz system clock, with a need for maximum CPU processing power. Since the specified 55 MHz clock is above the 50 MHz maximum, *set_power* returns PWR_INVALID_FREQ in *result[0]* without changing anything in the existing power setup.

##### 5.6.1.4.2 An applicable power setup

```
command[0] = 24;
command[1] = PWR_CPU_EFFICIENCY;
command[2] = 12;
(*rom)->pWRD->set_power(command, result);
```

The above code specifies that an application running at a system clock of 12 MHz will switch to 24 MHz with emphasis on efficiency. *set_power* returns PWR_CMD_SUCCESS in *result[0]* after configuring the microcontroller's internal power control features.

## 6.1 How to read this chapter

The C_CAN controller interrupt is available on parts LPC11Cxx only.

## 6.2 Introduction

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

## 6.3 Features

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

## 6.4 Interrupt sources

Table 52 lists the interrupt sources for each peripheral function. Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller. Each line may represent more than one interrupt source. There is no significance or priority about what line is connected where, except for certain standards from ARM.

See Section 22.5.2 for the NVIC register bit descriptions.

**Table 52. Connection of interrupt sources to the Vectored Interrupt Controller**

| Exception Number | Vector Offset | Function | Flag(s) |
|---|---|---|---|
| 12 to 0 | | start logic wake-up interrupts | Each interrupt is connected to a PIO input pin serving as wake-up pin from Deep-sleep mode; Interrupt 0 to 11 correspond to PIO0_0 to PIO0_11 and interrupt 12 corresponds to PIO1_0; see Section 3.5.28. |
| 13 | | C_CAN | C_CAN interrupt |
| 14 | | SPI/SSP1 | Tx FIFO half empty<br>Rx FIFO half full<br>Rx Timeout<br>Rx Overrun |
| 15 | | $I^2C$ | SI (state change) |

**Table 52. Connection of interrupt sources to the Vectored Interrupt Controller**

| Exception Number | Vector Offset | Function | Flag(s) |
|---|---|---|---|
| 16 | | CT16B0 | Match 0 - 2<br>Capture 0 |
| 17 | | CT16B1 | Match 0 - 1<br>Capture 0 |
| 18 | | CT32B0 | Match 0 - 3<br>Capture 0 |
| 19 | | CT32B1 | Match 0 - 3<br>Capture 0 |
| 20 | | SPI/SSP0 | Tx FIFO half empty<br>Rx FIFO half full<br>Rx Timeout<br>Rx Overrun |
| 21 | | UART | Rx Line Status (RLS)<br>Transmit Holding Register Empty (THRE)<br>Rx Data Available (RDA)<br>Character Time-out Indicator (CTI)<br>End of Auto-Baud (ABEO)<br>Auto-Baud Time-Out (ABTO) |
| 22 | | - | Reserved |
| 23 | | - | Reserved |
| 24 | | ADC | A/D Converter end of conversion |
| 25 | | WDT | Watchdog interrupt (WDINT) |
| 26 | | BOD | Brown-out detect |
| 27 | | - | Reserved |
| 28 | | PIO_3 | GPIO interrupt status of port 3 |
| 29 | | PIO_2 | GPIO interrupt status of port 2 |
| 30 | | PIO_1 | GPIO interrupt status of port 1 |
| 31 | | PIO_0 | GPIO interrupt status of port 0 |

## 7.1 How to read this chapter

The implementation of the I/O configuration registers varies for different LPC111x/LPC11Cxx parts and packages. Table 54 shows which IOCON registers are used on the different packages.

On the LPC11C12/C14, functions PIO3_4 and PIO3_5 are not available. Instead, two pins are dedicated to the C_CAN receive and transmit functions (see Table 54) without pull-up or pull-down resistors. The C_CAN pins have no programmable pin configuration.

On the LPC11C22/C24, pins PIO1_9, PIO2_4, PIO2_5, and PIO2_9 are not available and are replaced by the on-chip CAN transceiver pins. The CAN transceiver pins have no programmable pin configuration.

## 7.2 Introduction

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

- pin function
- internal pull-up/pull-down resistor or bus keeper function
- hysteresis
- analog input or digital mode for pads hosting the ADC inputs
- $I^2C$ mode for pads hosting the $I^2C$-bus function

## 7.3 General description

The IOCON registers control the function (GPIO or peripheral function), the input mode, and the hysteresis of all PIOn_m pins. In addition, the $I^2C$-bus pins can be configured for different $I^2C$-bus modes. If a pin is used as input pin for the ADC, an analog input mode can be selected.

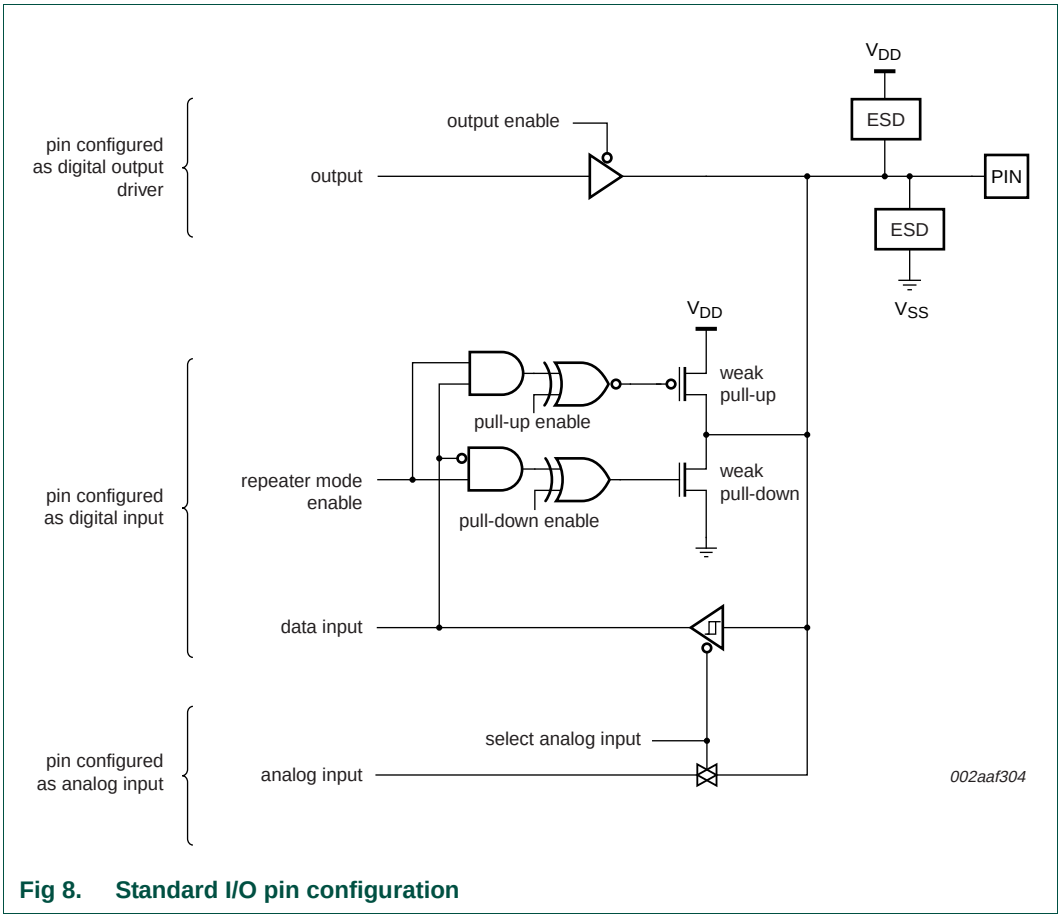



**Fig 8. Standard I/O pin configuration**

### 7.3.1 Pin function

The FUNC bits in the IOCON registers can be set to GPIO (FUNC = 000) or to a peripheral function. If the pins are GPIO pins, the GPIOnDIR registers determine whether the pin is configured as an input or output (see Section 9.3.2). For any peripheral function, the pin direction is controlled automatically depending on the pin's functionality. The GPIOnDIR registers have no effect for peripheral functions.

### 7.3.2 Pin mode

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

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

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

### 7.3.3 Hysteresis

The input buffer for digital functions can be configured with hysteresis or as plain buffer through the IOCON registers (see the *LPC111x and LPC11Cx data sheets* for details).

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

### 7.3.4 A/D-mode

In A/D-mode, the digital receiver is disconnected to obtain an accurate input voltage for analog-to-digital conversions. This mode can be selected in those IOCON registers that control pins with an analog function. If A/D mode is selected, Hysteresis and Pin mode settings have no effect.

For pins without analog functions, the A/D-mode setting has no effect.

### 7.3.5 $I^2C$ mode

If the $I^2C$ function is selected by the FUNC bits of registers IOCON_PIO0_4 (Table 65) and IOCON_PIO0_5 (Table 66), then the $I^2C$-bus pins can be configured for different $I^2C$-modes:

- Standard mode/Fast-mode $I^2C$ with input glitch filter (this includes an open-drain output according to the $I^2C$-bus specification).
- Fast-mode Plus with input glitch filter (this includes an open-drain output according to the $I^2C$-bus specification). In this mode, the pins function as high-current sinks.
- Standard open-drain I/O functionality without input filter.

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

## 7.4 Register description

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

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

Some input functions (SCK0, DSR0, DCD0, and RI0) are multiplexed to several physical pins. The IOCON_LOC registers select the pin location for each of these functions.

**Remark:** The IOCON registers are listed in order of their memory locations in Table 53, which correspond to the order of their physical pin numbers in the LQFP48 package starting at the upper left corner with pin 1 (PIO2_6). See Table 54 for a listing of IOCON registers ordered by port number.

The IOCON location registers are used to select a physical pin for multiplexed functions.

**Remark:** Note that once the pin location has been selected, the function still must be configured in the corresponding IOCON registers for the function to be usable on that pin.

**Table 53. Register overview: I/O configuration (base address 0x4004 4000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| IOCON_PIO2_6 | R/W | 0x000 | I/O configuration for pin PIO2_6 | 0xD0 | Table 55 |
| - | R/W | 0x004 | Reserved | - | - |
| IOCON_PIO2_0 | R/W | 0x008 | I/O configuration for pin PIO2_0/DTR/SSEL1 | 0xD0 | Table 56 |
| IOCON_RESET_PIO0_0 | R/W | 0x00C | I/O configuration for pin RESET/PIO0_0 | 0xD0 | Table 57 |
| IOCON_PIO0_1 | R/W | 0x010 | I/O configuration for pin PIO0_1/CLKOUT/CT32B0_MAT2 | 0xD0 | Table 55 |
| IOCON_PIO1_8 | R/W | 0x014 | I/O configuration for pin PIO1_8/CT16B1_CAP0 | 0xD0 | Table 59 |
| - | R/W | 0x018 | Reserved | - | - |
| IOCON_PIO0_2 | R/W | 0x01C | I/O configuration for pin PIO0_2/SSEL0/CT16B0_CAP0 | 0xD0 | Table 60 |
| IOCON_PIO2_7 | R/W | 0x020 | I/O configuration for pin PIO2_7 | 0xD0 | Table 61 |
| IOCON_PIO2_8 | R/W | 0x024 | I/O configuration for pin PIO2_8 | 0xD0 | Table 62 |
| IOCON_PIO2_1 | R/W | 0x028 | I/O configuration for pin PIO2_1/DSR/SCK1 | 0xD0 | Table 63 |
| IOCON_PIO0_3 | R/W | 0x02C | I/O configuration for pin PIO0_3 | 0xD0 | Table 64 |
| IOCON_PIO0_4 | R/W | 0x030 | I/O configuration for pin PIO0_4/SCL | 0x00 | Table 65 |
| IOCON_PIO0_5 | R/W | 0x034 | I/O configuration for pin PIO0_5/SDA | 0x00 | Table 66 |
| IOCON_PIO1_9 | R/W | 0x038 | I/O configuration for pin PIO1_9/CT16B1_MAT0 | 0xD0 | Table 67 |
| IOCON_PIO3_4 | R/W | 0x03C | I/O configuration for pin PIO3_4 | 0xD0 | Table 68 |
| IOCON_PIO2_4 | R/W | 0x040 | I/O configuration for pin PIO2_4 | 0xD0 | Table 69 |
| IOCON_PIO2_5 | R/W | 0x044 | I/O configuration for pin PIO2_5 | 0xD0 | Table 70 |
| IOCON_PIO3_5 | R/W | 0x048 | I/O configuration for pin PIO3_5 | 0xD0 | Table 71 |
| IOCON_PIO0_6 | R/W | 0x04C | I/O configuration for pin PIO0_6/SCK0 | 0xD0 | Table 72 |
| IOCON_PIO0_7 | R/W | 0x050 | I/O configuration for pin PIO0_7/CTS | 0xD0 | Table 73 |
| IOCON_PIO2_9 | R/W | 0x054 | I/O configuration for pin PIO2_9 | 0xD0 | Table 74 |
| IOCON_PIO2_10 | R/W | 0x058 | I/O configuration for pin PIO2_10 | 0xD0 | Table 75 |
| IOCON_PIO2_2 | R/W | 0x05C | I/O configuration for pin PIO2_2/DCD/MISO1 | 0xD0 | Table 76 |
| IOCON_PIO0_8 | R/W | 0x060 | I/O configuration for pin PIO0_8/MISO0/CT16B0_MAT0 | 0xD0 | Table 77 |
| IOCON_PIO0_9 | R/W | 0x064 | I/O configuration for pin PIO0_9/MOSI0/CT16B0_MAT1 | 0xD0 | Table 78 |
| IOCON_SWCLK_PIO0_10 | R/W | 0x068 | I/O configuration for pin SWCLK/PIO0_10/ SCK0/CT16B0_MAT2 | 0xD0 | Table 79 |
| IOCON_PIO1_10 | R/W | 0x06C | I/O configuration for pin PIO1_10/AD6/CT16B1_MAT1 | 0xD0 | Table 80 |
| IOCON_PIO2_11 | R/W | 0x070 | I/O configuration for pin PIO2_11/SCK0 | 0xD0 | Table 81 |
| IOCON_R_PIO0_11 | R/W | 0x074 | I/O configuration for pin R/PIO0_11/AD0/CT32B0_MAT3 | 0xD0 | Table 82 |

**Table 53. Register overview: I/O configuration (base address 0x4004 4000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| IOCON_R_PIO1_0 | R/W | 0x078 | I/O configuration for pin R/PIO1_0/AD1/CT32B1_CAP0 | 0xD0 | Table 83 |
| IOCON_R_PIO1_1 | R/W | 0x07C | I/O configuration for pin R/PIO1_1/AD2/CT32B1_MAT0 | 0xD0 | Table 84 |
| IOCON_R_PIO1_2 | R/W | 0x080 | I/O configuration for pin R/PIO1_2/AD3/CT32B1_MAT1 | 0xD0 | Table 85 |
| IOCON_PIO3_0 | R/W | 0x084 | I/O configuration for pin PIO3_0/$\overline{\text{DTR}}$ | 0xD0 | Table 86 |
| IOCON_PIO3_1 | R/W | 0x088 | I/O configuration for pin PIO3_1/$\overline{\text{DSR}}$ | 0xD0 | Table 87 |
| IOCON_PIO2_3 | R/W | 0x08C | I/O configuration for pin PIO2_3/$\overline{\text{RI}}$/MOSI1 | 0xD0 | Table 88 |
| IOCON_SWDIO_PIO1_3 | R/W | 0x090 | I/O configuration for pin SWDIO/PIO1_3/AD4/CT32B1_MAT2 | 0xD0 | Table 89 |
| IOCON_PIO1_4 | R/W | 0x094 | I/O configuration for pin PIO1_4/AD5/CT32B1_MAT3 | 0xD0 | Table 90 |
| IOCON_PIO1_11 | R/W | 0x098 | I/O configuration for pin PIO1_11/AD7 | 0xD0 | Table 91 |
| IOCON_PIO3_2 | R/W | 0x09C | I/O configuration for pin PIO3_2/$\overline{\text{DCD}}$ | 0xD0 | Table 92 |
| IOCON_PIO1_5 | R/W | 0x0A0 | I/O configuration for pin PIO1_5/$\overline{\text{RTS}}$/CT32B0_CAP0 | 0xD0 | Table 93 |
| IOCON_PIO1_6 | R/W | 0x0A4 | I/O configuration for pin PIO1_6/RXD/CT32B0_MAT0 | 0xD0 | Table 94 |
| IOCON_PIO1_7 | R/W | 0x0A8 | I/O configuration for pin PIO1_7/TXD/CT32B0_MAT1 | 0xD0 | Table 95 |
| IOCON_PIO3_3 | R/W | 0x0AC | I/O configuration for pin PIO3_3/$\overline{\text{RI}}$ | 0xD0 | Table 96 |
| IOCON_SCK_LOC | R/W | 0x0B0 | SCK pin location select register | 0x00 | Table 97 |
| IOCON_DSR_LOC | R/W | 0x0B4 | $\overline{\text{DSR}}$ pin location select register | 0x00 | Table 98 |
| IOCON_DCD_LOC | R/W | 0x0B8 | $\overline{\text{DCD}}$ pin location select register | 0x00 | Table 99 |
| IOCON_RI_LOC | R/W | 0x0BC | $\overline{\text{RI}}$ pin location register | 0x00 | Table 100 |

**Table 54. I/O configuration registers ordered by port number**

| Port pin | Register name | LPC1111/12/13/14 | LPC1114 | LPC1113/14 | LPC11C12/C14 | LPC11C22/C24 | Reference |
|---|---|---|---|---|---|---|---|
| | | HVQFN33 | PLCC44 | LQFP48 | LQFP48 | LQFP48 | |
| PIO0_0 | IOCON_RESET_PIO0_0 | yes | yes | yes | yes | yes | Table 57 |
| PIO0_1 | IOCON_PIO0_1 | yes | yes | yes | yes | yes | Table 55 |
| PIO0_2 | IOCON_PIO0_2 | yes | yes | yes | yes | yes | Table 60 |
| PIO0_3 | IOCON_PIO0_3 | yes | yes | yes | yes | yes | Table 64 |
| PIO0_4 | IOCON_PIO0_4 | yes | yes | yes | yes | yes | Table 65 |
| PIO0_5 | IOCON_PIO0_5 | yes | yes | yes | yes | yes | Table 66 |
| PIO0_6 | IOCON_PIO0_6 | yes | yes | yes | yes | yes | Table 72 |
| PIO0_7 | IOCON_PIO0_7 | yes | yes | yes | yes | yes | Table 73 |
| PIO0_8 | IOCON_PIO0_8 | yes | yes | yes | yes | yes | Table 77 |
| PIO0_9 | IOCON_PIO0_9 | yes | yes | yes | yes | yes | Table 78 |
| PIO0_10 | IOCON_SWCLK_PIO0_10 | yes | yes | yes | yes | yes | Table 79 |

**Table 54. I/O configuration registers ordered by port number**

| Port pin | Register name | LPC1111/12/13/14 | LPC1114 | LPC1113/14 | LPC11C12/C14 | LPC11C22/C24 | Reference |
|---|---|---|---|---|---|---|---|
| | | HVQFN33 | PLCC44 | LQFP48 | LQFP48 | LQFP48 | |
| PIO0_11 | IOCON_R_PIO0_11 | yes | yes | yes | yes | yes | Table 82 |
| PIO1_0 | IOCON_R_PIO1_0 | yes | yes | yes | yes | yes | Table 83 |
| PIO1_1 | IOCON_R_PIO1_1 | yes | yes | yes | yes | yes | Table 84 |
| PIO1_2 | IOCON_R_PIO1_2 | yes | yes | yes | yes | yes | Table 85 |
| PIO1_3 | IOCON_SWDIO_PIO1_3 | yes | yes | yes | yes | yes | Table 89 |
| PIO1_4 | IOCON_PIO1_4 | yes | yes | yes | yes | yes | Table 90 |
| PIO1_5 | IOCON_PIO1_5 | yes | yes | yes | yes | yes | Table 93 |
| PIO1_6 | IOCON_PIO1_6 | yes | yes | yes | yes | yes | Table 94 |
| PIO1_7 | IOCON_PIO1_7 | yes | yes | yes | yes | yes | Table 95 |
| PIO1_8 | IOCON_PIO1_8 | yes | yes | yes | yes | yes | Table 59 |
| PIO1_9 | IOCON_PIO1_9 | yes | yes | yes | yes | no | Table 67 |
| PIO1_10 | IOCON_PIO1_10 | yes | yes | yes | yes | yes | Table 80 |
| PIO1_11 | IOCON_PIO1_11 | yes | yes | yes | yes | yes | Table 91 |
| PIO2_0 | IOCON_PIO2_0 | yes | yes | yes | yes | yes | Table 56 |
| PIO2_1 | IOCON_PIO2_1 | no | yes | yes | yes | yes | Table 63 |
| PIO2_2 | IOCON_PIO2_2 | no | yes | yes | yes | yes | Table 76 |
| PIO2_3 | IOCON_PIO2_3 | no | yes | yes | yes | yes | Table 88 |
| PIO2_4 | IOCON_PIO2_4 | no | yes | yes | yes | no | Table 69 |
| PIO2_5 | IOCON_PIO2_5 | no | yes | yes | yes | no | Table 70 |
| PIO2_6 | IOCON_PIO2_6 | no | yes | yes | yes | yes | Table 55 |
| PIO2_7 | IOCON_PIO2_7 | no | yes | yes | yes | yes | Table 61 |
| PIO2_8 | IOCON_PIO2_8 | no | yes | yes | yes | yes | Table 62 |
| PIO2_9 | IOCON_PIO2_9 | no | yes | yes | yes | no | Table 74 |
| PIO2_10 | IOCON_PIO2_10 | no | yes | yes | yes | yes | Table 75 |
| PIO2_11 | IOCON_PIO2_11 | no | yes | yes | yes | yes | Table 81 |
| PIO3_0 | IOCON_PIO3_0 | no | no | yes | yes | yes | Table 86 |
| PIO3_1 | IOCON_PIO3_1 | no | no | yes | yes | yes | Table 87 |
| PIO3_2 | IOCON_PIO3_2 | yes | no | yes | yes | yes | Table 92 |
| PIO3_3 | IOCON_PIO3_3 | no | no | yes | yes | yes | Table 96 |
| PIO3_4 | IOCON_PIO3_4 | yes | yes | yes | no | no | Table 68 |
| PIO3_5 | IOCON_PIO3_5 | yes | yes | yes | no | no | Table 71 |
| - | IOCON_SCK_LOC | yes (SCKLOC = 01 reserved) | yes | yes | yes | yes | Table 97 |
| - | IOCON_DSR_LOC | no | no | yes | yes | yes | Table 98 |
| - | IOCON_DCD_LOC | no | no | yes | yes | yes | Table 99 |
| - | IOCON_RI_LOC | no | no | yes | yes | yes | Table 100 |

### 7.4.1 IOCON_PIO2_6

Table 55. IOCON_PIO2_6 register (IOCON_PIO2_6, address 0x4004 4000) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_6. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.2 IOCON_PIO2_0

Table 56. IOCON_PIO2_0 register (IOCON_PIO2_0, address 0x4004 4008) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_0. | |
| | | 0x1 | Select function $\overline{\text{DTR}}$. | |
| | | 0x2 | Select function SSEL1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.3 IOCON_PIO_RESET_PIO0_0

Table 57. IOCON_RESET_PIO0_0 register (IOCON_RESET_PIO0_0, address 0x4004 400C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function $\overline{\text{RESET}}$. | |
| | | 0x1 | Selects function PIO0_0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.4 IOCON_PIO0_1

Table 58. IOCON_PIO0_1 register (IOCON_PIO0_1, address 0x4004 4010) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_1. | |
| | | 0x1 | Selects function CLKOUT. | |
| | | 0x2 | Selects function CT32B0_MAT2. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.5 IOCON_PIO1_8

Table 59. IOCON_PIO1_8 register (IOCON_PIO1_8, address 0x4004 4014) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_8. | |
| | | 0x1 | Selects function CT16B1_CAP0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.6 IOCON_PIO0_2

Table 60. IOCON_PIO0_2 register (IOCON_PIO0_2, address 0x4004 401C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_2. | |
| | | 0x1 | Selects function SSEL0. | |
| | | 0x2 | Selects function CT16B0_CAP0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.7 IOCON_PIO2_7

Table 61. IOCON_PIO2_7 register (IOCON_PIO2_7, address 0x4004 4020) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_7. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.8 IOCON_PIO2_8

Table 62. IOCON_PIO2_8 register (IOCON_PIO2_8, address 0x4004 4024) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_8. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.9 IOCON_PIO2_1

Table 63. IOCON_PIO2_1 register (IOCON_PIO2_1, address 0x4004 4028) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_1. | |
| | | 0x1 | Select function $\overline{\text{DSR}}$. | |
| | | 0x2 | Select function SCK1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.10 IOCON_PIO0_3

Table 64. IOCON_PIO0_3 register (IOCON_PIO0_3, address 0x4004 402C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_3. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.11 IOCON_PIO0_4

Table 65. IOCON_PIO0_4 register (IOCON_PIO0_4, address 0x4004 4030) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_4 (open-drain pin). | |
| | | 0x1 | Selects I2C function SCL (open-drain pin). | |
| 7:3 | | - | Reserved. | 00000 |
| 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). | 00 |
| | | 0x0 | Standard mode/ Fast-mode I2C. | |
| | | 0x1 | Standard I/O functionality | |
| | | 0x2 | Fast-mode Plus I2C | |
| | | 0x3 | Reserved. | |
| 31:10 | - | - | Reserved. | - |

### 7.4.12 IOCON_PIO0_5

Table 66. IOCON_PIO0_5 register (IOCON_PIO0_5, address 0x4004 4034) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_5 (open-drain pin). | |
| | | 0x1 | Selects I2C function SDA (open-drain pin). | |
| 7:3 | | - | Reserved. | 00000 |
| 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). | 00 |
| | | 0x0 | Standard mode/ Fast-mode I2C. | |
| | | 0x1 | Standard I/O functionality | |
| | | 0x2 | Fast-mode Plus I2C | |
| | | 0x3 | Reserved. | |
| 31:10 | - | - | Reserved. | - |

### 7.4.13 IOCON_PIO1_9

Table 67. IOCON_PIO1_9 register (IOCON_PIO1_9, address 0x4004 4038) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_9. | |
| | | 0x1 | Selects function CT16B1_MAT0. | |

Table 67. IOCON_PIO1_9 register (IOCON_PIO1_9, address 0x4004 4038) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.14 IOCON_PIO3_4

Table 68. IOCON_PIO3_4 register (IOCON_PIO3_4, address 0x4004 403C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_4. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.15 IOCON_PIO2_4

Table 69. IOCON_PIO2_4 register (IOCON_PIO2_4, address 0x4004 4040) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_4. | |

Table 69. IOCON_PIO2_4 register (IOCON_PIO2_4, address 0x4004 4040) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.16 IOCON_PIO2_5

Table 70. IOCON_PIO2_5 register (IOCON_PIO2_5, address 0x4004 4044) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_5. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.17 IOCON_PIO3_5

Table 71. IOCON_PIO3_5 register (IOCON_PIO3_5, address 0x4004 4048) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_5. | |

Table 71. IOCON_PIO3_5 register (IOCON_PIO3_5, address 0x4004 4048) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.18 IOCON_PIO0_6

Table 72. IOCON_PIO0_6 register (IOCON_PIO0_6, address 0x4004 404C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_6. | |
| | | 0x1 | Reserved. | |
| | | 0x2 | Selects function SCK0 (only if pin PIO0_6/SCK0 selected in Table 97). | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.19 IOCON_PIO0_7

Table 73. IOCON_PIO0_7 register (IOCON_PIO0_7, address 0x4004 4050) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_7. | |
| | | 0x1 | Select function $\overline{\text{CTS}}$. | |

Table 73. IOCON_PIO0_7 register (IOCON_PIO0_7, address 0x4004 4050) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.20 IOCON_PIO2_9

Table 74. IOCON_PIO2_9 register (IOCON_PIO2_9, address 0x4004 4054) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_9. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.21 IOCON_PIO2_10

Table 75. IOCON_PIO2_10 register (IOCON_PIO2_10, address 0x4004 4058) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_10. | |

Table 75. IOCON_PIO2_10 register (IOCON_PIO2_10, address 0x4004 4058) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.22 IOCON_PIO2_2

Table 76. IOCON_PIO2_2 register (IOCON_PIO2_2, address 0x4004 405C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_2. | |
| | | 0x1 | Select function $\overline{\text{DCD}}$. | |
| | | 0x2 | Select function MISO1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.23 IOCON_PIO0_8

Table 77. IOCON_PIO0_8 register (IOCON_PIO0_8, address 0x4004 4060) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_8. | |
| | | 0x1 | Selects function MISO0. | |
| | | 0x2 | Selects function CT16B0_MAT0. | |

Table 77. IOCON_PIO0_8 register (IOCON_PIO0_8, address 0x4004 4060) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.24 IOCON_PIO0_9

Table 78. IOCON_PIO0_9 register (IOCON_PIO0_9, address 0x4004 4064) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO0_9. | |
| | | 0x1 | Selects function MOSI0. | |
| | | 0x2 | Selects function CT16B0_MAT1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.25 IOCON_SWCLK_PIO0_10

Table 79. IOCON_SWCLK_PIO0_10 register (IOCON_SWCLK_PIO0_10, address 0x4004 4068) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function SWCLK. | |
| | | 0x1 | Selects function PIO0_10. | |
| | | 0x2 | Selects function SCK0 (only if pin SWCLK/PIO0_10/SCK0/CT16B0_MAT2 selected in Table 97). | |
| | | 0x3 | Selects function CT16B0_MAT2. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.26 IOCON_PIO1_10

Table 80. IOCON_PIO1_10 register (IOCON_PIO1_10, address 0x4004 406C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_10. | |
| | | 0x1 | Selects function AD6. | |
| | | 0x2 | Selects function CT16B1_MAT1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |

Table 80. IOCON_PIO1_10 register (IOCON_PIO1_10, address 0x4004 406C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7 | ADMODE | | Selects Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

## 7.4.27 IOCON_PIO2_11

Table 81. IOCON_PIO2_11 register (IOCON_PIO2_11, address 0x4004 4070) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_11. | |
| | | 0x1 | Select function SCK0 (only if pin PIO2_11/SCK0 selected in Table 97). | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.28 IOCON_R_PIO0_11

Table 82. IOCON_R_PIO0_11 register (IOCON_R_PIO0_11, address 0x4004 4074) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function R. This function is reserved. Select one of the alternate functions below. | |
| | | 0x1 | Selects function PIO0_11. | |
| | | 0x2 | Selects function AD0. | |
| | | 0x3 | Selects function CT32B0_MAT3. | |

**Table 82. IOCON_R_PIO0_11 register (IOCON_R_PIO0_11, address 0x4004 4074) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Selects Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

## 7.4.29 IOCON_R_PIO1_0

**Table 83. IOCON_R_PIO1_0 register (IOCON_R_PIO1_0, address 0x4004 4078) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function R. This function is reserved. Select one of the alternate functions below. | |
| | | 0x1 | Selects function PIO1_0. | |
| | | 0x2 | Selects function AD1. | |
| | | 0x3 | Selects function CT32B1_CAP0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Selects Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

### 7.4.30 IOCON_R_PIO1_1

Table 84. IOCON_R_PIO1_1 register (IOCON_R_PIO1_1, address 0x4004 407C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function R. This function is reserved. Select one of the alternate functions below. | |
| | | 0x1 | Selects function PIO1_1. | |
| | | 0x2 | Selects function AD2. | |
| | | 0x3 | Selects function CT32B1_MAT0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Selects Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

### 7.4.31 IOCON_R_PIO1_2

Table 85. IOCON_R_PIO1_2 register (IOCON_R_PIO1_2, address 0x4004 4080) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function R. This function is reserved. Select one of the alternate functions below. | |
| | | 0x1 | Selects function PIO1_2. | |
| | | 0x2 | Selects function AD3. | |
| | | 0x3 | Selects function CT32B1_MAT1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |

**Table 85. IOCON_R_PIO1_2 register (IOCON_R_PIO1_2, address 0x4004 4080) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Selects Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

## 7.4.32 IOCON_PIO3_0

**Table 86. IOCON_PIO3_0 register (IOCON_PIO3_0, address 0x4004 4084) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_0. | |
| | | 0x1 | Selects function $\overline{\text{DTR}}$. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.33 IOCON_PIO3_1

**Table 87. IOCON_PIO3_1 register (IOCON_PIO3_1, address 0x4004 4088) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_1. | |
| | | 0x1 | Selects function $\overline{\text{DSR}}$. | |

Table 87. IOCON_PIO3_1 register (IOCON_PIO3_1, address 0x4004 4088) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.34 IOCON_PIO2_3

Table 88. IOCON_PIO2_3 register (IOCON_PIO2_3, address 0x4004 408C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO2_3. | |
| | | 0x1 | Selects function $\overline{RI}$. | |
| | | 0x2 | Selects function MOSI1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

### 7.4.35 IOCON_SWDIO_PIO1_3

**Table 89. IOCON_SWDIO_PIO1_3 register (IOCON_SWDIO_PIO1_3, address 0x4004 4090) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function SWDIO. | |
| | | 0x1 | Selects function PIO1_3. | |
| | | 0x2 | Selects function AD4. | |
| | | 0x3 | Selects function CT32B1_MAT2. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Select Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

### 7.4.36 IOCON_PIO1_4

**Table 90. IOCON_PIO1_4 register (IOCON_PIO1_4, address 0x4004 4094) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. This pin functions as WAKEUP pin if the LPC111x/LPC11Cxx is in Deep power-down mode regardless of the value of FUNC. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_4. | |
| | | 0x1 | Selects function AD5. | |
| | | 0x2 | Selects function CT32B1_MAT3. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |

Table 90. IOCON_PIO1_4 register (IOCON_PIO1_4, address 0x4004 4094) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Select Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

## 7.4.37 IOCON_PIO1_11

Table 91. IOCON_PIO1_11 register (IOCON_PIO1_11 address 0x4004 4098) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_11. | |
| | | 0x1 | Selects function AD7. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 6 | - | - | Reserved. | 1 |
| 7 | ADMODE | | Select Analog/Digital mode. | 1 |
| | | 0 | Analog input mode. | |
| | | 1 | Digital functional mode. | |
| 31:8 | - | - | Reserved. | - |

## 7.4.38 IOCON_PIO3_2

Table 92. IOCON_PIO3_2 register (IOCON_PIO3_2, address 0x4004 409C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_2. | |
| | | 0x1 | Selects function $\overline{\text{DCD}}$. | |

Table 92. IOCON_PIO3_2 register (IOCON_PIO3_2, address 0x4004 409C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.39 IOCON_PIO1_5

Table 93. IOCON_PIO1_5 register (IOCON_PIO1_5, address 0x4004 40A0) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_5. | |
| | | 0x1 | Selects function $\overline{\text{RTS}}$. | |
| | | 0x2 | Selects function CT32B0_CAP0. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.40 IOCON_PIO1_6

Table 94. IOCON_PIO1_6 register (IOCON_PIO1_6, address 0x4004 40A4) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_6. | |
| | | 0x1 | Selects function RXD. | |
| | | 0x2 | Selects function CT32B0_MAT0. | |

Table 94. IOCON_PIO1_6 register (IOCON_PIO1_6, address 0x4004 40A4) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.41 IOCON_PIO1_7

Table 95. IOCON_PIO1_7 register (IOCON_PIO1_7, address 0x4004 40A8) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO1_7. | |
| | | 0x1 | Selects function TXD. | |
| | | 0x2 | Selects function CT32B0_MAT1. | |
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.42 IOCON_PIO3_3

Table 96. IOCON_PIO3_3 register (IOCON_PIO3_3, address 0x4004 40AC) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2:0 | FUNC | | Selects pin function. All other values are reserved. | 000 |
| | | 0x0 | Selects function PIO3_3. | |
| | | 0x1 | Selects function $\overline{RI}$. | |

Table 96. IOCON_PIO3_3 register (IOCON_PIO3_3, address 0x4004 40AC) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4:3 | MODE | | Selects function mode (on-chip pull-up/pull-down resistor control). | 10 |
| | | 0x0 | Inactive (no pull-down/pull-up resistor enabled). | |
| | | 0x1 | Pull-down resistor enabled. | |
| | | 0x2 | Pull-up resistor enabled. | |
| | | 0x3 | Repeater mode. | |
| 5 | HYS | | Hysteresis. | 0 |
| | | 0 | Disable. | |
| | | 1 | Enable. | |
| 7:6 | - | - | Reserved. | 11 |
| 31:8 | - | - | Reserved. | 0 |

## 7.4.43 IOCON_SCK_LOC

Table 97. IOCON SCK location register (IOCON_SCK_LOC, address 0x4004 40B0) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | SCKLOC | | Selects pin location for SCK0 pin. | 00 |
| | | 0x0 | Selects SCK0 function in pin location SWCLK/PIO0_10/SCK0/CT16B0_MAT2 (see Table 79). | |
| | | 0x1 | Selects SCK0 function in pin location PIO2_11/SCK0 (see Table 81. | |
| | | 0x2 | Selects SCK0 function in pin location PIO0_6/SCK0 (see Table 72). | |
| | | 0x3 | Reserved. | |
| 31:2 | - | - | Reserved. | - |

## 7.4.44 IOCON_DSR_LOC

Table 98. IOCON $\overline{\text{DSR}}$ location register (IOCON_DSR_LOC, address 0x4004 40B4) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | DSRLOC | | Selects pin location for DSR0 pin. | 00 |
| | | 0x0 | Selects $\overline{\text{DSR}}$ function in pin location PIO2_1/$\overline{\text{DSR}}$/SCK1. | |
| | | 0x1 | Selects $\overline{\text{DSR}}$ function in pin location PIO3_1/$\overline{\text{DSR}}$. | |
| | | 0x2 | Reserved. | |
| | | 0x3 | Reserved. | |
| 31:2 | - | - | Reserved. | - |

### 7.4.45 IOCON_DCD_LOC

Table 99. IOCON $\overline{\text{DCD}}$ location register (IOCON_DCD_LOC, address 0x4004 40B8) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | DCDLOC | | Selects pin location for DCD pin. | 00 |
| | | 0x0 | Selects $\overline{\text{DCD}}$ function in pin location PIO2_2/$\overline{\text{DCD}}$/MISO1. | |
| | | 0x1 | Selects $\overline{\text{DCD}}$ function in pin location PIO3_2/$\overline{\text{DCD}}$. | |
| | | 0x2 | Reserved. | |
| | | 0x3 | Reserved. | |
| 31:2 | - | - | Reserved. | - |

### 7.4.46 IOCON_RI_LOC

Table 100. IOCON $\overline{\text{RI}}$ location register (IOCON_RI_LOC, address 0x4004 40BC) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | RILOC | | Selects pin location for RI pin. | 00 |
| | | 0x0 | Selects $\overline{\text{RI}}$ function in pin location PIO2_3/$\overline{\text{RI}}$/MOSI1. | |
| | | 0x1 | Selects $\overline{\text{RI}}$ function in pin location PIO3_3/$\overline{\text{RI}}$. | |
| | | 0x2 | Reserved. | |
| | | 0x3 | Reserved. | |
| 31:2 | - | - | Reserved. | - |

## 8.1 How to read this chapter

The LPC111x are available in three packages: LQFP48 (LPC1113, LPC1114), PLCC44 (LPC1114), and HVQFN33 (LPC1111, LPC1112, LPC1113, LPC1114).

The LPC11Cxx parts are available in a LQFP48 package.

**Table 101. LPC111x/LPC11Cxx pin configurations**

| Part | | LQFP48 | PLCC44 | HVQFN33 |
|---|---|---|---|---|
| LPC1111 | Pin configuration | - | - | Figure 11 |
| | Pin description | - | - | Table 104 |
| LPC1112 | Pin configuration | - | - | Figure 11 |
| | Pin description | - | - | Table 104 |
| LPC1113 | Pin configuration | Figure 9 | - | Figure 11 |
| | Pin description | Table 102 | - | Table 104 |
| LPC1114 | Pin configuration | Figure 9 | Figure 10 | Figure 11 |
| | Pin description | Table 102 | Table 103 | Table 104 |
| LPC11C12 | Pin configuration | Figure 12 | - | - |
| | Pin description | Table 102 | - | - |
| LPC11C14 | Pin configuration | Figure 12 | - | - |
| | Pin description | Table 102 | - | - |
| LPC11C22 | Pin configuration | Figure 13 | - | - |
| | Pin description | Table 105 | - | - |
| LPC11C24 | Pin configuration | Figure 13 | - | - |
| | Pin description | Table 105 | - | - |

## 8.2 LPC111x Pin configuration

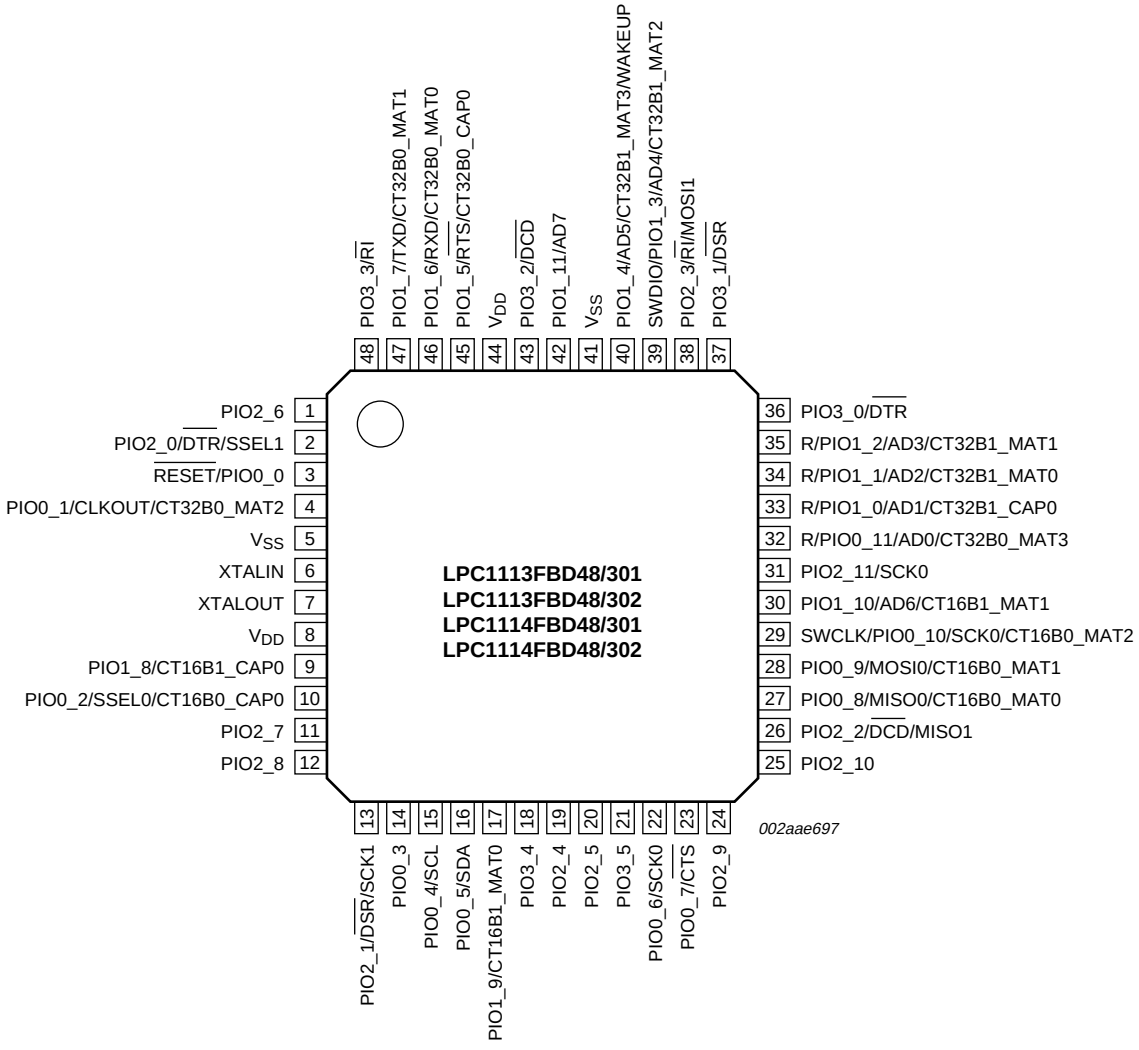



**Fig 9. Pin configuration LQFP48 package**

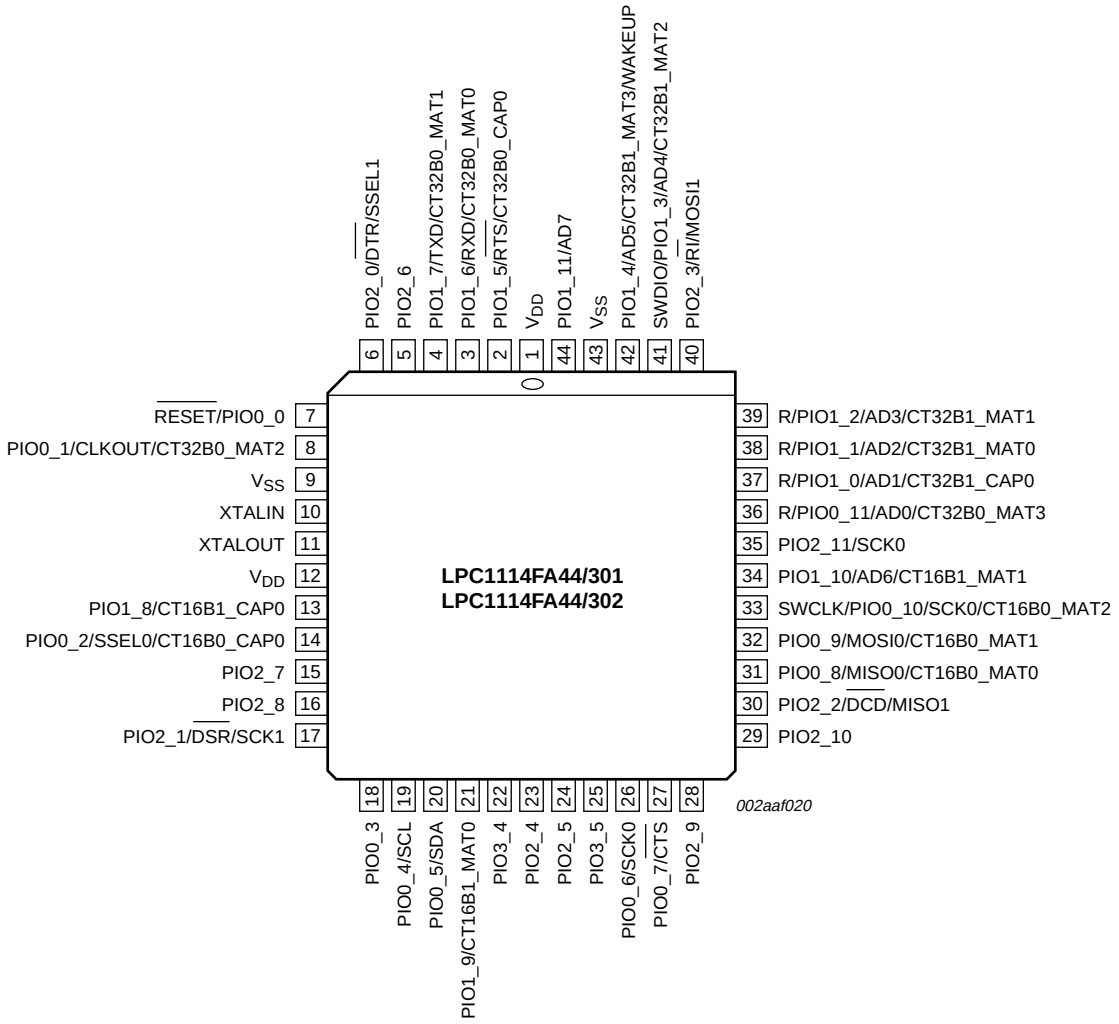



**Fig 10. Pin configuration PLCC44 package**

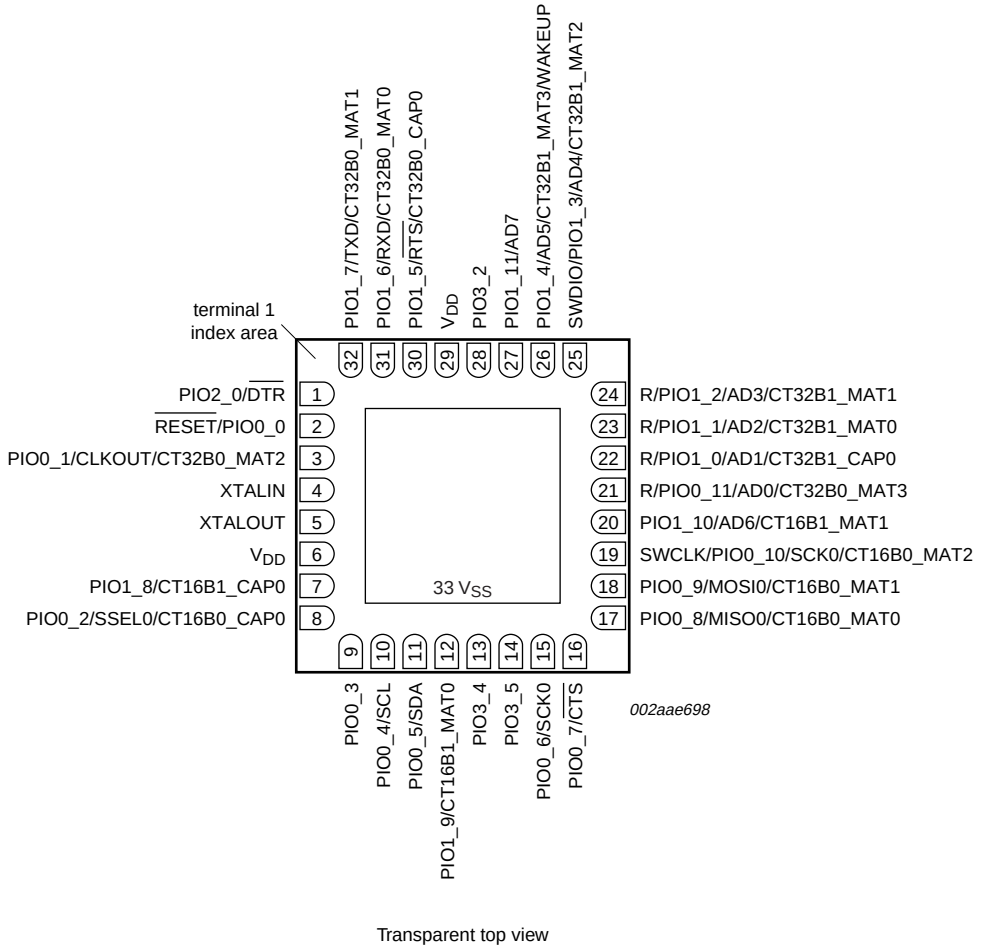



**Fig 11. Pin configuration HVQFN 33 package**

## 8.3 LPC11Cxx Pin configuration


**Fig 12. Pin configuration LQFP48 package**

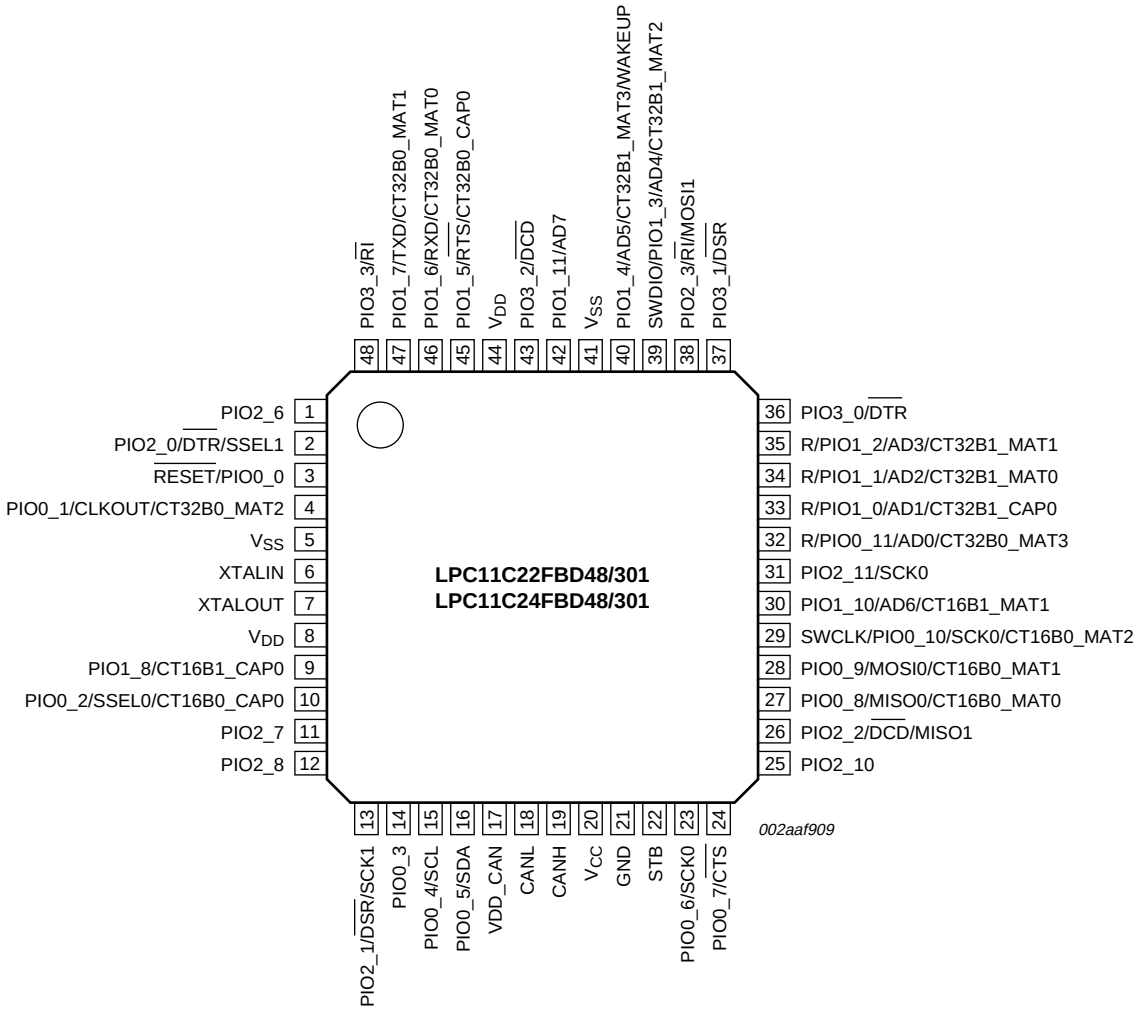



**Fig 13. Pin configuration (LPC11C22/C24)**

## 8.4 LPC111x/LPC11Cxx Pin description

**Table 102. LPC1113/14 and LPC11C12/C14 pin description table (LQFP48 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_0 to PIO0_11 | | I/O | **Port 0 —** Port 0 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 0 pins depends on the function selected through the IOCONFIG register block. |
| RESET/PIO0_0 | 3[1][2] | I | **RESET —** External reset input: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. |
| | | I/O | **PIO0_0 —** General purpose digital input/output pin. |
| PIO0_1/CLKOUT/ CT32B0_MAT2 | 4[3][2] | I/O | **PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the flash ISP command handler via UART (if PIO0_3 is HIGH) or via C_CAN (if PIO0_3 is LOW). |
| | | O | **CLKOUT —** Clockout pin. |
| | | O | **CT32B0_MAT2 —** Match output 2 for 32-bit timer 0. |

**Table 102. LPC1113/14 and LPC11C12/C14 pin description table (LQFP48 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_2/SSEL0/ CT16B0_CAP0 | 10[3][2] | I/O | **PIO0_2 —** General purpose digital input/output pin. |
| | | O | **SSEL0 —** Slave Select for SPI0. |
| | | I | **CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0. |
| PIO0_3 | 14[3][2] | I/O | **PIO0_3 —** General purpose digital input/output pin. This pin is monitored during reset: Together with a LOW level on pin PIO0_1, a LOW level starts the flash ISP command handler via C_CAN and a HIGH level starts the flash ISP command handler via UART. |
| PIO0_4/SCL | 15[4][2] | I/O | **PIO0_4 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SCL —** $I^2C$-bus, open-drain clock input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_5/SDA | 16[4][2] | I/O | **PIO0_5 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SDA —** $I^2C$-bus, open-drain data input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_6/SCK0 | 22[3][2] | I/O | **PIO0_6 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO0_7/$\overline{\text{CTS}}$ | 23[3][2] | I/O | **PIO0_7 —** General purpose digital input/output pin (high-current output driver). |
| | | I | **$\overline{\text{CTS}}$ —** Clear To Send input for UART. |
| PIO0_8/MISO0/ CT16B0_MAT0 | 27[3][2] | I/O | **PIO0_8 —** General purpose digital input/output pin. |
| | | I/O | **MISO0 —** Master In Slave Out for SPI0. |
| | | O | **CT16B0_MAT0 —** Match output 0 for 16-bit timer 0. |
| PIO0_9/MOSI0/ CT16B0_MAT1 | 28[3][2] | I/O | **PIO0_9 —** General purpose digital input/output pin. |
| | | I/O | **MOSI0 —** Master Out Slave In for SPI0. |
| | | O | **CT16B0_MAT1 —** Match output 1 for 16-bit timer 0. |
| SWCLK/PIO0_10/ SCK0/CT16B0_MAT2 | 29[3][2] | I | **SWCLK —** Serial wire clock. |
| | | I/O | **PIO0_10 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| | | O | **CT16B0_MAT2 —** Match output 2 for 16-bit timer 0. |
| R/PIO0_11/ AD0/CT32B0_MAT3 | 32[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO0_11 —** General purpose digital input/output pin. |
| | | I | **AD0 —** A/D converter, input 0. |
| | | O | **CT32B0_MAT3 —** Match output 3 for 32-bit timer 0. |
| PIO1_0 to PIO1_11 | | I/O | **Port 1 —** Port 1 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 1 pins depends on the function selected through the IOCONFIG register block. |
| R/PIO1_0/ AD1/CT32B1_CAP0 | 33[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_0 —** General purpose digital input/output pin. |
| | | I | **AD1 —** A/D converter, input 1. |
| | | I | **CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1. |
| R/PIO1_1/ AD2/CT32B1_MAT0 | 34[5] | O | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_1 —** General purpose digital input/output pin. |
| | | I | **AD2 —** A/D converter, input 2. |
| | | O | **CT32B1_MAT0 —** Match output 0 for 32-bit timer 1. |

**Table 102. LPC1113/14 and LPC11C12/C14 pin description table (LQFP48 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| R/PIO1_2/ AD3/CT32B1_MAT1 | 35[5] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_2 —** General purpose digital input/output pin. |
| | | I | **AD3 —** A/D converter, input 3. |
| | | O | **CT32B1_MAT1 —** Match output 1 for 32-bit timer 1. |
| SWDIO/PIO1_3/AD4/ CT32B1_MAT2 | 39[5] | I/O | **SWDIO —** Serial wire debug input/output. |
| | | I/O | **PIO1_3 —** General purpose digital input/output pin. |
| | | I | **AD4 —** A/D converter, input 4. |
| | | O | **CT32B1_MAT2 —** Match output 2 for 32-bit timer 1. |
| PIO1_4/AD5/ CT32B1_MAT3/WAKEUP | 40[5] | I/O | **PIO1_4 —** General purpose digital input/output pin. |
| | | I | **AD5 —** A/D converter, input 5. |
| | | O | **CT32B1_MAT3 —** Match output 3 for 32-bit timer 1. |
| | | I | **WAKEUP —** Deep power-down mode wake-up pin. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. |
| PIO1_5/$\overline{\text{RTS}}$/ CT32B0_CAP0 | 45[3] | I/O | **PIO1_5 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{RTS}}$ —** Request To Send output for UART. |
| | | I | **CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0. |
| PIO1_6/RXD/ CT32B0_MAT0 | 46[3] | I/O | **PIO1_6 —** General purpose digital input/output pin. |
| | | I | **RXD —** Receiver input for UART. |
| | | O | **CT32B0_MAT0 —** Match output 0 for 32-bit timer 0. |
| PIO1_7/TXD/ CT32B0_MAT1 | 47[3] | I/O | **PIO1_7 —** General purpose digital input/output pin. |
| | | O | **TXD —** Transmitter output for UART. |
| | | O | **CT32B0_MAT1 —** Match output 1 for 32-bit timer 0. |
| PIO1_8/CT16B1_CAP0 | 9[3] | I/O | **PIO1_8 —** General purpose digital input/output pin. |
| | | I | **CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1. |
| PIO1_9/CT16B1_MAT0 | 17[3] | I/O | **PIO1_9 —** General purpose digital input/output pin. |
| | | O | **CT16B1_MAT0 —** Match output 0 for 16-bit timer 1. |
| PIO1_10/AD6/ CT16B1_MAT1 | 30[5] | I/O | **PIO1_10 —** General purpose digital input/output pin. |
| | | I | **AD6 —** A/D converter, input 6. |
| | | O | **CT16B1_MAT1 —** Match output 1 for 16-bit timer 1. |
| PIO1_11/AD7 | 42[5] | I/O | **PIO1_11 —** General purpose digital input/output pin. |
| | | I | **AD7 —** A/D converter, input 7. |
| PIO2_0 to PIO2_11 | | I/O | **Port 2 —** Port 2 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 2 pins depends on the function selected through the IOCONFIG register block. |
| PIO2_0/$\overline{\text{DTR}}$/SSEL1 | 2[3] | I/O | **PIO2_0 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| | | O | **SSEL1 —** Slave Select for SPI1. |
| PIO2_1/$\overline{\text{DSR}}$/SCK1 | 13[3] | I/O | **PIO2_1 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DSR}}$ —** Data Set Ready input for UART. |
| | | I/O | **SCK1 —** Serial clock for SPI1. |

**Table 102. LPC1113/14 and LPC11C12/C14 pin description table (LQFP48 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO2_2/$\overline{\text{DCD}}$/MISO1 | 26[3] | I/O | **PIO2_2 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DCD}}$ —** Data Carrier Detect input for UART. |
| | | I/O | **MISO1 —** Master In Slave Out for SPI1. |
| PIO2_3/$\overline{\text{RI}}$/MOSI1 | 38[3] | I/O | **PIO2_3 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{RI}}$ —** Ring Indicator input for UART. |
| | | I/O | **MOSI1 —** Master Out Slave In for SPI1. |
| PIO2_4 | 19[3] | I/O | **PIO2_4 —** General purpose digital input/output pin. (LPC1113/14 only). |
| PIO2_4 | 18[3] | I/O | **PIO2_4 —** General purpose digital input/output pin. (LPC11C12/C14 only). |
| PIO2_5 | 20[3] | I/O | **PIO2_5 —** General purpose digital input/output pin. LPC1113/14 only). |
| PIO2_5 | 21[3] | I/O | **PIO2_5 —** General purpose digital input/output pin. (LPC11C12/C14 only). |
| PIO2_6 | 1[3] | I/O | **PIO2_6 —** General purpose digital input/output pin. |
| PIO2_7 | 11[3] | I/O | **PIO2_7 —** General purpose digital input/output pin. |
| PIO2_8 | 12[3] | I/O | **PIO2_8 —** General purpose digital input/output pin. |
| PIO2_9 | 24[3] | I/O | **PIO2_9 —** General purpose digital input/output pin. |
| PIO2_10 | 25[3] | I/O | **PIO2_10 —** General purpose digital input/output pin. |
| PIO2_11/SCK0 | 31[3] | I/O | **PIO2_11 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO3_0 to PIO3_5 | | I/O | **Port 3 —** Port 3 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 3 pins depends on the function selected through the IOCONFIG register block. Pins PIO3_6 to PIO3_11 are not available. |
| PIO3_0/$\overline{\text{DTR}}$ | 36[3] | I/O | **PIO3_0 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| PIO3_1/$\overline{\text{DSR}}$ | 37[3] | I/O | **PIO3_1 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DSR}}$ —** Data Set Ready input for UART. |
| PIO3_2/$\overline{\text{DCD}}$ | 43[3] | I/O | **PIO3_2 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DCD}}$ —** Data Carrier Detect input for UART. |
| PIO3_3/$\overline{\text{RI}}$ | 48[3] | I/O | **PIO3_3 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{RI}}$ —** Ring Indicator input for UART. |
| PIO3_4 | 18[3] | I/O | **PIO3_4 —** General purpose digital input/output pin. (LPC1113/14 only). |
| PIO3_5 | 21[3] | I/O | **PIO3_5 —** General purpose digital input/output pin. (LPC1113/14 only). |
| CAN_RXD | 19[6] | I | **CAN_RXD —** C_CAN receive data input. (LPC11C12/14 only). |
| CAN_TXD | 20[6] | O | **CAN_TXD —** C_CAN transmit data output. (LPC11C12/14 only). |
| $V_{DD}$ | 8; 44 | I | 3.3 V supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage. |
| XTALIN | 6[7] | I | Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V. |
| XTALOUT | 7[7] | O | Output from the oscillator amplifier. |
| $V_{SS}$ | 5; 41 | I | Ground. |

[1] $\overline{\text{RESET}}$ 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.

[2] Serves as Deep-sleep wake-up input pin to the start logic independently of selected pin function (see the *LPC111x/11C1x user manual*).

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

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

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

[6] 5 V tolerant digital I/O pad without pull-up/pull-down resistors.

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

**Table 103. LPC1114 pin description table (PLCC44 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_0 to PIO0_11 | | I/O | **Port 0 —** Port 0 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 0 pins depends on the function selected through the IOCONFIG register block. |
| $\overline{\text{RESET}}$/PIO0_0 | 7[1][2] | I | **$\overline{\text{RESET}}$ —** External reset input: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. |
| | | I/O | **PIO0_0 —** General purpose digital input/output pin. |
| PIO0_1/CLKOUT/ CT32B0_MAT2 | 8[3][2] | I/O | **PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler. |
| | | O | **CLKOUT —** Clockout pin. |
| | | O | **CT32B0_MAT2 —** Match output 2 for 32-bit timer 0. |
| PIO0_2/SSEL0/ CT16B0_CAP0 | 14[3][2] | I/O | **PIO0_2 —** General purpose digital input/output pin. |
| | | O | **SSEL0 —** Slave Select for SPI0. |
| | | I | **CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0. |
| PIO0_3 | 18[3][2] | I/O | **PIO0_3 —** General purpose digital input/output pin. |
| PIO0_4/SCL | 19[4][2] | I/O | **PIO0_4 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SCL —** $I^2C$-bus, open-drain clock input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_5/SDA | 20[4][2] | I/O | **PIO0_5 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SDA —** $I^2C$-bus, open-drain data input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_6/SCK0 | 26[3][2] | I/O | **PIO0_6 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO0_7/$\overline{\text{CTS}}$ | 27[3][2] | I/O | **PIO0_7 —** General purpose digital input/output pin (high-current output driver). |
| | | I | **$\overline{\text{CTS}}$ —** Clear To Send input for UART. |
| PIO0_8/MISO0/ CT16B0_MAT0 | 31[3][2] | I/O | **PIO0_8 —** General purpose digital input/output pin. |
| | | I/O | **MISO0 —** Master In Slave Out for SPI0. |
| | | O | **CT16B0_MAT0 —** Match output 0 for 16-bit timer 0. |
| PIO0_9/MOSI0/ CT16B0_MAT1 | 32[3][2] | I/O | **PIO0_9 —** General purpose digital input/output pin. |
| | | I/O | **MOSI0 —** Master Out Slave In for SPI0. |
| | | O | **CT16B0_MAT1 —** Match output 1 for 16-bit timer 0. |
| SWCLK/PIO0_10/ SCK0/CT16B0_MAT2 | 33[3][2] | I | **SWCLK —** Serial wire clock. |
| | | I/O | **PIO0_10 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| | | O | **CT16B0_MAT2 —** Match output 2 for 16-bit timer 0. |

**Table 103. LPC1114 pin description table (PLCC44 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| R/PIO0_11/ AD0/CT32B0_MAT3 | 36[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO0_11 —** General purpose digital input/output pin. |
| | | I | **AD0 —** A/D converter, input 0. |
| | | O | **CT32B0_MAT3 —** Match output 3 for 32-bit timer 0. |
| PIO1_0 to PIO1_11 | | I/O | **Port 1 —** Port 1 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 1 pins depends on the function selected through the IOCONFIG register block. |
| R/PIO1_0/ AD1/CT32B1_CAP0 | 37[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_0 —** General purpose digital input/output pin. |
| | | I | **AD1 —** A/D converter, input 1. |
| | | I | **CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1. |
| R/PIO1_1/ AD2/CT32B1_MAT0 | 38[5] | O | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_1 —** General purpose digital input/output pin. |
| | | I | **AD2 —** A/D converter, input 2. |
| | | O | **CT32B1_MAT0 —** Match output 0 for 32-bit timer 1. |
| R/PIO1_2/ AD3/CT32B1_MAT1 | 39[5] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_2 —** General purpose digital input/output pin. |
| | | I | **AD3 —** A/D converter, input 3. |
| | | O | **CT32B1_MAT1 —** Match output 1 for 32-bit timer 1. |
| SWDIO/PIO1_3/AD4/ CT32B1_MAT2 | 41[5] | I/O | **SWDIO —** Serial wire debug input/output. |
| | | I/O | **PIO1_3 —** General purpose digital input/output pin. |
| | | I | **AD4 —** A/D converter, input 4. |
| | | O | **CT32B1_MAT2 —** Match output 2 for 32-bit timer 1. |
| PIO1_4/AD5/ CT32B1_MAT3/WAKEUP | 42[5] | I/O | **PIO1_4 —** General purpose digital input/output pin. |
| | | I | **AD5 —** A/D converter, input 5. |
| | | O | **CT32B1_MAT3 —** Match output 3 for 32-bit timer 1. |
| | | I | **WAKEUP —** Deep power-down mode wake-up pin. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. |
| PIO1_5/$\overline{\text{RTS}}$/ CT32B0_CAP0 | 2[3] | I/O | **PIO1_5 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{RTS}}$ —** Request To Send output for UART. |
| | | I | **CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0. |
| PIO1_6/RXD/ CT32B0_MAT0 | 3[3] | I/O | **PIO1_6 —** General purpose digital input/output pin. |
| | | I | **RXD —** Receiver input for UART. |
| | | O | **CT32B0_MAT0 —** Match output 0 for 32-bit timer 0. |
| PIO1_7/TXD/ CT32B0_MAT1 | 4[3] | I/O | **PIO1_7 —** General purpose digital input/output pin. |
| | | O | **TXD —** Transmitter output for UART. |
| | | O | **CT32B0_MAT1 —** Match output 1 for 32-bit timer 0. |
| PIO1_8/CT16B1_CAP0 | 13[3] | I/O | **PIO1_8 —** General purpose digital input/output pin. |
| | | I | **CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1. |
| PIO1_9/CT16B1_MAT0 | 21[3] | I/O | **PIO1_9 —** General purpose digital input/output pin. |
| | | O | **CT16B1_MAT0 —** Match output 0 for 16-bit timer 1. |

**Table 103. LPC1114 pin description table (PLCC44 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO1_10/AD6/ CT16B1_MAT1 | 34[5] | I/O | **PIO1_10 —** General purpose digital input/output pin. |
| | | I | **AD6 —** A/D converter, input 6. |
| | | O | **CT16B1_MAT1 —** Match output 1 for 16-bit timer 1. |
| PIO1_11/AD7 | 44[5] | I/O | **PIO1_11 —** General purpose digital input/output pin. |
| | | I | **AD7 —** A/D converter, input 7. |
| PIO2_0 to PIO2_11 | | I/O | **Port 2 —** Port 2 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 2 pins depends on the function selected through the IOCONFIG register block. |
| PIO2_0/$\overline{\text{DTR}}$/SSEL1 | 6[3] | I/O | **PIO2_0 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| | | O | **SSEL1 —** Slave Select for SPI1. |
| PIO2_1/$\overline{\text{DSR}}$/SCK1 | 17[3] | I/O | **PIO2_1 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DSR}}$ —** Data Set Ready input for UART. |
| | | I/O | **SCK1 —** Serial clock for SPI1. |
| PIO2_2/$\overline{\text{DCD}}$/MISO1 | 30[3] | I/O | **PIO2_2 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DCD}}$ —** Data Carrier Detect input for UART. |
| | | I/O | **MISO1 —** Master In Slave Out for SPI1. |
| PIO2_3/$\overline{\text{RI}}$/MOSI1 | 40[3] | I/O | **PIO2_3 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{RI}}$ —** Ring Indicator input for UART. |
| | | I/O | **MOSI1 —** Master Out Slave In for SPI1. |
| PIO2_4 | 23[3] | I/O | **PIO2_4 —** General purpose digital input/output pin. |
| PIO2_5 | 24[3] | I/O | **PIO2_5 —** General purpose digital input/output pin. |
| PIO2_6 | 5[3] | I/O | **PIO2_6 —** General purpose digital input/output pin. |
| PIO2_7 | 15[3] | I/O | **PIO2_7 —** General purpose digital input/output pin. |
| PIO2_8 | 16[3] | I/O | **PIO2_8 —** General purpose digital input/output pin. |
| PIO2_9 | 28[3] | I/O | **PIO2_9 —** General purpose digital input/output pin. |
| PIO2_10 | 29[3] | I/O | **PIO2_10 —** General purpose digital input/output pin. |
| PIO2_11/SCK0 | 35[3] | I/O | **PIO2_11 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO3_0 to PIO3_5 | | I/O | **Port 3 —** Port 3 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 3 pins depends on the function selected through the IOCONFIG register block. Pins PIO3_0 to PIO3_3 and PIO3_6 to PIO3_11 are not available. |
| PIO3_4 | 22[3] | I/O | **PIO3_4 —** General purpose digital input/output pin. |
| PIO3_5 | 25[3] | I/O | **PIO3_5 —** General purpose digital input/output pin. |
| $V_{DD}$ | 1; 12 | I | 3.3 V supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage. |
| XTALIN | 10[7] | I | Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V. |
| XTALOUT | 11[7] | O | Output from the oscillator amplifier. |
| $V_{SS}$ | 9; 43 | I | Ground. |

[1] $\overline{\text{RESET}}$ 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.

[2] Serves as Deep-sleep wake-up input pin to the start logic independently of selected pin function (see the *LPC111x/11C1x user manual*).

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

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

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

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

**Table 104. LPC1111/12/13/14 pin description table (HVQFN33 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_0 to PIO0_11 | | I/O | **Port 0 —** Port 0 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 0 pins depends on the function selected through the IOCONFIG register block. |
| $\overline{\text{RESET}}$/PIO0_0 | 2[1][2] | I | **$\overline{\text{RESET}}$ —** External reset input: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. |
| | | I/O | **PIO0_0 —** General purpose digital input/output pin. |
| PIO0_1/CLKOUT/ CT32B0_MAT2 | 3[3][2] | I/O | **PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the ISP command handler. |
| | | O | **CLKOUT —** Clock out pin. |
| | | O | **CT32B0_MAT2 —** Match output 2 for 32-bit timer 0. |
| PIO0_2/SSEL0/ CT16B0_CAP0 | 8[3][2] | I/O | **PIO0_2 —** General purpose digital input/output pin. |
| | | O | **SSEL0 —** Slave select for SPI0. |
| | | I | **CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0. |
| PIO0_3 | 9[3][2] | I/O | **PIO0_3 —** General purpose digital input/output pin. |
| PIO0_4/SCL | 10[4][2] | I/O | **PIO0_4 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SCL —** $I^2C$-bus, open-drain clock input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_5/SDA | 11[4][2] | I/O | **PIO0_5 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SDA —** $I^2C$-bus, open-drain data input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_6/SCK0 | 15[3][2] | I/O | **PIO0_6 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO0_7/$\overline{\text{CTS}}$ | 16[3][2] | I/O | **PIO0_7 —** General purpose digital input/output pin (high-current output driver). |
| | | I | **$\overline{\text{CTS}}$ —** Clear To Send input for UART. |
| PIO0_8/MISO0/ CT16B0_MAT0 | 17[3][2] | I/O | **PIO0_8 —** General purpose digital input/output pin. |
| | | I/O | **MISO0 —** Master In Slave Out for SPI0. |
| | | O | **CT16B0_MAT0 —** Match output 0 for 16-bit timer 0. |
| PIO0_9/MOSI0/ CT16B0_MAT1 | 18[3][2] | I/O | **PIO0_9 —** General purpose digital input/output pin. |
| | | I/O | **MOSI0 —** Master Out Slave In for SPI0. |
| | | O | **CT16B0_MAT1 —** Match output 1 for 16-bit timer 0. |
| SWCLK/PIO0_10/SCK0/ CT16B0_MAT2 | 19[3][2] | I | **SWCLK —** Serial wire clock. |
| | | I/O | **PIO0_10 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| | | O | **CT16B0_MAT2 —** Match output 2 for 16-bit timer 0. |
| R/PIO0_11/AD0/ CT32B0_MAT3 | 21[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO0_11 —** General purpose digital input/output pin. |
| | | I | **AD0 —** A/D converter, input 0. |
| | | O | **CT32B0_MAT3 —** Match output 3 for 32-bit timer 0. |
| PIO1_0 to PIO1_11 | | I/O | **Port 1 —** Port 1 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 1 pins depends on the function selected through the IOCONFIG register block. |

**Table 104. LPC1111/12/13/14 pin description table (HVQFN33 package)** *...continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| R/PIO1_0/AD1/ CT32B1_CAP0 | 22[5][2] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_0 —** General purpose digital input/output pin. |
| | | I | **AD1 —** A/D converter, input 1. |
| | | I | **CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1. |
| R/PIO1_1/AD2/ CT32B1_MAT0 | 23[5] | O | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_1 —** General purpose digital input/output pin. |
| | | I | **AD2 —** A/D converter, input 2. |
| | | O | **CT32B1_MAT0 —** Match output 0 for 32-bit timer 1. |
| R/PIO1_2/AD3/ CT32B1_MAT1 | 24[5] | I | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_2 —** General purpose digital input/output pin. |
| | | I | **AD3 —** A/D converter, input 3. |
| | | O | **CT32B1_MAT1 —** Match output 1 for 32-bit timer 1. |
| SWDIO/PIO1_3/AD4/ CT32B1_MAT2 | 25[5] | I/O | **SWDIO —** Serial wire debug input/output. |
| | | I/O | **PIO1_3 —** General purpose digital input/output pin. |
| | | I | **AD4 —** A/D converter, input 4. |
| | | O | **CT32B1_MAT2 —** Match output 2 for 32-bit timer 1. |
| PIO1_4/AD5/ CT32B1_MAT3/WAKEUP | 26[5] | I/O | **PIO1_4 —** General purpose digital input/output pin. |
| | | I | **AD5 —** A/D converter, input 5. |
| | | O | **CT32B1_MAT3 —** Match output 3 for 32-bit timer 1. |
| | | I | **WAKEUP —** Deep power-down mode wake-up pin. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. |
| PIO1_5/$\overline{\text{RTS}}$/ CT32B0_CAP0 | 30[3] | I/O | **PIO1_5 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{RTS}}$ —** Request To Send output for UART. |
| | | I | **CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0. |
| PIO1_6/RXD/ CT32B0_MAT0 | 31[3] | I/O | **PIO1_6 —** General purpose digital input/output pin. |
| | | I | **RXD —** Receiver input for UART. |
| | | O | **CT32B0_MAT0 —** Match output 0 for 32-bit timer 0. |
| PIO1_7/TXD/ CT32B0_MAT1 | 32[3] | I/O | **PIO1_7 —** General purpose digital input/output pin. |
| | | O | **TXD —** Transmitter output for UART. |
| | | O | **CT32B0_MAT1 —** Match output 1 for 32-bit timer 0. |
| PIO1_8/CT16B1_CAP0 | 7[3] | I/O | **PIO1_8 —** General purpose digital input/output pin. |
| | | I | **CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1. |
| PIO1_9/CT16B1_MAT0 | 12[3] | I/O | **PIO1_9 —** General purpose digital input/output pin. |
| | | O | **CT16B1_MAT0 —** Match output 0 for 16-bit timer 1. |
| PIO1_10/AD6/ CT16B1_MAT1 | 20[5] | I/O | **PIO1_10 —** General purpose digital input/output pin. |
| | | I | **AD6 —** A/D converter, input 6. |
| | | O | **CT16B1_MAT1 —** Match output 1 for 16-bit timer 1. |
| PIO1_11/AD7 | 27[5] | I/O | **PIO1_11 —** General purpose digital input/output pin. |
| | | I | **AD7 —** A/D converter, input 7. |

**Table 104. LPC1111/12/13/14 pin description table (HVQFN33 package)** *…continued*

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO2_0 | | I/O | **Port 2 —** Port 2 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 2 pins depends on the function selected through the IOCONFIG register block. Pins PIO2_1 to PIO2_11 are not available. |
| PIO2_0/$\overline{\text{DTR}}$ | 1[3] | I/O | **PIO2_0 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| PIO3_0 to PIO3_5 | | I/O | **Port 3 —** Port 3 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 3 pins depends on the function selected through the IOCONFIG register block. Pins PIO3_0, PIO3_1, PIO3_3 and PIO3_6 to PIO3_11 are not available. |
| PIO3_2 | 28[3] | I/O | **PIO3_2 —** General purpose digital input/output pin. |
| PIO3_4 | 13[3] | I/O | **PIO3_4 —** General purpose digital input/output pin. |
| PIO3_5 | 14[3] | I/O | **PIO3_5 —** General purpose digital input/output pin. |
| $V_{DD}$ | 6; 29 | I | 3.3 V supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage. |
| XTALIN | 4[6] | I | Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V. |
| XTALOUT | 5[6] | O | Output from the oscillator amplifier. |
| $V_{SS}$ | 33 | - | Thermal pad. Connect to ground. |

[1] $\overline{\text{RESET}}$ 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.

[2] Serves as Deep-sleep wake-up input pin to the start logic independently of selected pin function (see the *LPC111x/11C1x user manual*).

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

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

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

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

**Table 105. LPC11C24/C22 pin description table (LQFP48 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_0 to PIO0_11 | | | **Port 0 —** Port 0 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 0 pins depends on the function selected through the IOCONFIG register block. |
| $\overline{\text{RESET}}$/PIO0_0 | 3[1] | I | **$\overline{\text{RESET}}$ —** External reset input with 20 ns glitch filter. A LOW-going pulse as short as 50 ns on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. |
| | | I/O | **PIO0_0 —** General purpose digital input/output pin with 10 ns glitch filter. |
| PIO0_1/CLKOUT/ CT32B0_MAT2 | 4[3] | I/O | **PIO0_1 —** General purpose digital input/output pin. A LOW level on this pin during reset starts the flash ISP command handler via UART (if PIO0_3 is HIGH) or via C_CAN (if PIO0_3 is LOW). |
| | | O | **CLKOUT —** Clockout pin. |
| | | O | **CT32B0_MAT2 —** Match output 2 for 32-bit timer 0. |
| PIO0_2/SSEL0/ CT16B0_CAP0 | 10[3] | I/O | **PIO0_2 —** General purpose digital input/output pin. |
| | | I/O | **SSEL0 —** Slave Select for SPI0. |
| | | I | **CT16B0_CAP0 —** Capture input 0 for 16-bit timer 0. |

**Table 105. LPC11C24/C22 pin description table (LQFP48 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO0_3 | 14[3] | I/O | **PIO0_3 —** General purpose digital input/output pin. This pin is monitored during reset: Together with a LOW level on pin PIO0_1, a LOW level starts the flash ISP command handler via C_CAN and a HIGH level starts the flash ISP command handler via UART. |
| PIO0_4/SCL | 15[4] | I/O | **PIO0_4 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SCL —** $I^2C$-bus, open-drain clock input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_5/SDA | 16[4] | I/O | **PIO0_5 —** General purpose digital input/output pin (open-drain). |
| | | I/O | **SDA —** $I^2C$-bus, open-drain data input/output. High-current sink only if $I^2C$ Fast-mode Plus is selected in the I/O configuration register. |
| PIO0_6/SCK0 | 23[3] | I/O | **PIO0_6 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO0_7/$\overline{\text{CTS}}$ | 24[3] | I/O | **PIO0_7 —** General purpose digital input/output pin (high-current output driver). |
| | | I | **$\overline{\text{CTS}}$ —** Clear To Send input for UART. |
| PIO0_8/MISO0/ CT16B0_MAT0 | 27[3] | I/O | **PIO0_8 —** General purpose digital input/output pin. |
| | | I/O | **MISO0 —** Master In Slave Out for SPI0. |
| | | O | **CT16B0_MAT0 —** Match output 0 for 16-bit timer 0. |
| PIO0_9/MOSI0/ CT16B0_MAT1 | 28[3] | I/O | **PIO0_9 —** General purpose digital input/output pin. |
| | | I/O | **MOSI0 —** Master Out Slave In for SPI0. |
| | | O | **CT16B0_MAT1 —** Match output 1 for 16-bit timer 0. |
| SWCLK/PIO0_10/ SCK0/ CT16B0_MAT2 | 29[3] | I | **SWCLK —** Serial wire clock. |
| | | I/O | **PIO0_10 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| | | O | **CT16B0_MAT2 —** Match output 2 for 16-bit timer 0. |
| R/PIO0_11/ AD0/ CT32B0_MAT3 | 32[5] | - | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO0_11 —** General purpose digital input/output pin. |
| | | I | **AD0 —** A/D converter, input 0. |
| | | O | **CT32B0_MAT3 —** Match output 3 for 32-bit timer 0. |
| PIO1_0 to PIO1_11 | | | **Port 1 —** Port 1 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 1 pins depends on the function selected through the IOCONFIG register block. |
| R/PIO1_0/AD1/ CT32B1_CAP0 | 33[5] | - | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_0 —** General purpose digital input/output pin. |
| | | I | **AD1 —** A/D converter, input 1. |
| | | I | **CT32B1_CAP0 —** Capture input 0 for 32-bit timer 1. |
| R/PIO1_1/AD2/ CT32B1_MAT0 | 34[5] | - | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_1 —** General purpose digital input/output pin. |
| | | I | **AD2 —** A/D converter, input 2. |
| | | O | **CT32B1_MAT0 —** Match output 0 for 32-bit timer 1. |
| R/PIO1_2/AD3/ CT32B1_MAT1 | 35[5] | - | **R —** Reserved. Configure for an alternate function in the IOCONFIG block. |
| | | I/O | **PIO1_2 —** General purpose digital input/output pin. |
| | | I | **AD3 —** A/D converter, input 3. |
| | | O | **CT32B1_MAT1 —** Match output 1 for 32-bit timer 1. |

**Table 105. LPC11C24/C22 pin description table (LQFP48 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| SWDIO/PIO1_3/ AD4/ CT32B1_MAT2 | 39[5] | I/O | **SWDIO —** Serial wire debug input/output. |
| | | I/O | **PIO1_3 —** General purpose digital input/output pin. |
| | | I | **AD4 —** A/D converter, input 4. |
| | | O | **CT32B1_MAT2 —** Match output 2 for 32-bit timer 1. |
| PIO1_4/AD5/ CT32B1_MAT3/ WAKEUP | 40[5] | I/O | **PIO1_4 —** General purpose digital input/output pin with 10 ns glitch filter. |
| | | I | **AD5 —** A/D converter, input 5. |
| | | O | **CT32B1_MAT3 —** Match output 3 for 32-bit timer 1. |
| | | I | **WAKEUP —** Deep power-down mode wake-up pin with 20 ns glitch filter. This pin must be pulled HIGH externally to enter Deep power-down mode and pulled LOW to exit Deep power-down mode. A LOW-going pulse as short as 50 ns wakes up the part. |
| PIO1_5/$\overline{\text{RTS}}$/ CT32B0_CAP0 | 45[3] | I/O | **PIO1_5 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{RTS}}$ —** Request To Send output for UART. |
| | | I | **CT32B0_CAP0 —** Capture input 0 for 32-bit timer 0. |
| PIO1_6/RXD/ CT32B0_MAT0 | 46[3] | I/O | **PIO1_6 —** General purpose digital input/output pin. |
| | | I | **RXD —** Receiver input for UART. |
| | | O | **CT32B0_MAT0 —** Match output 0 for 32-bit timer 0. |
| PIO1_7/TXD/ CT32B0_MAT1 | 47[3] | I/O | **PIO1_7 —** General purpose digital input/output pin. |
| | | O | **TXD —** Transmitter output for UART. |
| | | O | **CT32B0_MAT1 —** Match output 1 for 32-bit timer 0. |
| PIO1_8/ CT16B1_CAP0 | 9[3] | I/O | **PIO1_8 —** General purpose digital input/output pin. |
| | | I | **CT16B1_CAP0 —** Capture input 0 for 16-bit timer 1. |
| PIO1_10/AD6/ CT16B1_MAT1 | 30[5] | I/O | **PIO1_10 —** General purpose digital input/output pin. |
| | | I | **AD6 —** A/D converter, input 6. |
| | | O | **CT16B1_MAT1 —** Match output 1 for 16-bit timer 1. |
| PIO1_11/AD7 | 42[5] | I/O | **PIO1_11 —** General purpose digital input/output pin. |
| | | I | **AD7 —** A/D converter, input 7. |
| PIO2_0 to PIO2_11 | | | **Port 2 —** Port 2 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 2 pins depends on the function selected through the IOCONFIG register block. |
| PIO2_0/$\overline{\text{DTR}}$/ SSEL1 | 2[3] | I/O | **PIO2_0 —** General purpose digital input/output pin. |
| | | I/O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| | | I/O | **SSEL1 —** Slave Select for SPI1. |
| PIO2_1/$\overline{\text{DSR}}$/SCK1 | 13[3] | I/O | **PIO2_1 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DSR}}$ —** Data Set Ready input for UART. |
| | | I/O | **SCK1 —** Serial clock for SPI1. |
| PIO2_2/$\overline{\text{DCD}}$/ MISO1 | 26[3] | I/O | **PIO2_2 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DCD}}$ —** Data Carrier Detect input for UART. |
| | | I/O | **MISO1 —** Master In Slave Out for SPI1. |
| PIO2_3/$\overline{\text{RI}}$/MOSI1 | 38[3] | I/O | **PIO2_3 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{RI}}$ —** Ring Indicator input for UART. |
| | | I/O | **MOSI1 —** Master Out Slave In for SPI1. |

**Table 105. LPC11C24/C22 pin description table (LQFP48 package)**

| Symbol | Pin | Type | Description |
|---|---|---|---|
| PIO2_6 | 1[3] | I/O | **PIO2_6 —** General purpose digital input/output pin. |
| PIO2_7 | 11[3] | I/O | **PIO2_7 —** General purpose digital input/output pin. |
| PIO2_8 | 12[3] | I/O | **PIO2_8 —** General purpose digital input/output pin. |
| PIO2_10 | 25[3] | I/O | **PIO2_10 —** General purpose digital input/output pin. |
| PIO2_11/SCK0 | 31[3] | I/O | **PIO2_11 —** General purpose digital input/output pin. |
| | | I/O | **SCK0 —** Serial clock for SPI0. |
| PIO3_0 to PIO3_3 | | | **Port 3 —** Port 3 is a 12-bit I/O port with individual direction and function controls for each bit. The operation of port 3 pins depends on the function selected through the IOCONFIG register block. Pins PIO3_4 to PIO3_11 are not available. |
| PIO3_0/$\overline{\text{DTR}}$ | 36[3] | I/O | **PIO3_0 —** General purpose digital input/output pin. |
| | | O | **$\overline{\text{DTR}}$ —** Data Terminal Ready output for UART. |
| PIO3_1/$\overline{\text{DSR}}$ | 37[3] | I/O | **PIO3_1 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DSR}}$ —** Data Set Ready input for UART. |
| PIO3_2/$\overline{\text{DCD}}$ | 43[3] | I/O | **PIO3_2 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{DCD}}$ —** Data Carrier Detect input for UART. |
| PIO3_3/$\overline{\text{RI}}$ | 48[3] | I/O | **PIO3_3 —** General purpose digital input/output pin. |
| | | I | **$\overline{\text{RI}}$ —** Ring Indicator input for UART. |
| CANL | 18 | I/O | LOW-level CAN bus line. |
| CANH | 19 | I/O | HIGH-level CAN bus line. |
| STB | 22 | I | Silent mode control input for CAN transceiver (LOW = Normal mode, HIGH = silent mode). |
| VDD_CAN | 17 | - | Supply voltage for I/O level of CAN transceiver. |
| $V_{CC}$ | 20 | - | Supply voltage for CAN transceiver. |
| GND | 21 | - | Ground for CAN transceiver. |
| $V_{DD}$ | 8;44 | I | Supply voltage to the internal regulator, the external rail, and the ADC. Also used as the ADC reference voltage. |
| XTALIN | 6[7] | I | Input to the oscillator circuit and internal clock generator circuits. Input voltage must not exceed 1.8 V. |
| XTALOUT | 7[7] | O | Output from the oscillator amplifier. |
| $V_{SS}$ | 5; 41 | I | Ground. |

[1] $\overline{\text{RESET}}$ 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.

[2] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors and configurable hysteresis.

[3] $I^2C$-bus pads compliant with the $I^2C$-bus specification for $I^2C$ standard mode and $I^2C$ Fast-mode Plus.

[4] 5 V tolerant pad providing digital I/O functions with configurable pull-up/pull-down resistors, configurable hysteresis, and analog input. When configured as a ADC input, digital section of the pad is disabled and the pin is not 5 V tolerant.

[5] 5 V tolerant digital I/O pad without pull-up/pull-down resistors.

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

## 9.1 How to read this chapter

The number of GPIO pins available on each port depends on the LPC111x/LPC11Cxx part and the package. See Table 106 for available GPIO pins:

**Table 106. GPIO configuration**

| Part | Package | GPIO port 0 | GPIO port 1 | GPIO port 2 | GPIO port 3 | Total GPIO pins |
|---|---|---|---|---|---|---|
| LPC1111 | HVQFN33 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 | PIO3_2; PIO3_4; PIO3_5 | 28 |
| LPC1112 | HVQFN33 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 | PIO3_2; PIO3_4; PIO3_5 | 28 |
| LPC1113 | HVQFN33 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 | PIO3_2; PIO3_4; PIO3_5 | 28 |
| | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 to PIO2_11 | PIO3_0 to PIO3_5 | 42 |
| LPC1114 | HVQFN33 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 | PIO3_2; PIO3_4; PIO3_5 | 28 |
| | PLCC44 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 to PIO2_11 | PIO3_4 and PIO3_5 | 38 |
| | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 to PIO2_11 | PIO3_0 to PIO3_5 | 42 |
| LPC11C12 | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 to PIO2_11 | PIO3_0 to PIO3_3 | 40 |
| LPC11C14 | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 | PIO2_0 to PIO2_11 | PIO3_0 to PIO3_3 | 40 |
| LPC11C22 | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 except PIO1_9 | PIO2_0 to PIO2_11 except PIO2_4, PIO2_5, PIO2_9 | PIO3_0 to PIO3_3 | 36 |
| LPC11C24 | LQFP48 | PIO0_0 to PIO0_11 | PIO1_0 to PIO1_11 except PIO1_9 | PIO2_0 to PIO2_11 except PIO2_4, PIO2_5, PIO2_9 | PIO3_0 to PIO3_3 | 36 |

Register bits corresponding to PIOn_m pins which are not available are reserved.

## 9.2 Introduction

### 9.2.1 Features

- GPIO pins can be configured as input or output by software.
- Each individual port pin can serve as an edge or level-sensitive interrupt request.
- Interrupts can be configured on single falling or rising edges and on both edges.
- Level-sensitive interrupt pins can be HIGH or LOW-active.
- All GPIO pins are inputs by default.
- Reading and writing of data registers are masked by address bits 13:2.

## 9.3 Register description

Each GPIO register can be up to 12 bits wide and can be read or written using word or half-word operations at word addresses.

**Table 107. Register overview: GPIO (base address port 0: 0x5000 0000; port 1: 0x5001 0000, port 2: 0x5002 0000; port 3: 0x5003 0000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| GPIOnDATA | R/W | 0x0000 to 0x3FF8 | Port n data address masking register locations for pins PIOn_0 to PIOn_11 (see Section 9.4.1). | n/a |
| GPIOnDATA | R/W | 0x3FFC | Port n data register for pins PIOn_0 to PIOn_11 | n/a |
| - | - | 0x4000 to 0x7FFC | reserved | - |
| GPIOnDIR | R/W | 0x8000 | Data direction register for port n | 0x00 |
| GPIOnIS | R/W | 0x8004 | Interrupt sense register for port n | 0x00 |
| GPIOnIBE | R/W | 0x8008 | Interrupt both edges register for port n | 0x00 |
| GPIOnIEV | R/W | 0x800C | Interrupt event register for port n | 0x00 |
| GPIOnIE | R/W | 0x8010 | Interrupt mask register for port n | 0x00 |
| GPIOnRIS | R | 0x8014 | Raw interrupt status register for port n | 0x00 |
| GPIOnMIS | R | 0x8018 | Masked interrupt status register for port n | 0x00 |
| GPIOnIC | W | 0x801C | Interrupt clear register for port n | 0x00 |
| - | - | 0x8020 - 0xFFFF | reserved | 0x00 |

### 9.3.1 GPIO data register

The GPIOnDATA register holds the current logic state of the pin (HIGH or LOW), independently of whether the pin is configured as an GPIO input or output or as another digital function. If the pin is configured as GPIO output, the current value of the GPIOnDATA register is driven to the pin.

**Table 108. GPIOnDATA register (GPIO0DATA, address 0x5000 0000 to 0x5000 3FFC; GPIO1DATA, address 0x5001 0000 to 0x5001 3FFC; GPIO2DATA, address 0x5002 0000 to 0x5002 3FFC; GPIO3DATA, address 0x5003 0000 to 0x5003 3FFC) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | DATA | Logic levels for pins PIOn_0 to PIOn_11. HIGH = 1, LOW = 0. | n/a | R/W |
| 31:12 | - | Reserved | - | - |

A read of the GPIOnDATA register always returns the current logic level (state) of the pin independently of its configuration. Because there is a single data register for both the value of the output driver and the state of the pin's input, write operations have different effects depending on the pin's configuration:

- If a pin is configured as GPIO input, a write to the GPIOnDATA register has no effect on the pin level. A read returns the current state of the pin.
- If a pin is configured as GPIO output, the current value of GPIOnDATA register is driven to the pin. This value can be a result of writing to the GPIOnDATA register, or it can reflect the previous state of the pin if the pin is switched to GPIO output from GPIO input or another digital function. A read returns the current state of the pin.

- If a pin is configured as another digital function (input or output), a write to the GPIOnDATA register has no effect on the pin level. A read returns the current state of the pin even if it is configured as an output. This means that by reading the GPIOnDATA register, the digital output or input value of a function other than GPIO on that pin can be observed.

The following rules apply when the pins are switched from input to output:

- Pin is configured as input with a HIGH level applied:
  - Change pin to output: pin drives HIGH level.
- Pin is configured as input with a LOW level applied:
  - Change pin to output: pin drives LOW level.

The rules show that the pins mirror the current logic level. Therefore floating pins may drive an unpredictable level when switched from input to output.

### 9.3.2 GPIO data direction register

**Table 109. GPIOnDIR register (GPIO0DIR, address 0x5000 8000 to GPIO3DIR, address 0x5003 8000) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | IO | Selects pin x as input or output (x = 0 to 11).<br>0 = Pin PIOn_x is configured as input.<br>1 = Pin PIOn_x is configured as output. | 0x00 | R/W |
| 31:12 | - | Reserved | - | - |

### 9.3.3 GPIO interrupt sense register

**Table 110. GPIOnIS register (GPIO0IS, address 0x5000 8004 to GPIO3IS, address 0x5003 8004) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | ISENSE | Selects interrupt on pin x as level or edge sensitive (x = 0 to 11).<br>0 = Interrupt on pin PIOn_x is configured as edge sensitive.<br>1 = Interrupt on pin PIOn_x is configured as level sensitive. | 0x00 | R/W |
| 31:12 | - | Reserved | - | - |

### 9.3.4 GPIO interrupt both edges sense register

**Table 111. GPIOnIBE register (GPIO0IBE, address 0x5000 8008 to GPIO3IBE, address 0x5003 8008) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | IBE | Selects interrupt on pin x to be triggered on both edges (x = 0 to 11).<br>0 = Interrupt on pin PIOn_x is controlled through register GPIOnIEV.<br>1 = Both edges on pin PIOn_x trigger an interrupt. | 0x00 | R/W |
| 31:12 | - | Reserved | - | - |

### 9.3.5 GPIO interrupt event register

**Table 112. GPIOnIEV register (GPIO0IEV, address 0x5000 800C to GPIO3IEV, address 0x5003 800C) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | IEV | Selects interrupt on pin x to be triggered rising or falling edges (x = 0 to 11).<br>0 = Depending on setting in register GPIOnIS (see Table 110), falling edges or LOW level on pin PIOn_x trigger an interrupt.<br>1 = Depending on setting in register GPIOnIS (see Table 110), rising edges or HIGH level on pin PIOn_x trigger an interrupt. | 0x00 | R/W |
| 31:12 | - | Reserved | - | - |

### 9.3.6 GPIO interrupt mask register

Bits set to HIGH in the GPIOnIE register allow the corresponding pins to trigger their individual interrupts and the combined GPIOnINTR line. Clearing a bit disables interrupt triggering on that pin.

**Table 113. GPIOnIE register (GPIO0IE, address 0x5000 8010 to GPIO3IE, address 0x5003 8010) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | MASK | Selects interrupt on pin x to be masked (x = 0 to 11).<br>0 = Interrupt on pin PIOn_x is masked.<br>1 = Interrupt on pin PIOn_x is not masked. | 0x00 | R/W |
| 31:12 | - | Reserved | - | - |

### 9.3.7 GPIO raw interrupt status register

Bits read HIGH in the GPIOnRIS register reflect the raw (prior to masking) interrupt status of the corresponding pins indicating that all the requirements have been met before they are allowed to trigger the GPIOIE. Bits read as zero indicate that the corresponding input pins have not initiated an interrupt. The register is read-only.

**Table 114. GPIOnRIS register (GPIO0RIS, address 0x5000 8014 to GPIO3RIS, address 0x5003 8014) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | RAWST | Raw interrupt status (x = 0 to 11).<br>0 = No interrupt on pin PIOn_x.<br>1 = Interrupt requirements met on PIOn_x. | 0x00 | R |
| 31:12 | - | Reserved | - | - |

### 9.3.8 GPIO masked interrupt status register

Bits read HIGH in the GPIOnMIS register reflect the status of the input lines triggering an interrupt. Bits read as LOW indicate that either no interrupt on the corresponding input pins has been generated or that the interrupt is masked. GPIOMIS is the state of the interrupt after masking. The register is read-only.

**Table 115. GPIOnMIS register (GPIO0MIS, address 0x5000 8018 to GPIO3MIS, address 0x5003 8018) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | MASK | Selects interrupt on pin x to be masked (x = 0 to 11).<br>0 = No interrupt or interrupt masked on pin PIOn_x.<br>1 = Interrupt on PIOn_x. | 0x00 | R |
| 31:12 | - | Reserved | - | - |

### 9.3.9 GPIO interrupt clear register

This register allows software to clear edge detection for port bits that are identified as edge-sensitive in the Interrupt Sense register. This register has no effect on port bits identified as level-sensitive.

**Table 116. GPIOnIC register (GPIO0IC, address 0x5000 801C to GPIO3IC, address 0x5003 801C) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 11:0 | CLR | Selects interrupt on pin x to be cleared (x = 0 to 11). Clears the interrupt edge detection logic. This register is write-only.<br>**Remark:** The synchronizer between the GPIO and the NVIC blocks causes a delay of 2 clocks. It is recommended to add two NOPs after the clear of the interrupt edge detection logic before the exit of the interrupt service routine.<br>0 = No effect.<br>1 = Clears edge detection logic for pin PIOn_x. | 0x00 | W |
| 31:12 | - | Reserved | - | - |

## 9.4 Functional description

### 9.4.1 Write/read data operation

In order for software to be able to set GPIO bits without affecting any other pins in a single write operation, bits [13:2] of a 14-bit wide address bus are used to create a 12-bit wide mask for write and read operations on the 12 GPIO pins for each port. Only GPIOnDATA bits masked by 1 are affected by read and write operations. The masked GPIOnDATA register can be located anywhere between address offsets 0x0000 to 0x3FFC in the GPIOn address space. Reading and writing to the GPIOnDATA register at address 0x3FFC sets all masking bits to 1.

#### Write operation

If the address bit (i+2) associated with the GPIO port bit i (i = 0 to 11) to be written is HIGH, the value of the GPIODATA register bit i is updated. If the address bit (i+2) is LOW, the corresponding GPIODATA register bit i is left unchanged.

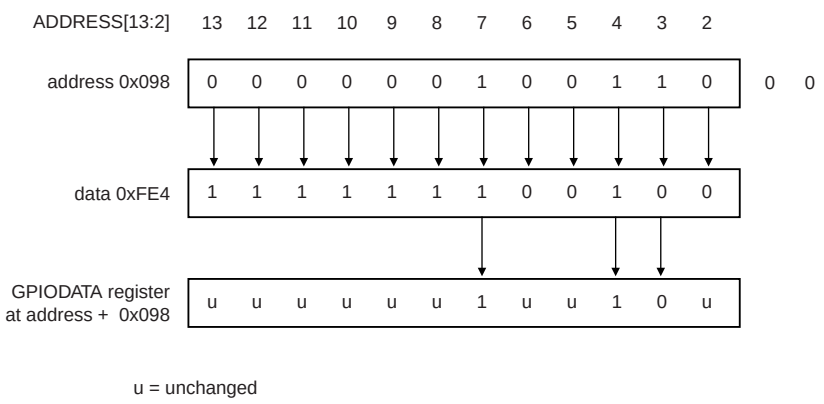



**Fig 14. Masked write operation to the GPIODATA register**

### Read operation

If the address bit associated with the GPIO data bit is HIGH, the value is read. If the address bit is LOW, the GPIO data bit is read as 0. Reading a port DATA register yields the state of port pins 11:0 ANDed with address bits 13:2.

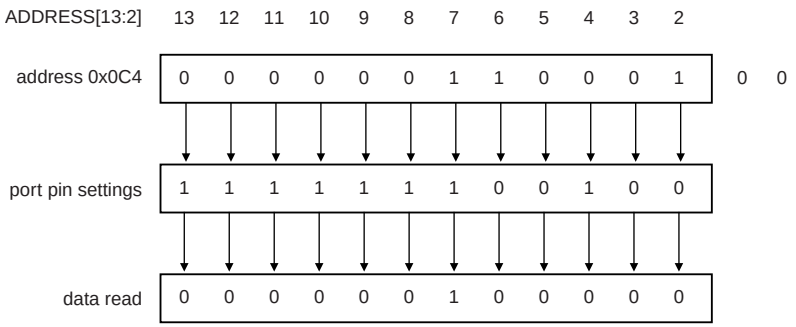



**Fig 15. Masked read operation**

## 10.1 How to read this chapter

The UART block is identical for all LPC111x and LPC11Cxx parts. The $\overline{DSR}$, $\overline{DCD}$, and $\overline{RI}$ modem signals are pinned out for LQFP48 and PLCC44 packages only.

## 10.2 Basic configuration

The UART is configured using the following registers:

1. Pins: The UART pins must be configured in the IOCONFIG register block (Section 7.4) before the UART clocks can be enabled.

   **Remark:** If the modem input pins are used, the modem function location must be also selected in the UART location registers (Section 7.4)
2. Power: In the SYSAHBCLKCTRL register, set bit 12 (Table 20).
3. Peripheral clock: Enable the UART peripheral clock by writing to the UARTCLKDIV register (Table 22).

## 10.3 Features

- 16-byte receive and transmit FIFOs.
- Register locations conform to ‘550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Built-in baud rate generator.
- UART allows for implementation of either software or hardware flow control.
- RS-485/EIA-485 9-bit mode support with output enable.
- Modem control.

## 10.4 Pin description

**Table 117. UART pin description**

| Pin | Type | Description |
|---|---|---|
| RXD | Input | **Serial Input.** Serial receive data. |
| TXD | Output | **Serial Output.** Serial transmit data. |
| $\overline{\text{RTS}}$ | Output | Request To Send. RS-485 direction control pin. |
| $\overline{\text{DTR}}$ | Output | Data Terminal Ready. |
| $\overline{\text{DSR}}$[1] | Input | Data Set Ready. |
| $\overline{\text{CTS}}$ | Input | Clear To Send. |
| $\overline{\text{DCD}}$[1] | Input | Data Carrier Detect. |
| $\overline{\text{RI}}$[1] | Input | Ring Indicator. |

[1] LQFP48 packages only.

The $\overline{\text{DSR}}$, $\overline{\text{DCD}}$, and $\overline{\text{RI}}$ modem inputs are multiplexed to two different pin locations. Use the IOCON_LOC registers (see Section 7.4) to select a physical location for each function on the LQFP48 pin package in addition to selecting the function in the IOCON registers.

The $\overline{\text{DTR}}$ output is available in two pin locations as well. The output value of the $\overline{\text{DTR}}$ pin is driven in both locations identically, and the $\overline{\text{DTR}}$ function at any location can be selected simply by selecting the function in the IOCON register for that pin location.

## 10.5 Register description

The UART contains registers organized as shown in Table 118. The Divisor Latch Access Bit (DLAB) is contained in U0LCR[7] and enables access to the Divisor Latches.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

**Table 118. Register overview: UART (base address: 0x4000 8000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| U0RBR | RO | 0x000 | Receiver Buffer Register. Contains the next received character to be read. (DLAB=0) | NA |
| U0THR | WO | 0x000 | Transmit Holding Register. The next character to be transmitted is written here. (DLAB=0) | NA |
| U0DLL | R/W | 0x000 | Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider. (DLAB=1) | 0x01 |
| U0DLM | R/W | 0x004 | Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider. (DLAB=1) | 0x00 |
| U0IER | R/W | 0x004 | Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART interrupts. (DLAB=0) | 0x00 |
| U0IIR | RO | 0x008 | Interrupt ID Register. Identifies which interrupt(s) are pending. | 0x01 |
| U0FCR | WO | 0x008 | FIFO Control Register. Controls UART FIFO usage and modes. | 0x00 |

**Table 118. Register overview: UART (base address: 0x4000 8000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| U0LCR | R/W | 0x00C | Line Control Register. Contains controls for frame formatting and break generation. | 0x00 |
| U0MCR | R/W | 0x010 | Modem control register | 0x00 |
| U0LSR | RO | 0x014 | Line Status Register. Contains flags for transmit and receive status, including line errors. | 0x60 |
| U0MSR | RO | 0x018 | Modem status register | 0x00 |
| U0SCR | R/W | 0x01C | Scratch Pad Register. Eight-bit temporary storage for software. | 0x00 |
| U0ACR | R/W | 0x020 | Auto-baud Control Register. Contains controls for the auto-baud feature. | 0x00 |
| - | - | 0x024 | Reserved | - |
| U0FDR | R/W | 0x028 | Fractional Divider Register. Generates a clock input for the baud rate divider. | 0x10 |
| - | - | 0x02C | Reserved | - |
| U0TER | R/W | 0x030 | Transmit Enable Register. Turns off UART transmitter for use with software flow control. | 0x80 |
| - | - | 0x034 - 0x048 | Reserved | - |
| U0RS485CTRL | R/W | 0x04C | RS-485/EIA-485 Control. Contains controls to configure various aspects of RS-485/EIA-485 modes. | 0x00 |
| U0RS485ADRMATCH | R/W | 0x050 | RS-485/EIA-485 address match. Contains the address match value for RS-485/EIA-485 mode. | 0x00 |
| U0RS485DLY | R/W | 0x054 | RS-485/EIA-485 direction control delay. | 0x00 |

### 10.5.1 UART Receiver Buffer Register (U0RBR - 0x4000 8000, when DLAB = 0, Read Only)

The U0RBR is the top byte of the UART RX FIFO. The top byte of the RX FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. The U0RBR is always Read Only.

Since PE, FE and BI bits (see Table 130) correspond to the byte sitting on the top of the RBR FIFO (i.e. the one that will be read in the next read from the RBR), the right approach for fetching the valid pair of received byte and its status bits is first to read the content of the U0LSR register, and then to read a byte from the U0RBR.

**Table 119. UART Receiver Buffer Register (U0RBR - address 0x4000 8000 when DLAB = 0, Read Only) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | RBR | The UART Receiver Buffer Register contains the oldest received byte in the UART RX FIFO. | undefined |
| 31:8 | - | Reserved | - |

### 10.5.2 UART Transmitter Holding Register (U0THR - 0x4000 8000 when DLAB = 0, Write Only)

The U0THR is the top byte of the UART TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0THR. The U0THR is always Write Only.

**Table 120. UART Transmitter Holding Register (U0THR - address 0x4000 8000 when DLAB = 0, Write Only) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | THR | Writing to the UART Transmit Holding Register causes the data to be stored in the UART transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available. | NA |
| 31:8 | - | Reserved | - |

### 10.5.3 UART Divisor Latch LSB and MSB Registers (U0DLL - 0x4000 8000 and U0DLM - 0x4000 8004, when DLAB = 1)

The UART Divisor Latch is part of the UART Baud Rate Generator and holds the value used, along with the Fractional Divider, to divide the UART_PCLK clock in order to produce the baud rate clock, which must be 16x the desired baud rate. The U0DLL and U0DLM registers together form a 16-bit divisor where U0DLL contains the lower 8 bits of the divisor and U0DLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART Divisor Latches. Details on how to select the right value for U0DLL and U0DLM can be found in Section 10.5.15.

**Table 121. UART Divisor Latch LSB Register (U0DLL - address 0x4000 8000 when DLAB = 1) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DLLSB | The UART Divisor Latch LSB Register, along with the U0DLM register, determines the baud rate of the UART. | 0x01 |
| 31:8 | - | Reserved | - |

**Table 122. UART Divisor Latch MSB Register (U0DLM - address 0x4000 8004 when DLAB = 1) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DLMSB | The UART Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UART. | 0x00 |
| 31:8 | - | Reserved | - |

### 10.5.4 UART Interrupt Enable Register (U0IER - 0x4000 8004, when DLAB = 0)

The U0IER is used to enable the four UART interrupt sources.

**Table 123. UART Interrupt Enable Register (U0IER - address 0x4000 8004 when DLAB = 0) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | RBRIE | | RBR Interrupt Enable. Enables the Receive Data Available interrupt for UART. It also controls the Character Receive Time-out interrupt. | 0 |
| | | 0 | Disable the RDA interrupt. | |
| | | 1 | Enable the RDA interrupt. | |
| 1 | THREIE | | THRE Interrupt Enable. Enables the THRE interrupt for UART. The status of this interrupt can be read from U0LSR[5]. | 0 |
| | | 0 | Disable the THRE interrupt. | |
| | | 1 | Enable the THRE interrupt. | |
| 2 | RXLIE | | RX Line Interrupt Enable. Enables the UART RX line status interrupts. The status of this interrupt can be read from U0LSR[4:1]. | 0 |
| | | 0 | Disable the RX line status interrupts. | |
| | | 1 | Enable the RX line status interrupts. | |
| 3 | - | - | Reserved | - |
| 6:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7 | - | - | Reserved | 0 |
| 8 | ABEOIntEn | | Enables the end of auto-baud interrupt. | 0 |
| | | 0 | Disable end of auto-baud Interrupt. | |
| | | 1 | Enable end of auto-baud Interrupt. | |

**Table 123. UART Interrupt Enable Register (U0IER - address 0x4000 8004 when DLAB = 0) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 9 | ABTOIntEn | | Enables the auto-baud time-out interrupt. | 0 |
| | | 0 | Disable auto-baud time-out Interrupt. | |
| | | 1 | Enable auto-baud time-out Interrupt. | |
| 31:10 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 10.5.5 UART Interrupt Identification Register (U0IIR - 0x4004 8008, Read Only)

U0IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during a U0IIR access. If an interrupt occurs during a U0IIR access, the interrupt is recorded for the next U0IIR access.

**Table 124. UART Interrupt Identification Register (U0IIR - address 0x4004 8008, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | IntStatus | | Interrupt status. Note that U0IIR[0] is active low. The pending interrupt can be determined by evaluating U0IIR[3:1]. | 1 |
| | | 0 | At least one interrupt is pending. | |
| | | 1 | No interrupt is pending. | |
| 3:1 | IntId | | Interrupt identification. U0IER[3:1] identifies an interrupt corresponding to the UART Rx FIFO. All other combinations of U0IER[3:1] not listed below are reserved (100,101,111). | 0 |
| | | 0x3 | 1 - Receive Line Status (RLS). | |
| | | 0x2 | 2a - Receive Data Available (RDA). | |
| | | 0x6 | 2b - Character Time-out Indicator (CTI). | |
| | | 0x1 | 3 - THRE Interrupt. | |
| | | 0x0 | 4 - Modem interrupt. | |
| 5:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7:6 | FIFOEnable | | These bits are equivalent to U0FCR[0]. | 0 |
| 8 | ABEOInt | | End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled. | 0 |
| 9 | ABTOInt | | Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled. | 0 |
| 31:10 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

Bits U0IIR[9:8] are set by the auto-baud function and signal a time-out or end of auto-baud condition. The auto-baud interrupt conditions are cleared by setting the corresponding Clear bits in the Auto-baud Control Register.

If the IntStatus bit is one and no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in Table 125. Given the status of U0IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The U0IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UART RLS interrupt (U0IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART RX input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART Rx error condition that set the interrupt can be observed via U0LSR[4:1]. The interrupt is cleared upon a U0LSR read.

The UART RDA interrupt (U0IIR[3:1] = 010) shares the second level priority with the CTI interrupt (U0IIR[3:1] = 110). The RDA is activated when the UART Rx FIFO reaches the trigger level defined in U0FCR7:6 and is reset when the UART Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (U0IIR[3:1] = 110) is a second level interrupt and is set when the UART Rx FIFO contains at least one character and no UART Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART Rx FIFO activity (read or write of UART RSR) will clear the interrupt. This interrupt is intended to flush the UART RBR after a message has been received that is not a multiple of the trigger level size. For example, if a peripheral wished to send a 105 character message and the trigger level was 10 characters, the CPU would receive 10 RDA interrupts resulting in the transfer of 100 characters and 1 to 5 CTI interrupts (depending on the service routine) resulting in the transfer of the remaining 5 characters.

**Table 125. UART Interrupt Handling**

| U0IIR[3:0] value[1] | Priority | Interrupt type | Interrupt source | Interrupt reset |
|---|---|---|---|---|
| 0001 | - | None | None | - |
| 0110 | Highest | RX Line Status / Error | OE[2] or PE[2] or FE[2] or BI[2] | U0LSR Read[2] |

**Table 125. UART Interrupt Handling**

| U0IIR[3:0] value[1] | Priority | Interrupt type | Interrupt source | Interrupt reset |
|---|---|---|---|---|
| 0100 | Second | RX Data Available | Rx data available or trigger level reached in FIFO (U0FCR0=1) | U0RBR Read[3] or UART FIFO drops below trigger level |
| 1100 | Second | Character Time-out indication | Minimum of one character in the RX FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).<br>The exact time will be:<br>[(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs | U0RBR Read[3] |
| 0010 | Third | THRE | THRE[2] | U0IIR Read[4] (if source of interrupt) or THR write |

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

[2] For details see Section 10.5.9 “UART Line Status Register (U0LSR - 0x4000 8014, Read Only)”

[3] For details see Section 10.5.1 “UART Receiver Buffer Register (U0RBR - 0x4000 8000, when DLAB = 0, Read Only)”

[4] For details see Section 10.5.5 “UART Interrupt Identification Register (U0IIR - 0x4004 8008, Read Only)” and Section 10.5.2 “UART Transmitter Holding Register (U0THR - 0x4000 8000 when DLAB = 0, Write Only)”

The UART THRE interrupt (U0IIR[3:1] = 001) is a third level interrupt and is activated when the UART THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE = 1 and there have not been at least two characters in the U0THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to U0THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UART THR FIFO has held two or more characters at one time and currently, the U0THR is empty. The THRE interrupt is reset when a U0THR write occurs or a read of the U0IIR occurs and the THRE is the highest interrupt (U0IIR[3:1] = 001).

### 10.5.6 UART FIFO Control Register (U0FCR - 0x4000 8008, Write Only)

The U0FCR controls the operation of the UART RX and TX FIFOs.

**Table 126. UART FIFO Control Register (U0FCR - address 0x4000 8008, Write Only) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | FIFOEn | | FIFO Enable | 0 |
| | | 0 | UART FIFOs are disabled. Must not be used in the application. | |
| | | 1 | Active high enable for both UART Rx and TX FIFOs and U0FCR[7:1] access. This bit must be set for proper UART operation. Any transition on this bit will automatically clear the UART FIFOs. | |
| 1 | RXFIFO Res | | RX FIFO Reset | 0 |
| | | 0 | No impact on either of UART FIFOs. | |
| | | 1 | Writing a logic 1 to U0FCR[1] will clear all bytes in UART Rx FIFO, reset the pointer logic. This bit is self-clearing. | |
| 2 | TXFIFO Res | | TX FIFO Reset | 0 |
| | | 0 | No impact on either of UART FIFOs. | |
| | | 1 | Writing a logic 1 to U0FCR[2] will clear all bytes in UART TX FIFO, reset the pointer logic. This bit is self-clearing. | |
| 3 | - | - | Reserved | 0 |
| 5:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7:6 | RXTL | | RX Trigger Level. These two bits determine how many receiver UART FIFO characters must be written before an interrupt is activated. | 0 |
| | | 0x0 | Trigger level 0 (1 character or 0x01). | |
| | | 0x1 | Trigger level 1 (4 characters or 0x04). | |
| | | 0x2 | Trigger level 2 (8 characters or 0x08). | |
| | | 0x3 | Trigger level 3 (14 characters or 0x0E). | |
| 31:8 | - | - | Reserved | - |

## 10.5.7 UART Line Control Register (U0LCR - 0x4000 800C)

The U0LCR determines the format of the data character that is to be transmitted or received.

**Table 127. UART Line Control Register (U0LCR - address 0x4000 800C) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 1:0 | WLS | | Word Length Select | 0 |
| | | 0x0 | 5-bit character length. | |
| | | 0x1 | 6-bit character length. | |
| | | 0x2 | 7-bit character length. | |
| | | 0x3 | 8-bit character length. | |
| 2 | SBS | | Stop Bit Select | 0 |
| | | 0 | 1 stop bit. | |
| | | 1 | 2 stop bits (1.5 if U0LCR[1:0]=00). | |

**Table 127. UART Line Control Register (U0LCR - address 0x4000 800C) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 3 | PE | | Parity Enable | 0 |
| | | 0 | Disable parity generation and checking. | |
| | | 1 | Enable parity generation and checking. | |
| 5:4 | PS | | Parity Select | 0 |
| | | 0x0 | Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd. | |
| | | 0x1 | Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even. | |
| | | 0x2 | Forced 1 stick parity. | |
| | | 0x3 | Forced 0 stick parity. | |
| 6 | BC | | Break Control | 0 |
| | | 0 | Disable break transmission. | |
| | | 1 | Enable break transmission. Output pin UART TXD is forced to logic 0 when U0LCR[6] is active high. | |
| 7 | DLAB | | Divisor Latch Access Bit | 0 |
| | | 0 | Disable access to Divisor Latches. | |
| | | 1 | Enable access to Divisor Latches. | |
| 31:8 | - | - | Reserved | - |

## 10.5.8 UART Modem Control Register

The U0MCR enables the modem loopback mode and controls the modem output signals.

**Table 128. UART0 Modem Control Register (U0MCR - address 0x4000 8010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | DTRC | | DTR Control. Source for modem output pin, $\overline{\text{DTR}}$. This bit reads as 0 when modem loopback mode is active. | 0 |
| 1 | RTSC | | RTS Control. Source for modem output pin $\overline{\text{RTS}}$. This bit reads as 0 when modem loopback mode is active. | 0 |
| 3:2 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |

**Table 128. UART0 Modem Control Register (U0MCR - address 0x4000 8010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4 | LMS | | Loopback Mode Select. The modem loopback mode provides a mechanism to perform diagnostic loopback testing. Serial data from the transmitter is connected internally to serial input of the receiver. Input pin, RXD, has no effect on loopback and output pin, TXD is held in marking state. The four modem inputs (CTS, DSR, RI and DCD) are disconnected externally. Externally, the modem outputs (RTS, DTR) are set inactive. Internally, the four modem outputs are connected to the four modem inputs. As a result of these connections, the upper four bits of the U0MSR will be driven by the lower four bits of the U0MCR rather than the four modem inputs in normal mode. This permits modem status interrupts to be generated in loopback mode by writing the lower four bits of U0MCR. | 0 |
| | | 0 | Disable modem loopback mode. | |
| | | 1 | Enable modem loopback mode. | |
| 5 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |
| 6 | RTSen | | RTS flow control | 0 |
| | | 0 | Disable auto-rts flow control. | |
| | | 1 | Enable auto-rts flow control. | |
| 7 | CTSen | | CTS flow control | 0 |
| | | 0 | Disable auto-cts flow control. | |
| | | 1 | Enable auto-cts flow control. | |
| 31:8 | - | - | Reserved | - |

### 10.5.8.1 Auto-flow control

If auto-RTS mode is enabled the UART's receiver FIFO hardware controls the $\overline{RTS}$ output of the UART. If the auto-CTS mode is enabled the UART's U0TSR hardware will only start transmitting if the $\overline{CTS}$ input signal is asserted.

#### 10.5.8.1.1 Auto-RTS

The auto-RTS function is enabled by setting the RTSen bit. Auto-RTS data flow control originates in the U0RBR module and is linked to the programmed receiver FIFO trigger level. If auto-RTS is enabled, the data-flow is controlled as follows:

When the receiver FIFO level reaches the programmed trigger level, $\overline{RTS}$ is deasserted (to a high value). It is possible that the sending UART sends an additional byte after the trigger level is reached (assuming the sending UART has another byte to send) because it might not recognize the deassertion of $\overline{RTS}$ until after it has begun sending the additional byte. $\overline{RTS}$ is automatically reasserted (to a low value) once the receiver FIFO has reached the previous trigger level. The reassertion of $\overline{RTS}$ signals the sending UART to continue transmitting data.

If Auto-RTS mode is disabled, the RTSen bit controls the $\overline{RTS}$ output of the UART. If Auto-RTS mode is enabled, hardware controls the $\overline{RTS}$ output, and the actual value of $\overline{RTS}$ will be copied in the RTS Control bit of the UART. As long as Auto-RTS is enabled, the value of the RTS Control bit is read-only for software.

Example: Suppose the UART operating in type ‘550 mode has the trigger level in U0FCR set to 0x2, then, if Auto-RTS is enabled, the UART will deassert the $\overline{RTS}$ output as soon as the receive FIFO contains 8 bytes (Table 126 on page 121). The $\overline{RTS}$ output will be reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

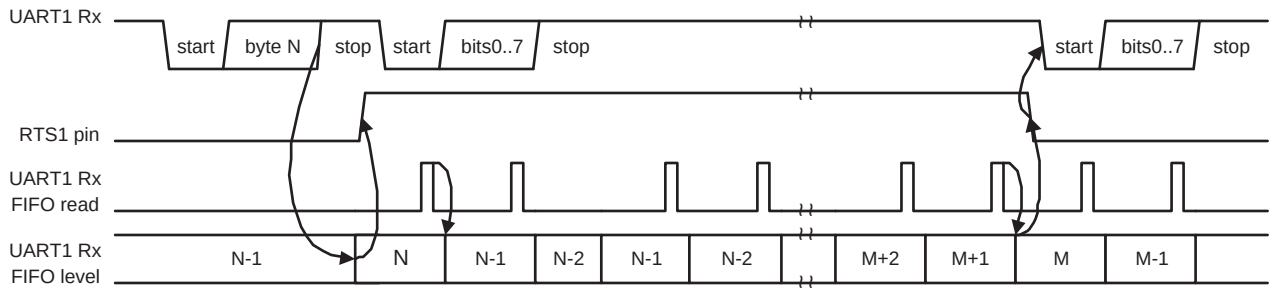



**Fig 16. Auto-RTS Functional Timing**

#### 10.5.8.1.2 Auto-CTS

The Auto-CTS function is enabled by setting the CTSen bit. If Auto-CTS is enabled, the transmitter circuitry in the U0TSR module checks $\overline{CTS}$ input before sending the next data byte. When $\overline{CTS}$ is active (low), the transmitter sends the next byte. To stop the transmitter from sending the following byte, $\overline{CTS}$ must be released before the middle of the last stop bit that is currently being sent. In Auto-CTS mode, a change of the $\overline{CTS}$ signal does not trigger a modem status interrupt unless the CTS Interrupt Enable bit is set, Delta CTS bit in the U0MSR will be set though. Table 129 lists the conditions for generating a Modem Status interrupt.

**Table 129. Modem status interrupt generation**

| Enable modem status interrupt (U0ER[3]) | CTSen (U0MCR[7]) | CTS interrupt enable (U0IER[7]) | Delta CTS (U0MSR[0]) | Delta DCD or trailing edge RI or Delta DSR (U0MSR[3] or U0MSR[2] or U0MSR[1]) | Modem status interrupt |
|---|---|---|---|---|---|
| 0 | x | x | x | x | No |
| 1 | 0 | x | 0 | 0 | No |
| 1 | 0 | x | 1 | x | Yes |
| 1 | 0 | x | x | 1 | Yes |
| 1 | 1 | 0 | x | 0 | No |
| 1 | 1 | 0 | x | 1 | Yes |
| 1 | 1 | 1 | 0 | 0 | No |
| 1 | 1 | 1 | 1 | x | Yes |
| 1 | 1 | 1 | x | 1 | Yes |

The auto-$\overline{CTS}$ function reduces interrupts to the host system. When flow control is enabled, a $\overline{CTS}$ state change does not trigger host interrupts because the device automatically controls its own transmitter. Without Auto-CTS, the transmitter sends any data present in the transmit FIFO and a receiver overrun error can result. Figure 17 illustrates the Auto-CTS functional timing.

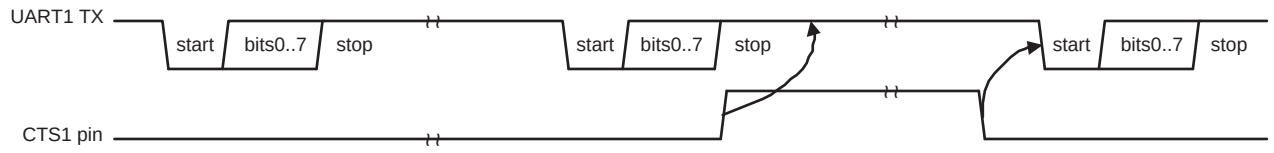



**Fig 17. Auto-CTS Functional Timing**

While starting transmission of the initial character, the $\overline{\text{CTS}}$ signal is asserted. Transmission will stall as soon as the pending transmission has completed. The UART will continue transmitting a 1 bit as long as $\overline{\text{CTS}}$ is de-asserted (high). As soon as $\overline{\text{CTS}}$ gets de-asserted, transmission resumes and a start bit is sent followed by the data bits of the next character.

### 10.5.9 UART Line Status Register (U0LSR - 0x4000 8014, Read Only)

The U0LSR is a Read Only register that provides status information on the UART TX and RX blocks.

**Table 130. UART Line Status Register (U0LSR - address 0x4000 8014, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 0 | RDR | | Receiver Data Ready. U0LSR[0] is set when the U0RBR holds an unread character and is cleared when the UART RBR FIFO is empty. | 0 |
| | | 0 | U0RBR is empty. | |
| | | 1 | U0RBR contains valid data. | |
| 1 | OE | | Overrun Error. The overrun error condition is set as soon as it occurs. A U0LSR read clears U0LSR[1]. U0LSR[1] is set when UART RSR has a new character assembled and the UART RBR FIFO is full. In this case, the UART RBR FIFO will not be overwritten and the character in the UART RSR will be lost. | 0 |
| | | 0 | Overrun error status is inactive. | |
| | | 1 | Overrun error status is active. | |
| 2 | PE | | Parity Error. When the parity bit of a received character is in the wrong state, a parity error occurs. A U0LSR read clears U0LSR[2]. Time of parity error detection is dependent on U0FCR[0].<br>**Note:** A parity error is associated with the character at the top of the UART RBR FIFO. | 0 |
| | | 0 | Parity error status is inactive. | |
| | | 1 | Parity error status is active. | |

**Table 130. UART Line Status Register (U0LSR - address 0x4000 8014, Read Only) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 3 | FE | | Framing Error. When the stop bit of a received character is a logic 0, a framing error occurs. A U0LSR read clears U0LSR[3]. The time of the framing error detection is dependent on U0FCR0. Upon detection of a framing error, the RX will attempt to re-synchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error.<br>**Note:** A framing error is associated with the character at the top of the UART RBR FIFO. | 0 |
| | | 0 | Framing error status is inactive. | |
| | | 1 | Framing error status is active. | |
| 4 | BI | | Break Interrupt. When RXD1 is held in the spacing state (all zeros) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXD1 goes to marking state (all ones). A U0LSR read clears this status bit. The time of break detection is dependent on U0FCR[0].<br>**Note:** The break interrupt is associated with the character at the top of the UART RBR FIFO. | 0 |
| | | 0 | Break interrupt status is inactive. | |
| | | 1 | Break interrupt status is active. | |
| 5 | THRE | | Transmitter Holding Register Empty. THRE is set immediately upon detection of an empty UART THR and is cleared on a U0THR write. | 1 |
| | | 0 | U0THR contains valid data. | |
| | | 1 | U0THR is empty. | |
| 6 | TEMT | | Transmitter Empty. TEMT is set when both U0THR and U0TSR are empty; TEMT is cleared when either the U0TSR or the U0THR contain valid data. | 1 |
| | | 0 | U0THR and/or the U0TSR contains valid data. | |
| | | 1 | U0THR and the U0TSR are empty. | |
| 7 | RXFE | | Error in RX FIFO. U0LSR[7] is set when a character with a RX error such as framing error, parity error or break interrupt, is loaded into the U0RBR. This bit is cleared when the U0LSR register is read and there are no subsequent errors in the UART FIFO. | 0 |
| | | 0 | U0RBR contains no UART RX errors or U0FCR[0]=0. | |
| | | 1 | UART RBR contains at least one UART RX error. | |
| 31:8 | - | - | Reserved | - |

### 10.5.10 UART Modem Status Register

The U0MSR is a read-only register that provides status information on the modem input signals. U0MSR[3:0] is cleared on U0MSR read. Note that modem signals have no direct effect on the UART operation. They facilitate the software implementation of modem signal operations.

**Table 131. UART Modem Status Register (U0MSR - address 0x4000 8018) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 0 | D CTS | | Delta CTS. Set upon state change of input $\overline{\text{CTS}}$. Cleared on a U0MSR read. | 0 |
| | | 0 | No change detected on modem input $\overline{\text{CTS}}$. | |
| | | 1 | State change detected on modem input $\overline{\text{CTS}}$. | |
| 1 | DDSR | | Delta DSR. Set upon state change of input $\overline{\text{DSR}}$. Cleared on a U0MSR read. | 0 |
| | | 0 | No change detected on modem input $\overline{\text{DSR}}$. | |
| | | 1 | State change detected on modem input $\overline{\text{DSR}}$. | |
| 2 | TERI | | Trailing Edge RI. Set upon low to high transition of input $\overline{\text{RI}}$. Cleared on a U0MSR read. | 0 |
| | | 0 | No change detected on modem input, $\overline{\text{RI}}$. | |
| | | 1 | Low-to-high transition detected on $\overline{\text{RI}}$. | |
| 3 | DDCD | | Delta DCD. Set upon state change of input $\overline{\text{DCD}}$. Cleared on a U0MSR read. | 0 |
| | | 0 | No change detected on modem input $\overline{\text{DCD}}$. | |
| | | 1 | State change detected on modem input $\overline{\text{DCD}}$. | |
| 4 | CTS | | Clear To Send State. Complement of input signal $\overline{\text{CTS}}$. This bit is connected to U0MCR[1] in modem loopback mode. | 0 |
| 5 | DSR | | Data Set Ready State. Complement of input signal $\overline{\text{DSR}}$. This bit is connected to U0MCR[0] in modem loopback mode. | 0 |
| 6 | RI | | Ring Indicator State. Complement of input $\overline{\text{RI}}$. This bit is connected to U0MCR[2] in modem loopback mode. | 0 |
| 7 | DCD | | Data Carrier Detect State. Complement of input $\overline{\text{DCD}}$. This bit is connected to U0MCR[3] in modem loopback mode. | 0 |
| 31:8 | - | - | Reserved | - |

### 10.5.11 UART Scratch Pad Register (U0SCR - 0x4000 801C)

The U0SCR has no effect on the UART operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the U0SCR has occurred.

**Table 132. UART Scratch Pad Register (U0SCR - address 0x4000 8014) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | Pad | A readable, writable byte. | 0x00 |
| 31:8 | - | Reserved | - |

### 10.5.12 UART Auto-baud Control Register (U0ACR - 0x4000 8020)

The UART Auto-baud Control Register (U0ACR) controls the process of measuring the incoming clock/data rate for the baud rate generation and can be read and written at user's discretion.

**Table 133. Auto baud Control Register (U0ACR - address 0x4000 8020) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | Start | | Start bit. This bit is automatically cleared after auto-baud completion. | 0 |
| | | 0 | Auto-baud stop (auto-baud is not running). | |
| | | 1 | Auto-baud start (auto-baud is running). Auto-baud run bit. This bit is automatically cleared after auto-baud completion. | |
| 1 | Mode | | Auto-baud mode select | 0 |
| | | 0 | Mode 0. | |
| | | 1 | Mode 1. | |
| 2 | AutoRestart | | Restart enable | 0 |
| | | 0 | No restart | |
| | | 1 | Restart in case of time-out (counter restarts at next UART Rx falling edge) | 0 |
| 7:3 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |
| 8 | ABEOIntClr | | End of auto-baud interrupt clear (write only accessible) | 0 |
| | | 0 | Writing a 0 has no impact. | |
| | | 1 | Writing a 1 will clear the corresponding interrupt in the U0IIR. | |
| 9 | ABTOIntClr | | Auto-baud time-out interrupt clear (write only accessible) | 0 |
| | | 0 | Writing a 0 has no impact. | |
| | | 1 | Writing a 1 will clear the corresponding interrupt in the U0IIR. | |
| 31:10 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |

### 10.5.13 Auto-baud

The UART auto-baud function can be used to measure the incoming baud rate based on the "AT" protocol (Hayes command). If enabled the auto-baud feature will measure the bit time of the receive data stream and set the divisor latch registers U0DLM and U0DLL accordingly.

Auto-baud is started by setting the U0ACR Start bit. Auto-baud can be stopped by clearing the U0ACR Start bit. The Start bit will clear once auto-baud has finished and reading the bit will return the status of auto-baud (pending/finished).

Two auto-baud measuring modes are available which can be selected by the U0ACR Mode bit. In Mode 0 the baud rate is measured on two subsequent falling edges of the UART Rx pin (the falling edge of the start bit and the falling edge of the least significant bit). In Mode 1 the baud rate is measured between the falling edge and the subsequent rising edge of the UART Rx pin (the length of the start bit).

The U0ACR AutoRestart bit can be used to automatically restart baud rate measurement if a time-out occurs (the rate measurement counter overflows). If this bit is set, the rate measurement will restart at the next falling edge of the UART Rx pin.

The auto-baud function can generate two interrupts.

- The U0IIR ABTOInt interrupt will get set if the interrupt is enabled (U0IER ABToIntEn is set and the auto-baud rate measurement counter overflows).
- The U0IIR ABEOInt interrupt will get set if the interrupt is enabled (U0IER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding U0ACR ABTOIntClr and ABEOIntEn bits.

The fractional baud rate generator must be disabled (DIVADDVAL = 0) during auto-baud. Also, when auto-baud is used, any write to U0DLM and U0DLL registers should be done before U0ACR register write. The minimum and the maximum baud rates supported by UART are function of UART_PCLK, number of data bits, stop bits and parity bits.

(2)

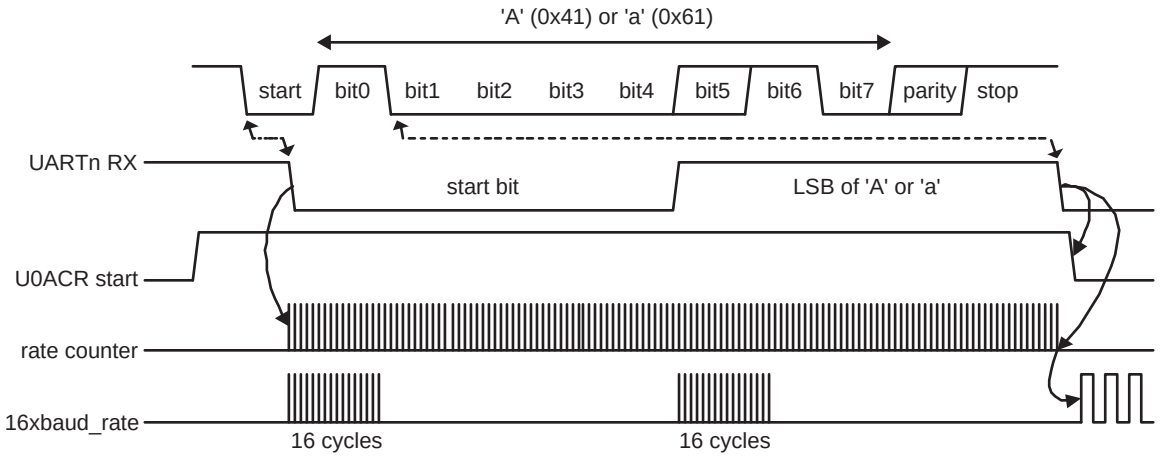
$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

## 10.5.14 Auto-baud modes

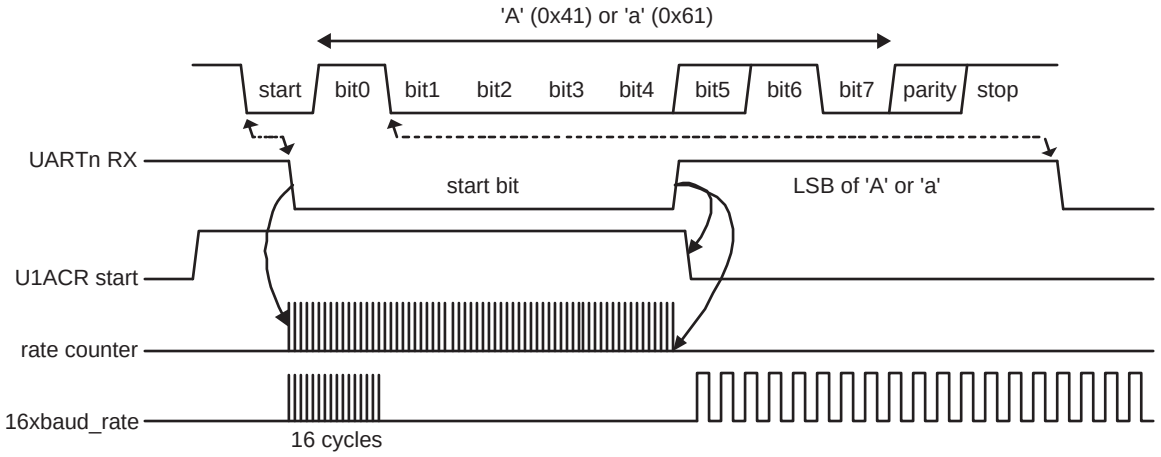
When the software is expecting an ”AT" command, it configures the UART with the expected character format and sets the U0ACR Start bit. The initial values in the divisor latches U0DLM and U0DLM don‘t care. Because of the ”A" or ”a" ASCII coding (”A" = 0x41, ”a" = 0x61), the UART Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the U0ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On U0ACR Start bit setting, the baud rate measurement counter is reset and the UART U0RSR is reset. The U0RSR baud rate is switched to the highest rate.
2. A falling edge on UART Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting UART_PCLK cycles.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the UART input clock, guaranteeing the start bit is stored in the U0RSR.

4. During the receipt of the start bit (and the character LSB for Mode = 0), the rate counter will continue incrementing with the pre-scaled UART input clock (UART_PCLK).
5. If Mode = 0, the rate counter will stop on next falling edge of the UART Rx pin. If Mode = 1, the rate counter will stop on the next rising edge of the UART Rx pin.
6. The rate counter is loaded into U0DLM/U0DLL and the baud rate will be switched to normal operation. After setting the U0DLM/U0DLL, the end of auto-baud interrupt U0IIR ABEOInt will be set, if enabled. The U0RSR will now continue receiving the remaining bits of the ”A/a" character.




a. Mode 0 (start bit and LSB are used for auto-baud)




b. Mode 1 (only start bit is used for auto-baud)

**Fig 18. Auto-baud a) mode 0 and b) mode 1 waveform**

### 10.5.15 UART Fractional Divider Register (U0FDR - 0x4000 8028)

The UART Fractional Divider Register (U0FDR) controls the clock pre-scaler for the baud rate generation and can be read and written at the user's discretion. This pre-scaler takes the APB clock and generates an output clock according to the specified fractional requirements.

**Important:** If the fractional divider is active (DIVADDVAL > 0) and DLM = 0, the value of the DLL register must be 3 or greater.

**Table 134. UART Fractional Divider Register (U0FDR - address 0x4000 8028) bit description**

| Bit | Function | Description | Reset value |
|---|---|---|---|
| 3:0 | DIVADDVAL | Baud rate generation pre-scaler divisor value. If this field is 0, fractional baud rate generator will not impact the UART baud rate. | 0 |
| 7:4 | MULVAL | Baud rate pre-scaler multiplier value. This field must be greater or equal 1 for UART to operate properly, regardless of whether the fractional baud rate generator is used or not. | 1 |
| 31:8 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of UART disabled making sure that UART is fully software and hardware compatible with UARTs not equipped with this feature.

The UART baud rate can be calculated as:

(3)

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

Where UART_PCLK is the peripheral clock, U0DLM and U0DLL are the standard UART baud rate divider registers, and DIVADDVAL and MULVAL are UART fractional baud rate generator specific parameters.

The value of MULVAL and DIVADDVAL should comply to the following conditions:

1. $1 \leq MULVAL \leq 15$
2. $0 \leq DIVADDVAL \leq 14$
3. DIVADDVAL< MULVAL

The value of the U0FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the U0FDR register value does not comply to these two requests, then the fractional divider output is undefined. If DIVADDVAL is zero then the fractional divider is disabled, and the clock will not be divided.

#### 10.5.15.1 Baud rate calculation

UART can operate with or without using the Fractional Divider. In real-life applications it is likely that the desired baud rate can be achieved using several different Fractional Divider settings. The following algorithm illustrates one way of finding a set of DLM, DLL, MULVAL, and DIVADDVAL values. Such set of parameters yields a baud rate with a relative error of less than 1.1% from the desired one.

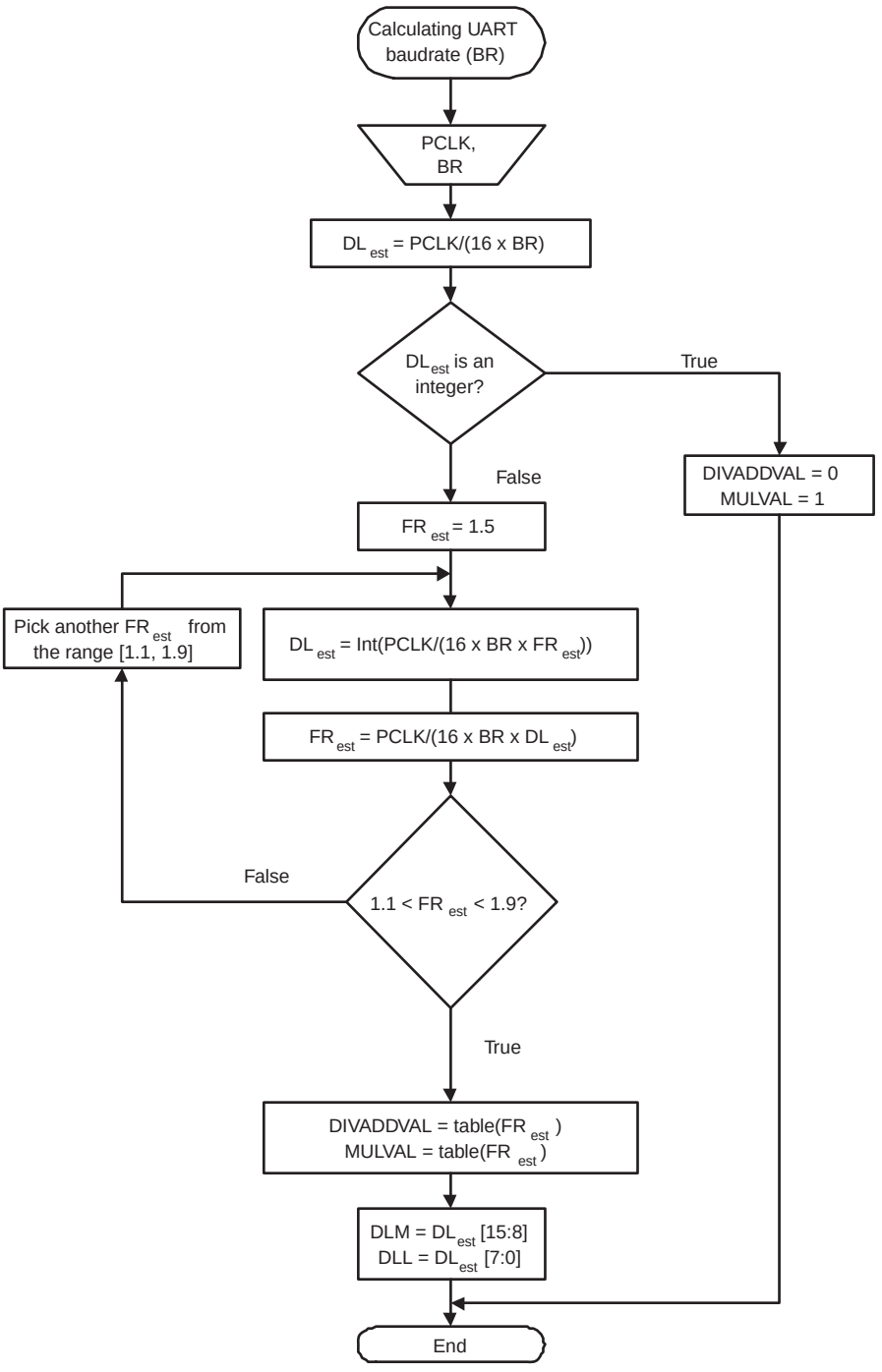



**Fig 19. Algorithm for setting UART dividers**

**Table 135. Fractional Divider setting look-up table**

| FR | DivAddVal/ MulVal | FR | DivAddVal/ MulVal | FR | DivAddVal/ MulVal | FR | DivAddVal/ MulVal |
|---|---|---|---|---|---|---|---|
| 1.000 | 0/1 | 1.250 | 1/4 | 1.500 | 1/2 | 1.750 | 3/4 |
| 1.067 | 1/15 | 1.267 | 4/15 | 1.533 | 8/15 | 1.769 | 10/13 |
| 1.071 | 1/14 | 1.273 | 3/11 | 1.538 | 7/13 | 1.778 | 7/9 |
| 1.077 | 1/13 | 1.286 | 2/7 | 1.545 | 6/11 | 1.786 | 11/14 |
| 1.083 | 1/12 | 1.300 | 3/10 | 1.556 | 5/9 | 1.800 | 4/5 |
| 1.091 | 1/11 | 1.308 | 4/13 | 1.571 | 4/7 | 1.818 | 9/11 |
| 1.100 | 1/10 | 1.333 | 1/3 | 1.583 | 7/12 | 1.833 | 5/6 |
| 1.111 | 1/9 | 1.357 | 5/14 | 1.600 | 3/5 | 1.846 | 11/13 |
| 1.125 | 1/8 | 1.364 | 4/11 | 1.615 | 8/13 | 1.857 | 6/7 |
| 1.133 | 2/15 | 1.375 | 3/8 | 1.625 | 5/8 | 1.867 | 13/15 |
| 1.143 | 1/7 | 1.385 | 5/13 | 1.636 | 7/11 | 1.875 | 7/8 |
| 1.154 | 2/13 | 1.400 | 2/5 | 1.643 | 9/14 | 1.889 | 8/9 |
| 1.167 | 1/6 | 1.417 | 5/12 | 1.667 | 2/3 | 1.900 | 9/10 |
| 1.182 | 2/11 | 1.429 | 3/7 | 1.692 | 9/13 | 1.909 | 10/11 |
| 1.200 | 1/5 | 1.444 | 4/9 | 1.700 | 7/10 | 1.917 | 11/12 |
| 1.214 | 3/14 | 1.455 | 5/11 | 1.714 | 5/7 | 1.923 | 12/13 |
| 1.222 | 2/9 | 1.462 | 6/13 | 1.727 | 8/11 | 1.929 | 13/14 |
| 1.231 | 3/13 | 1.467 | 7/15 | 1.733 | 11/15 | 1.933 | 14/15 |

#### 10.5.15.1.1 Example 1: UART_PCLK = 14.7456 MHz, BR = 9600

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

#### 10.5.15.1.2 Example 2: UART_PCLK = 12 MHz, BR = 115200

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

The closest value for FRest = 1.628 in the look-up Table 135 is FR = 1.625. It is equivalent to DIVADDVAL = 5 and MULVAL = 8.

Based on these findings, the suggested UART setup would be: DLM = 0, DLL = 4, DIVADDVAL = 5, and MULVAL = 8. According to Equation 3, the UART's baud rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

### 10.5.16 UART Transmit Enable Register (U0TER - 0x4000 8030)

In addition to being equipped with full hardware flow control (auto-cts and auto-rts mechanisms described above), U0TER enables implementation of software flow control. When TxEn = 1, UART transmitter will keep sending data as long as they are available. As soon as TxEn becomes 0, UART transmission will stop.

Although Table 136 describes how to use TxEn bit in order to achieve hardware flow control, it is strongly suggested to let UART hardware implemented auto flow control features take care of this, and limit the scope of TxEn to software flow control.

Table 136 describes how to use TXEn bit in order to achieve software flow control.

**Table 136. UART Transmit Enable Register (U0TER - address 0x4000 8030) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 6:0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7 | TXEN | When this bit is 1, as it is after a Reset, data written to the THR is output on the TXD pin as soon as any preceding data has been sent. If this bit cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again. In other words, a 0 in this bit blocks the transfer of characters from the THR or TX FIFO into the transmit shift register. Software can clear this bit when it detects that the a hardware-handshaking TX-permit signal ($\overline{CTS}$) has gone false, or with software handshaking, when it receives an XOFF character (DC3). Software can set this bit again when it detects that the TX-permit signal has gone true, or when it receives an XON (DC1) character. | 1 |
| 31:8 | - | Reserved | - |

## 10.5.17 UART RS485 Control register (U0RS485CTRL - 0x4000 804C)

The U0RS485CTRL register controls the configuration of the UART in RS-485/EIA-485 mode.

**Table 137. UART RS485 Control register (U0RS485CTRL - address 0x4000 804C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | NMMEN | | NMM enable. | 0 |
| | | 0 | RS-485/EIA-485 Normal Multidrop Mode (NMM) is disabled. | |
| | | 1 | RS-485/EIA-485 Normal Multidrop Mode (NMM) is enabled. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt. | |
| 1 | RXDIS | | Receiver enable. | 0 |
| | | 0 | The receiver is enabled. | |
| | | 1 | The receiver is disabled. | |
| 2 | AADEN | | AAD enable. | 0 |
| | | 0 | Auto Address Detect (AAD) is disabled. | |
| | | 1 | Auto Address Detect (AAD) is enabled. | |
| 3 | SEL | | Select direction control pin | 0 |
| | | 0 | If direction control is enabled (bit DCTRL = 1), pin $\overline{RTS}$ is used for direction control. | |
| | | 1 | If direction control is enabled (bit DCTRL = 1), pin $\overline{DTR}$ is used for direction control. | |

**Table 137. UART RS485 Control register (U0RS485CTRL - address 0x4000 804C) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4 | DCTRL | | Auto direction control enable. | 0 |
| | | 0 | Disable Auto Direction Control. | |
| | | 1 | Enable Auto Direction Control. | |
| 5 | OINV | | Polarity control. This bit reverses the polarity of the direction control signal on the RTS (or DTR) pin. | 0 |
| | | 0 | The direction control pin will be driven to logic 0 when the transmitter has data to be sent. It will be driven to logic 1 after the last bit of data has been transmitted. | |
| | | 1 | The direction control pin will be driven to logic 1 when the transmitter has data to be sent. It will be driven to logic 0 after the last bit of data has been transmitted. | |
| 31:6 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 10.5.18 UART RS485 Address Match register (U0RS485ADRMATCH - 0x4000 8050)

The U0RS485ADRMATCH register contains the address match value for RS-485/EIA-485 mode.

**Table 138. UART RS485 Address Match register (U0RS485ADRMATCH - address 0x4000 8050) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | ADRMATCH | Contains the address match value. | 0x00 |
| 31:8 | - | Reserved | - |

## 10.5.19 UART1 RS485 Delay value register (U0RS485DLY - 0x4000 8054)

The user may program the 8-bit RS485DLY register with a delay between the last stop bit leaving the TXFIFO and the de-assertion of RTS (or DTR). This delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

**Table 139. UART RS485 Delay value register (U0RS485DLY - address 0x4000 8054) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | DLY | Contains the direction control (RTS or DTR) delay value. This register works in conjunction with an 8-bit counter. | 0x00 |
| 31:8 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 10.5.20 RS-485/EIA-485 modes of operation

The RS-485/EIA-485 feature allows the UART to be configured as an addressable slave. The addressable slave is one of multiple slaves controlled by a single master.

The UART master transmitter will identify an address character by setting the parity (9th) bit to ‘1’. For data characters, the parity bit is set to ‘0’.

Each UART slave receiver can be assigned a unique address. The slave can be programmed to either manually or automatically reject data following an address which is not theirs.

### RS-485/EIA-485 Normal Multidrop Mode (NMM)

Setting the RS485CTRL bit 0 enables this mode. In this mode, an address is detected when a received byte causes the UART to set the parity error and generate an interrupt.

If the receiver is disabled (RS485CTRL bit 1 = ‘1’), any received data bytes will be ignored and will not be stored in the RXFIFO. When an address byte is detected (parity bit = ‘1’) it will be placed into the RXFIFO and an Rx Data Ready Interrupt will be generated. The processor can then read the address byte and decide whether or not to enable the receiver to accept the following data.

While the receiver is enabled (RS485CTRL bit 1 =’0’), all received bytes will be accepted and stored in the RXFIFO regardless of whether they are data or address. When an address character is received a parity error interrupt will be generated and the processor can decide whether or not to disable the receiver.

### RS-485/EIA-485 Auto Address Detection (AAD) mode

When both RS485CTRL register bits 0 (9-bit mode enable) and 2 (AAD mode enable) are set, the UART is in auto address detect mode.

In this mode, the receiver will compare any address byte received (parity = ‘1’) to the 8-bit value programmed into the RS485ADRMATCH register.

If the receiver is disabled (RS485CTRL bit 1 = ‘1’), any received byte will be discarded if it is either a data byte OR an address byte which fails to match the RS485ADRMATCH value.

When a matching address character is detected it will be pushed onto the RXFIFO along with the parity bit, and the receiver will be automatically enabled (RS485CTRL bit 1 will be cleared by hardware). The receiver will also generate an Rx Data Ready Interrupt.

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

### RS-485/EIA-485 Auto Direction Control

RS485/EIA-485 mode includes the option of allowing the transmitter to automatically control the state of the DIR pin as a direction control output signal.

Setting RS485CTRL bit 4 = ‘1’ enables this feature.

Direction control, if enabled, will use the $\overline{\text{RTS}}$ pin when RS485CTRL bit 3 = ‘0’. It will use the $\overline{\text{DTR}}$ pin when RS485CTRL bit 3 = ‘1’.

When Auto Direction Control is enabled, the selected pin will be asserted (driven LOW) when the CPU writes data into the TXFIFO. The pin will be de-asserted (driven HIGH) once the last bit of data has been transmitted. See bits 4 and 5 in the RS485CTRL register.

The RS485CTRL bit 4 takes precedence over all other mechanisms controlling the direction control pin with the exception of loopback mode.

### RS485/EIA-485 driver delay time

The driver delay time is the delay between the last stop bit leaving the TXFIFO and the de-assertion of $\overline{\text{RTS}}$. This delay time can be programmed in the 8-bit RS485DLY register. The delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be used.

### RS485/EIA-485 output inversion

The polarity of the direction control signal on the $\overline{\text{RTS}}$ (or $\overline{\text{DTR}}$) pins can be reversed by programming bit 5 in the U0RS485CTRL register. When this bit is set, the direction control pin will be driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin will be driven to logic 0 after the last bit of data has been transmitted.

## 10.6 Architecture

The architecture of the UART is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART.

The UART receiver block, U0RX, monitors the serial input line, RXD, for valid input. The UART RX Shift Register (U0RSR) accepts valid characters via RXD. After a valid character is assembled in the U0RSR, it is passed to the UART RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART transmitter block, U0TX, accepts data written by the CPU or host and buffers the data in the UART TX Holding Register FIFO (U0THR). The UART TX Shift Register (U0TSR) reads the data stored in the U0THR and assembles the data to transmit via the serial output pin, TXD1.

The UART Baud Rate Generator block, U0BRG, generates the timing enables used by the UART TX block. The U0BRG clock input source is UART_PCLK. The main clock is divided down per the divisor specified in the U0DLL and U0DLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

The interrupt interface contains registers U0IER and U0IIR. The interrupt interface receives several one clock wide enables from the U0TX and U0RX blocks.

Status information from the U0TX and U0RX is stored in the U0LSR. Control information for the U0TX and U0RX is stored in the U0LCR.

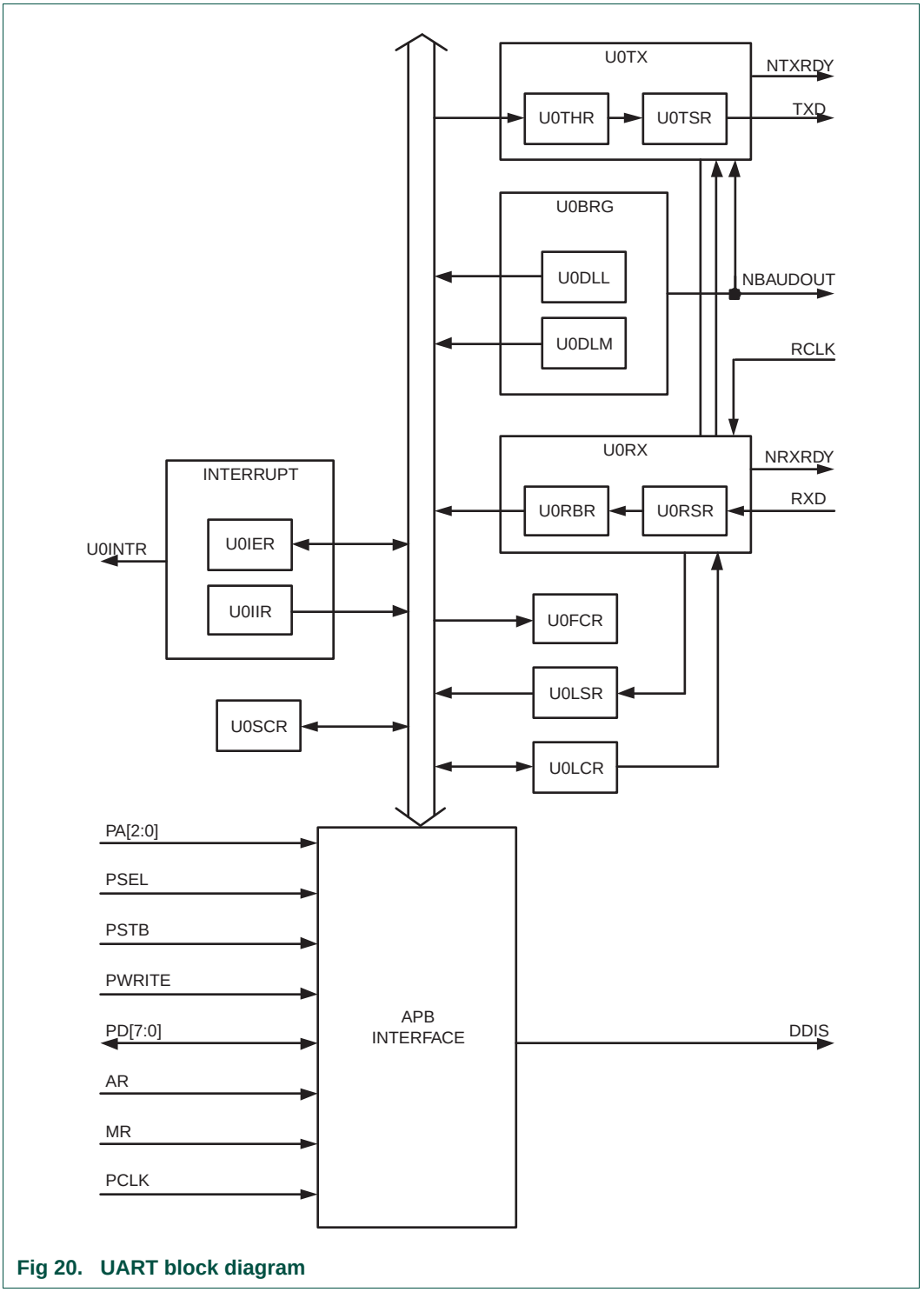



**Fig 20. UART block diagram**

## 11.1 How to read this chapter

The SPI blocks are identical for all LPC111x and LPC11Cxx parts. The second SPI block, SPI1, is available on LQFP48 and PLCC44 packages. SPI1 is not available on HVQFN33 packages.

**Remark:** Both SPI blocks include the full SSP feature set, and all register names use the SSP prefix.

## 11.2 Basic configuration

The SPI0/1 are configured using the following registers:

1. Pins: The SPI pins must be configured in the IOCONFIG register block. In addition, use the IOCON_LOC register (see Section 7.4) to select a location for the SCK0 function.
2. Power: In the SYSAHBCLKCTRL register, set bit 11 and bit 18 (Table 20).
3. Peripheral clock: Enable the SPI0/1 peripheral clock by writing to the SSP0/1CLKDIV registers (Section 3.5.15 and Section 3.5.17).
4. Reset: Before accessing the SPI blocks, ensure that the SSP_RST_N bits (bit 0 and bit 2) in the PRESETCTRL register (Table 8) is set to 1. This de-asserts the reset signal to the SPI blocks.

## 11.3 Features

- Compatible with Motorola SPI, 4-wire TI SSI, and National Semiconductor Microwire buses.
- Synchronous Serial Communication.
- Supports master or slave operation.
- Eight-frame FIFOs for both transmit and receive.
- 4-bit to 16-bit frame.

## 11.4 General description

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

The LPC111x/LPC11Cxx has two SPI/Synchronous Serial Port controllers.

## 11.5 Pin description

**Table 140. SPI pin descriptions**

| Pin name | Type | Interface pin name/function | | | Pin description |
|---|---|---|---|---|---|
| | | SPI | SSI | Microwire | |
| SCK0/1 | I/O | SCK | CLK | SK | **Serial Clock.** SCK/CLK/SK is a clock signal used to synchronize the transfer of data. It is driven by the master and received by the slave. When SPI/SSP interface is used, the clock is programmable to be active-high or active-low, otherwise it is always active-high. SCK only switches during a data transfer. Any other time, the SPI/SSP interface either holds it in its inactive state or does not drive it (leaves it in high-impedance state). |
| SSEL0/1 | I/O | SSEL | FS | CS | **Frame Sync/Slave Select.** When the SPI/SSP interface is a bus master, it drives this signal to an active state before the start of serial data and then releases it to an inactive state after the data has been sent.The active state of this signal can be high or low depending upon the selected bus and mode. When the SPI/SSP interface is a bus slave, this signal qualifies the presence of data from the Master according to the protocol in use.<br>When there is just one bus master and one bus slave, the Frame Sync or Slave Select signal from the Master can be connected directly to the slave's corresponding input. When there is more than one slave on the bus, further qualification of their Frame Select/Slave Select inputs will typically be necessary to prevent more than one slave from responding to a transfer. |
| MISO0/1 | I/O | MISO | DR(M)<br>DX(S) | SI(M)<br>SO(S) | **Master In Slave Out.** The MISO signal transfers serial data from the slave to the master. When the SPI/SSP is a slave, serial data is output on this signal. When the SPI/SSP is a master, it clocks in serial data from this signal. When the SPI/SSP is a slave and is not selected by FS/SSEL, it does not drive this signal (leaves it in high-impedance state). |
| MOSI0/1 | I/O | MOSI | DX(M)<br>DR(S) | SO(M)<br>SI(S) | **Master Out Slave In.** The MOSI signal transfers serial data from the master to the slave. When the SPI/SSP is a master, it outputs serial data on this signal. When the SPI/SSP is a slave, it clocks in serial data from this signal. |

**Remark:** The SCK0 function is multiplexed to three different pin locations (two locations on the HVQFN package). Use the IOCON_LOC register (see Section 7.4) to select a physical location for the SCK0 function in addition to selecting the function in the IOCON registers. The SCK1 pin is not multiplexed.

## 11.6 Register description

The register addresses of the SPI controllers are shown in Table 141 and Table 142.

The reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

**Remark:** Register names use the SSP prefix to indicate that the SPI controllers have full SSP capabilities.

**Table 141. Register overview: SPI0 (base address 0x4004 0000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| SSP0CR0 | R/W | 0x000 | Control Register 0. Selects the serial clock rate, bus type, and data size. | 0 |
| SSP0CR1 | R/W | 0x004 | Control Register 1. Selects master/slave and other modes. | 0 |
| SSP0DR | R/W | 0x008 | Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO. | 0 |
| SSP0SR | RO | 0x00C | Status Register | 0x0000 0003 |
| SSP0CPSR | R/W | 0x010 | Clock Prescale Register | 0 |
| SSP0IMSC | R/W | 0x014 | Interrupt Mask Set and Clear Register | 0 |
| SSP0RIS | RO | 0x018 | Raw Interrupt Status Register | 0x0000 0008 |
| SSP0MIS | RO | 0x01C | Masked Interrupt Status Register | 0 |
| SSP0ICR | WO | 0x020 | SSPICR Interrupt Clear Register | NA |

**Table 142. Register overview: SPI1 (base address 0x4005 8000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| SSP1CR0 | R/W | 0x000 | Control Register 0. Selects the serial clock rate, bus type, and data size. | 0 |
| SSP1CR1 | R/W | 0x004 | Control Register 1. Selects master/slave and other modes. | 0 |
| SSP1DR | R/W | 0x008 | Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO. | 0 |
| SSP1SR | RO | 0x00C | Status Register | 0x0000 0003 |
| SSP1CPSR | R/W | 0x010 | Clock Prescale Register | 0 |
| SSP1IMSC | R/W | 0x014 | Interrupt Mask Set and Clear Register | 0 |
| SSP1RIS | RO | 0x018 | Raw Interrupt Status Register | 0x0000 0008 |
| SSP1MIS | RO | 0x01C | Masked Interrupt Status Register | 0 |
| SSP1ICR | WO | 0x020 | SSPICR Interrupt Clear Register | NA |

### 11.6.1 SPI/SSP Control Register 0

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

Table 143: SPI/SSP Control Register 0 (SSP0CR0 - address 0x4004 0000, SSP1CR0 - address 0x4005 8000) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 3:0 | DSS | | Data Size Select. This field controls the number of bits transferred in each frame. Values 0000-0010 are not supported and should not be used. | 0000 |
| | | 0x3 | 4-bit transfer | |
| | | 0x4 | 5-bit transfer | |
| | | 0x5 | 6-bit transfer | |
| | | 0x6 | 7-bit transfer | |
| | | 0x7 | 8-bit transfer | |
| | | 0x8 | 9-bit transfer | |
| | | 0x9 | 10-bit transfer | |
| | | 0xA | 11-bit transfer | |
| | | 0xB | 12-bit transfer | |
| | | 0xC | 13-bit transfer | |
| | | 0xD | 14-bit transfer | |
| | | 0xE | 15-bit transfer | |
| | | 0xF | 16-bit transfer | |
| 5:4 | FRF | | Frame Format. | 00 |
| | | 0x0 | SPI | |
| | | 0x1 | TI | |
| | | 0x2 | Microwire | |
| | | 0x3 | This combination is not supported and should not be used. | |
| 6 | CPOL | | Clock Out Polarity. This bit is only used in SPI mode. | 0 |
| | | 0 | SPI controller maintains the bus clock low between frames. | |
| | | 1 | SPI controller maintains the bus clock high between frames. | |
| 7 | CPHA | | Clock Out Phase. This bit is only used in SPI mode. | 0 |
| | | 0 | SPI controller captures serial data on the first clock transition of the frame, that is, the transition **away from** the inter-frame state of the clock line. | |
| | | 1 | SPI controller captures serial data on the second clock transition of the frame, that is, the transition **back to** the inter-frame state of the clock line. | |
| 15:8 | SCR | | Serial Clock Rate. The number of prescaler output clocks per bit on the bus, minus one. Given that CPSDVSR is the prescale divider, and the APB clock PCLK clocks the prescaler, the bit frequency is PCLK / (CPSDVSR $\times$ [SCR+1]). | 0x00 |
| 31:16 | - | - | Reserved | - |

## 11.6.2 SPI/SSP0 Control Register 1

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

**Table 144: SPI/SSP Control Register 1 (SSP0CR1 - address 0x4004 0004, SSP1CR1 - address 0x4005 8004) bit description**

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

### 11.6.3 SPI/SSP Data Register

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

**Table 145: SPI/SSP Data Register (SSP0DR - address 0x4004 0008, SSP1DR - address 0x4005 8008) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 15:0 | DATA | **Write:** software can write data to be sent in a future frame to this register whenever the TNF bit in the Status register is 1, indicating that the Tx FIFO is not full. If the Tx FIFO was previously empty and the SPI controller is not busy on the bus, transmission of the data will begin immediately. Otherwise the data written to this register will be sent as soon as all previous data has been sent (and received). If the data length is less than 16 bit, software must right-justify the data written to this register.<br>**Read:** software can read data from this register whenever the RNE bit in the Status register is 1, indicating that the Rx FIFO is not empty. When software reads this register, the SPI controller returns data from the least recent frame in the Rx FIFO. If the data length is less than 16 bit, the data is right-justified in this field with higher order bits filled with 0s. | 0x0000 |
| 31:16 | - | Reserved. | - |

### 11.6.4 SPI/SSP Status Register

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

**Table 146: SPI/SSP Status Register (SSP0SR - address 0x4004 000C, SSP1SR - address 0x4005 800C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | TFE | Transmit FIFO Empty. This bit is 1 is the Transmit FIFO is empty, 0 if not. | 1 |
| 1 | TNF | Transmit FIFO Not Full. This bit is 0 if the Tx FIFO is full, 1 if not. | 1 |
| 2 | RNE | Receive FIFO Not Empty. This bit is 0 if the Receive FIFO is empty, 1 if not. | 0 |
| 3 | RFF | Receive FIFO Full. This bit is 1 if the Receive FIFO is full, 0 if not. | 0 |
| 4 | BSY | Busy. This bit is 0 if the SPI controller is idle, 1 if it is currently sending/receiving a frame and/or the Tx FIFO is not empty. | 0 |
| 31:5 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 11.6.5 SPI/SSP Clock Prescale Register

This register controls the factor by which the Prescaler divides the SPI peripheral clock SPI_PCLK to yield the prescaler clock that is, in turn, divided by the SCR factor in the SSPCR0 registers, to determine the bit clock.

**Table 147: SPI/SSP Clock Prescale Register (SSP0CPSR - address 0x4004 0010, SSP1CPSR - address 0x4005 8010) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | CPSDVSR | This even value between 2 and 254, by which SPI_PCLK is divided to yield the prescaler output clock. Bit 0 always reads as 0. | 0 |
| 31:8 | - | Reserved. | - |

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

In Slave mode, the SPI clock rate provided by the master must not exceed 1/12 of the SPI peripheral clock selected in Section 3.5.15. The content of the SSPnCPSR register is not relevant.

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

### 11.6.6 SPI/SSP Interrupt Mask Set/Clear Register

This register controls whether each of the four possible interrupt conditions in the SPI controller are enabled. Note that ARM uses the word “masked” in the opposite sense from classic computer terminology, in which “masked” meant “disabled”. ARM uses the word “masked” to mean “enabled”. To avoid confusion we will not use the word “masked”.

**Table 148: SPI/SSP Interrupt Mask Set/Clear register (SSP0IMSC - address 0x4004 0014, SSP1IMSC - address 0x4005 8014) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | RORIM | Software should set this bit to enable interrupt when a Receive Overrun occurs, that is, when the Rx FIFO is full and another frame is completely received. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs. | 0 |
| 1 | RTIM | Software should set this bit to enable interrupt when a Receive Time-out condition occurs. A Receive Time-out occurs when the Rx FIFO is not empty, and no has not been read for a time-out period. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]). | 0 |
| 2 | RXIM | Software should set this bit to enable interrupt when the Rx FIFO is at least half full. | 0 |
| 3 | TXIM | Software should set this bit to enable interrupt when the Tx FIFO is at least half empty. | 0 |
| 31:4 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 11.6.7 SPI/SSP Raw Interrupt Status Register

This read-only register contains a 1 for each interrupt condition that is asserted, regardless of whether or not the interrupt is enabled in the SSPIMSC registers.

**Table 149: SPI/SSP Raw Interrupt Status register (SSP0RIS - address 0x4004 0018, SSP1RIS - address 0x4005 8018) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | RORRIS | This bit is 1 if another frame was completely received while the RxFIFO was full. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs. | 0 |
| 1 | RTRIS | This bit is 1 if the Rx FIFO is not empty, and has not been read for a time-out period. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]). | 0 |
| 2 | RXRIS | This bit is 1 if the Rx FIFO is at least half full. | 0 |
| 3 | TXRIS | This bit is 1 if the Tx FIFO is at least half empty. | 1 |
| 31:4 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 11.6.8 SPI/SSP Masked Interrupt Status Register

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

**Table 150: SPI/SSP Masked Interrupt Status register (SSP0MIS - address 0x4004 001C, SSP1MIS - address 0x4005 801C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | RORMIS | This bit is 1 if another frame was completely received while the RxFIFO was full, and this interrupt is enabled. | 0 |
| 1 | RTMIS | This bit is 1 if the Rx FIFO is not empty, has not been read for a time-out period, and this interrupt is enabled. The time-out period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]). | 0 |
| 2 | RXMIS | This bit is 1 if the Rx FIFO is at least half full, and this interrupt is enabled. | 0 |
| 3 | TXMIS | This bit is 1 if the Tx FIFO is at least half empty, and this interrupt is enabled. | 0 |
| 31:4 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 11.6.9 SPI/SSP Interrupt Clear Register

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

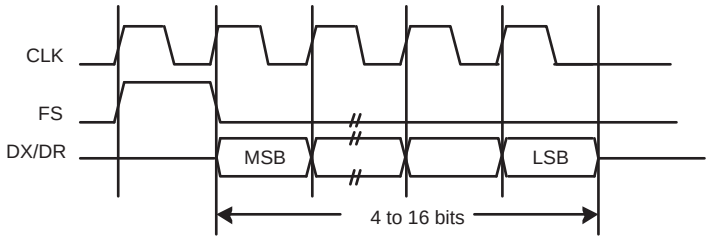
**Table 151: SPI/SSP interrupt Clear Register (SSP0ICR - address 0x4004 0020, SSP1ICR - address 0x4005 8020) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | RORIC | Writing a 1 to this bit clears the “frame was received when RxFIFO was full” interrupt. | NA |
| 1 | RTIC | Writing a 1 to this bit clears the Rx FIFO was not empty and has not been read for a timeout period interrupt. The timeout period is the same for master and slave modes and is determined by the SSP bit rate: 32 bits at PCLK / (CPSDVSR × [SCR+1]). | NA |
| 31:2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

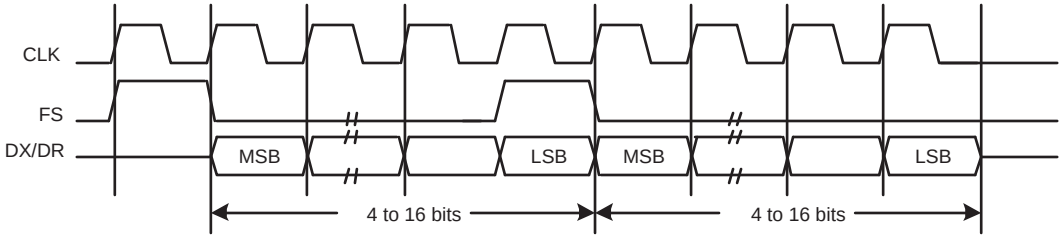
## 11.7 Functional description

### 11.7.1 Texas Instruments synchronous serial frame format

Figure 21 shows the 4-wire Texas Instruments synchronous serial frame format supported by the SPI module.




a. Single frame transfer




b. Continuous/back-to-back frames transfer

**Fig 21. Texas Instruments Synchronous Serial Frame Format: a) Single and b) Continuous/back-to-back Two Frames Transfer**

For device configured as a master in this mode, CLK and FS are forced LOW, and the transmit data line DX is in 3-state mode whenever the SSP is idle. Once the bottom entry of the transmit FIFO contains data, FS is pulsed HIGH for one CLK period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of CLK, the MSB of the 4-bit to 16-bit data frame is shifted out on the DX pin. Likewise, the MSB of the received data is shifted onto the DR pin by the off-chip serial slave device.

Both the SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each CLK. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of CLK after the LSB has been latched.

### 11.7.2 SPI frame format

The SPI interface is a four-wire interface where the SSEL signal behaves as a slave select. The main feature of the SPI format is that the inactive state and phase of the SCK signal are programmable through the CPOL and CPHA bits within the SSPCR0 control register.

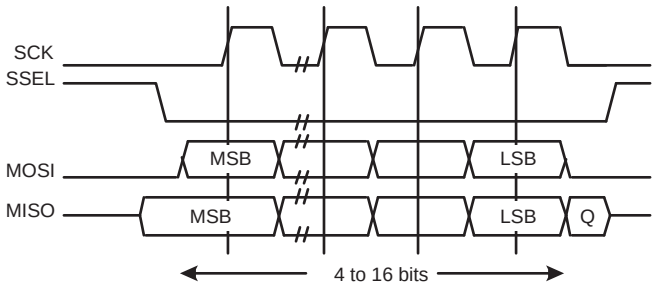
#### 11.7.2.1 Clock Polarity (CPOL) and Phase (CPHA) control

When the CPOL clock polarity control bit is LOW, it produces a steady state low value on the SCK pin. If the CPOL clock polarity control bit is HIGH, a steady state high value is placed on the CLK pin when data is not being transferred.

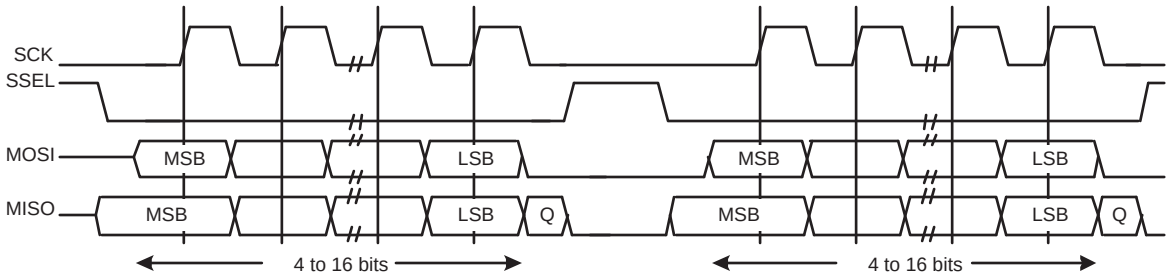
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is LOW, data is captured on the first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

### 11.7.2.2 SPI format with CPOL=0,CPHA=0

Single and continuous transmission signal sequences for SPI format with CPOL = 0, CPHA = 0 are shown in Figure 22.




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




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

**Fig 22. SPI frame format with CPOL=0 and CPHA=0 (a) Single and b) Continuous Transfer)**

In this configuration, during idle periods:

- The CLK signal is forced LOW.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

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

One half SCK period later, valid master data is transferred to the MOSI pin. Now that both the master and slave data have been set, the SCK master clock pin goes HIGH after one further half SCK period.

The data is captured on the rising and propagated on the falling edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

### 11.7.2.3 SPI format with CPOL=0,CPHA=1

The transfer signal sequence for SPI format with CPOL = 0, CPHA = 1 is shown in Figure 23, which covers both single and continuous transfers.

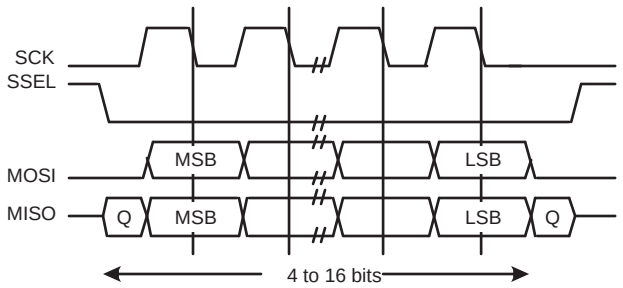



**Fig 23. SPI frame format with CPOL=0 and CPHA=1**

In this configuration, during idle periods:

- The CLK signal is forced LOW.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master's MOSI pin is enabled. After a further one half SCK period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the SCK is enabled with a rising edge transition.

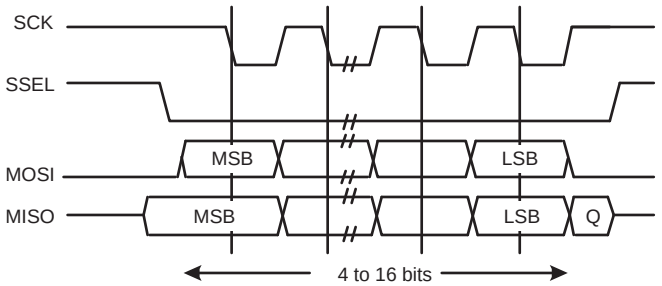
Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

In the case of a single word transfer, after all bits have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

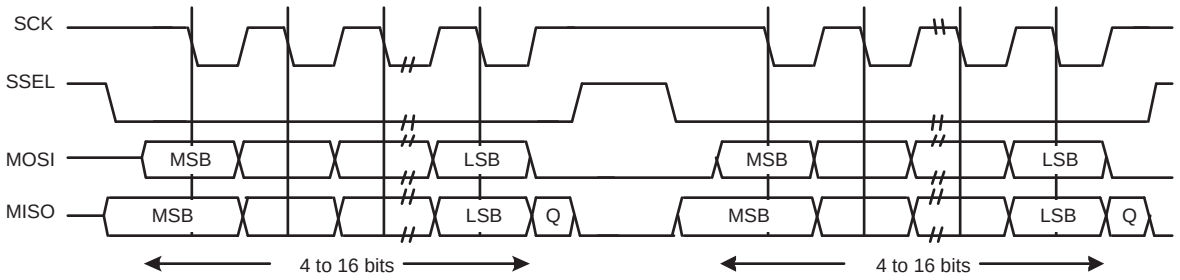
For continuous back-to-back transfers, the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

### 11.7.2.4 SPI format with CPOL = 1,CPHA = 0

Single and continuous transmission signal sequences for SPI format with CPOL=1, CPHA=0 are shown in Figure 24.




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




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

**Fig 24. SPI frame format with CPOL = 1 and CPHA = 0 (a) Single and b) Continuous Transfer)**

In this configuration, during idle periods:

- The CLK signal is forced HIGH.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW, which causes slave data to be immediately transferred onto the MISO line of the master. Master's MOSI pin is enabled.

One half period later, valid master data is transferred to the MOSI line. Now that both the master and slave data have been set, the SCK master clock pin becomes LOW after one further half SCK period. This means that data is captured on the falling edges and be propagated on the rising edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word are transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

#### 11.7.2.5 SPI format with CPOL = 1,CPHA = 1

The transfer signal sequence for SPI format with CPOL = 1, CPHA = 1 is shown in Figure 25, which covers both single and continuous transfers.

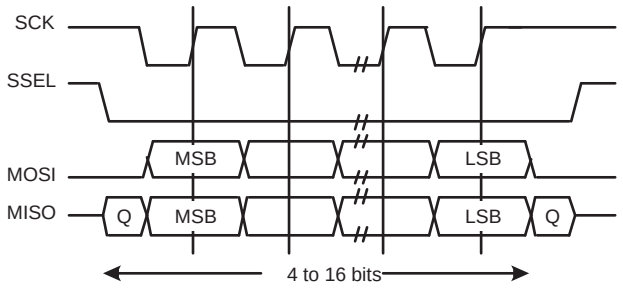



**Fig 25. SPI Frame Format with CPOL = 1 and CPHA = 1**

In this configuration, during idle periods:

- The CLK signal is forced HIGH.
- SSEL is forced HIGH.
- The transmit MOSI/MISO pad is in high impedance.

If the SPI/SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master's MOSI is enabled. After a further one half SCK period, both master and slave data are enabled onto their respective transmission lines. At the same time, the SCK is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the SCK signal.

After all bits have been transferred, in the case of a single word transmission, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured. For continuous back-to-back transmissions, the SSEL pins remains in its active LOW state, until the final bit of the last word has been captured, and then returns to its idle state as described above. In general, for continuous back-to-back transfers the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

### 11.7.3 Semiconductor Microwire frame format

Figure 26 shows the Microwire frame format for a single frame. Figure 27 shows the same format when back-to-back frames are transmitted.

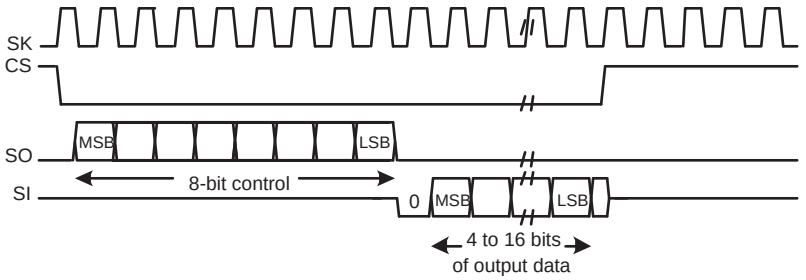



**Fig 26. Microwire frame format (single transfer)**

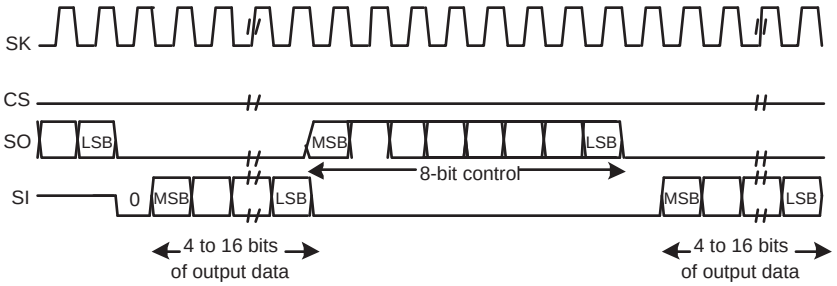



**Fig 27. Microwire frame format (continuos transfers)**

Microwire format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SPI/SSP to the off-chip slave device. During this transmission, no incoming data is received by the SPI/SSP. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bit in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- The SK signal is forced LOW.
- CS is forced HIGH.
- The transmit data line SO is arbitrarily forced LOW.

A transmission is triggered by writing a control byte to the transmit FIFO.The falling edge of CS causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the SO pin. CS remains LOW for the duration of the frame transmission. The SI pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SK. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SPI/SSP. Each bit is driven onto SI line on the falling edge of SK. The SPI/SSP in

turn latches each bit on the rising edge of SK. At the end of the frame, for single transfers, the CS signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

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

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

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

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

Figure 28 illustrates these setup and hold time requirements. With respect to the SK rising edge on which the first bit of receive data is to be sampled by the SPI/SSP slave, CS must have a setup of at least two times the period of SK on which the SPI/SSP operates. With respect to the SK rising edge previous to this edge, CS must have a hold of at least one SK period.

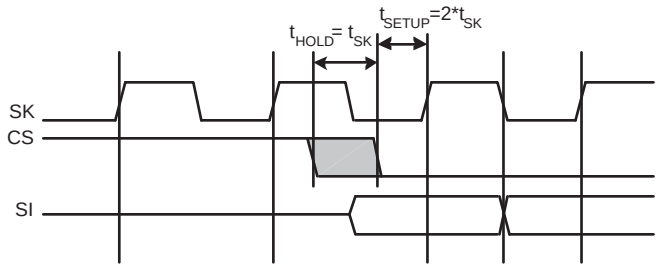



**Fig 28. Microwire frame format setup and hold details**

## 12.1 How to read this chapter

The I²C-bus block is identical for all LPC111x and LPC11Cxx parts.

## 12.2 Basic configuration

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

1. Pins: The I2C pin functions and the I2C mode are configured in the IOCONFIG register block (Section 7.4, Table 65 and Table 66).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 5 (Table 20).
3. Reset: Before accessing the I2C block, ensure that the I2C_RST_N bit (bit 1) in the PRESETCTRL register (Table 8) is set to 1. This de-asserts the reset signal to the I2C block.

## 12.3 Features

- Standard I²C-compliant bus interfaces may be configured as Master, Slave, or Master/Slave.
- Arbitration is handled between simultaneously transmitting masters without corruption of serial data on the bus.
- Programmable clock allows adjustment of I²C transfer rates.
- Data transfer is bidirectional between masters and slaves.
- Serial clock synchronization allows devices with different bit rates to communicate via one serial bus.
- Serial clock synchronization is used as a handshake mechanism to suspend and resume serial transfer.
- Supports Fast-mode Plus.
- Optional recognition of up to four distinct slave addresses.
- Monitor mode allows observing all I²C-bus traffic, regardless of slave address.
- I²C-bus can be used for test and diagnostic purposes.
- The I²C-bus contains a standard I²C-compliant bus interface with two pins.

## 12.4 Applications

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

## 12.5 General description

A typical I²C-bus configuration is shown in Figure 29. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I²C-bus:

- Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte.
- Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. Next follows the data bytes transmitted by the slave to the master. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a “not acknowledge” is returned. The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a Repeated START condition. Since a Repeated START condition is also the beginning of the next serial transfer, the $I^2C$ bus will not be released.

The $I^2C$ interface is byte oriented and has four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

The $I^2C$ interface complies with the entire $I^2C$ specification, supporting the ability to turn power off to the ARM Cortex-M0 without interfering with other devices on the same $I^2C$-bus.

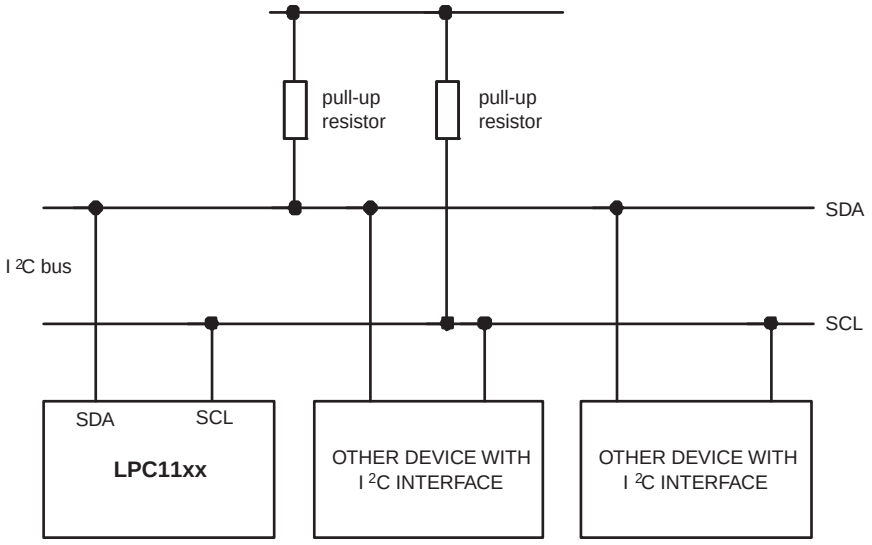



**Fig 29. $I^2C$-bus configuration**

### 12.5.1 $I^2C$ Fast-mode Plus

Fast-Mode Plus supports a 1 Mbit/sec transfer rate to communicate with the $I^2C$-bus products which NXP Semiconductors is now providing.

## 12.6 Pin description

**Table 152. I²C-bus pin description**

| Pin | Type | Description |
|---|---|---|
| SDA | Input/Output | I²C Serial Data |
| SCL | Input/Output | I²C Serial Clock |

The I²C-bus pins must be configured through the IOCON_PIO0_4 (Table 65) and IOCON_PIO0_5 (Table 66) registers for Standard/ Fast-mode or Fast-mode Plus. In Fast-mode Plus, rates above 400 kHz and up to 1 MHz may be selected. The I²C-bus pins are open-drain outputs and fully compatible with the I²C-bus specification.

## 12.7 Register description

**Table 153. Register overview: I²C (base address 0x4000 0000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| I2C0CONSET | R/W | 0x000 | **I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is set. Writing a zero has no effect on the corresponding bit in the I²C control register. | 0x00 |
| I2C0STAT | RO | 0x004 | **I2C Status Register.** During I²C operation, this register provides detailed status codes that allow software to determine the next action needed. | 0xF8 |
| I2C0DAT | R/W | 0x008 | **I2C Data Register.** During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register. | 0x00 |
| I2C0ADR0 | R/W | 0x00C | **I2C Slave Address Register 0.** Contains the 7-bit slave address for operation of the I²C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address. | 0x00 |
| I2C0SCLH | R/W | 0x010 | **SCH Duty Cycle Register High Half Word.** Determines the high time of the I²C clock. | 0x04 |
| I2C0SCLL | R/W | 0x014 | **SCL Duty Cycle Register Low Half Word.** Determines the low time of the I²C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I²C master and certain times used in slave mode. | 0x04 |
| I2C0CONCLR | WO | 0x018 | **I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I²C control register is cleared. Writing a zero has no effect on the corresponding bit in the I²C control register. | NA |
| I2C0MMCTRL | R/W | 0x01C | **Monitor mode control register.** | 0x00 |
| I2C0ADR1 | R/W | 0x020 | **I2C Slave Address Register 1.** Contains the 7-bit slave address for operation of the I²C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address. | 0x00 |
| I2C0ADR2 | R/W | 0x024 | **I2C Slave Address Register 2.** Contains the 7-bit slave address for operation of the I²C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address. | 0x00 |

**Table 153. Register overview: I²C (base address 0x4000 0000)** *…continued*

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| I2C0ADR3 | R/W | 0x028 | **I2C Slave Address Register 3.** Contains the 7-bit slave address for operation of the I²C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address. | 0x00 |
| I2C0DATA_BUFFER | RO | 0x02C | **Data buffer register.** The contents of the 8 MSBs of the I2DAT shift register will be transferred to the DATA_BUFFER automatically after every nine bits (8 bits of data plus ACK or NACK) has been received on the bus. | 0x00 |
| I2C0MASK0 | R/W | 0x030 | **I2C Slave address mask register 0**. This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’). | 0x00 |
| I2C0MASK1 | R/W | 0x034 | **I2C Slave address mask register 1**. This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’). | 0x00 |
| I2C0MASK2 | R/W | 0x038 | **I2C Slave address mask register 2**. This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’). | 0x00 |
| I2C0MASK3 | R/W | 0x03C | **I2C Slave address mask register 3**. This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’). | 0x00 |

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

### 12.7.1 I²C Control Set register (I2C0CONSET - 0x4000 0000)

The CONSET registers control setting of bits in the CON register that controls operation of the I²C interface. Writing a one to a bit of this register causes the corresponding bit in the I²C control register to be set. Writing a zero has no effect.

**Table 154. I²C Control Set register (I2C0CONSET - address 0x4000 0000) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 1:0 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | AA | Assert acknowledge flag. | |
| 3 | SI | I²C interrupt flag. | 0 |
| 4 | STO | STOP flag. | 0 |
| 5 | STA | START flag. | 0 |
| 6 | I2EN | I²C interface enable. | 0 |
| 31:7 | - | Reserved. The value read from a reserved bit is not defined. | - |

**I2EN** I²C Interface Enable. When I2EN is 1, the I²C interface is enabled. I2EN can be cleared by writing 1 to the I2ENC bit in the CONCLR register. When I2EN is 0, the I²C interface is disabled.

When I2EN is “0”, the SDA and SCL input signals are ignored, the I²C block is in the “not addressed” slave state, and the STO bit is forced to “0”.

I2EN should not be used to temporarily release the $I^2C$-bus since, when I2EN is reset, the $I^2C$-bus status is lost. The AA flag should be used instead.

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

When STA is 1 and the $I^2C$ interface is not already in master mode, it enters master mode, checks the bus and generates a START condition if the bus is free. If the bus is not free, it waits for a STOP condition (which will free the bus) and generates a START condition after a delay of a half clock period of the internal clock generator. If the $I^2C$ interface is already in master mode and data has been transmitted or received, it transmits a Repeated START condition. STA may be set at any time, including when the $I^2C$ interface is in an addressed slave mode.

STA can be cleared by writing 1 to the STAC bit in the CONCLR register. When STA is 0, no START condition or Repeated START condition will be generated.

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

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

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

**SI** is the $I^2C$ Interrupt Flag. This bit is set when the $I^2C$ state changes. However, entering state F8 does not set SI since there is nothing for an interrupt service routine to do in that case.

While SI is set, the low period of the serial clock on the SCL line is stretched, and the serial transfer is suspended. When SCL is HIGH, it is unaffected by the state of the SI flag. SI must be reset by software, by writing a 1 to the SIC bit in the CONCLR register.

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

1. The address in the Slave Address Register has been received.
2. The General Call address has been received while the General Call bit (GC) in the ADR register is set.
3. A data byte has been received while the $I^2C$ is in the master receiver mode.
4. A data byte has been received while the $I^2C$ is in the addressed slave receiver mode

The AA bit can be cleared by writing 1 to the AAC bit in the CONCLR register. When AA is 0, a not acknowledge (HIGH level to SDA) will be returned during the acknowledge clock pulse on the SCL line on the following situations:

1. A data byte has been received while the $I^2C$ is in the master receiver mode.
2. A data byte has been received while the $I^2C$ is in the addressed slave receiver mode.

### 12.7.2 $I^2C$ Status register (I2C0STAT - 0x4000 0004)

Each $I^2C$ Status register reflects the condition of the corresponding $I^2C$ interface. The $I^2C$ Status register is Read-Only.

**Table 155. $I^2C$ Status register (I2C0STAT - 0x4000 0004) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 2:0 | - | These bits are unused and are always 0. | 0 |
| 7:3 | Status | These bits give the actual status information about the $I^2C$ interface. | 0x1F |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | - |

The three least significant bits are always 0. Taken as a byte, the status register contents represent a status code. There are 26 possible status codes. When the status code is 0xF8, there is no relevant information available and the SI bit is not set. All other 25 status codes correspond to defined $I^2C$ states. When any of these states entered, the SI bit will be set. For a complete list of status codes, refer to tables from Table 170 to Table 175.

### 12.7.3 $I^2C$ Data register (I2C0DAT - 0x4000 0008)

This register contains the data to be transmitted or the data just received. The CPU can read and write to this register only while it is not in the process of shifting a byte, when the SI bit is set. Data in DAT register remains stable as long as the SI bit is set. Data in DAT register is always shifted from right to left: the first bit to be transmitted is the MSB (bit 7), and after a byte has been received, the first bit of received data is located at the MSB of the DAT register.

**Table 156. $I^2C$ Data register (I2C0DAT - 0x4000 0008) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | Data | This register holds data values that have been received or are to be transmitted. | 0 |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | - |

### 12.7.4 $I^2C$ Slave Address register 0 (I2C0ADR0- 0x4000 000C)

This register is readable and writable and are only used when an $I^2C$ interface is set to slave mode. In master mode, this register has no effect. The LSB of the ADR register is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

If this register contains 0x00, the $I^2C$ will not acknowledge any address on the bus. All four registers (ADR0 to ADR3) will be cleared to this disabled state on reset. See also Table 163.

**Table 157. I²C Slave Address register 0 (I2C0ADR0- 0x4000 000C) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | GC | General Call enable bit. | 0 |
| 7:1 | Address | The I²C device address for slave mode. | 0x00 |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | - |

### 12.7.5 I²C SCL HIGH and LOW duty cycle registers (I2C0SCLH - 0x4000 0010 and I2C0SCLL- 0x4000 0014)

**Table 158. I²C SCL HIGH Duty Cycle register (I2C0SCLH - address 0x4000 0010) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 15:0 | SCLH | Count for SCL HIGH time period selection. | 0x0004 |
| 31:16 | - | Reserved. The value read from a reserved bit is not defined. | - |

**Table 159. I²C SCL Low duty cycle register (I2C0SCLL - 0x4000 0014) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 15:0 | SCLL | Count for SCL low time period selection. | 0x0004 |
| 31:16 | - | Reserved. The value read from a reserved bit is not defined. | - |

#### 12.7.5.1 Selecting the appropriate I²C data rate and duty cycle

Software must set values for the registers SCLH and SCLL to select the appropriate data rate and duty cycle. SCLH defines the number of I2C_PCLK cycles for the SCL HIGH time, SCLL defines the number of I2C_PCLK cycles for the SCL low time. The frequency is determined by the following formula (I2C_PCLK is the frequency of the peripheral I2C clock):

(4)

$$I^2C_{bitfrequency} = \frac{I2CPCLK}{SCLH + SCLL}$$

The values for SCLL and SCLH must ensure that the data rate is in the appropriate I²C data rate range. Each register value must be greater than or equal to 4. Table 160 gives some examples of I²C-bus rates based on I2C_PCLK frequency and SCLL and SCLH values.

**Table 160. SCLL + SCLH values for selected I²C clock values**

| I²C mode | I²C bit frequency | I2C_PCLK (MHz) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 6 | 8 | 10 | 12 | 16 | 20 | 30 | 40 | 50 |
| | | SCLH + SCLL | | | | | | | | |
| Standard mode | 100 kHz | 60 | 80 | 100 | 120 | 160 | 200 | 300 | 400 | 500 |
| Fast-mode | 400 kHz | 15 | 20 | 25 | 30 | 40 | 50 | 75 | 100 | 125 |
| Fast-mode Plus | 1 MHz | - | 8 | 10 | 12 | 16 | 20 | 30 | 40 | 50 |

SCLL and SCLH values should not necessarily be the same. Software can set different duty cycles on SCL by setting these two registers. For example, the $I^2C$-bus specification defines the SCL low time and high time at different values for a Fast-mode and Fast-mode Plus $I^2C$.

### 12.7.6 $I^2C$ Control Clear register (I2C0CONCLR - 0x4000 0018)

The CONCLR register control clearing of bits in the CON register that controls operation of the $I^2C$ interface. Writing a one to a bit of this register causes the corresponding bit in the $I^2C$ control register to be cleared. Writing a zero has no effect.

**Table 161. $I^2C$ Control Clear register (I2C0CONCLR - 0x4000 0018) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 1:0 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | AAC | Assert acknowledge Clear bit. | |
| 3 | SIC | $I^2C$ interrupt Clear bit. | 0 |
| 4 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 5 | STAC | START flag Clear bit. | 0 |
| 6 | I2ENC | $I^2C$ interface Disable bit. | 0 |
| 7 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | - |

**AAC** is the Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the CONSET register. Writing 0 has no effect.

**SIC** is the $I^2C$ Interrupt Clear bit. Writing a 1 to this bit clears the SI bit in the CONSET register. Writing 0 has no effect.

**STAC** is the START flag Clear bit. Writing a 1 to this bit clears the STA bit in the CONSET register. Writing 0 has no effect.

**I2ENC** is the $I^2C$ Interface Disable bit. Writing a 1 to this bit clears the I2EN bit in the CONSET register. Writing 0 has no effect.

### 12.7.7 $I^2C$ Monitor mode control register (I2C0MMCTRL - 0x4000 001C)

This register controls the Monitor mode which allows the $I^2C$ module to monitor traffic on the $I^2C$ bus without actually participating in traffic or interfering with the $I^2C$ bus.

**Table 162. I²C Monitor mode control register (I2C0MMCTRL - 0x4000 001C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | MM_ENA | | Monitor mode enable. | 0 |
| | | 0 | Monitor mode disabled. | |
| | | 1 | The I²C module will enter monitor mode. In this mode the SDA output will be forced high. This will prevent the I²C module from outputting data of any kind (including ACK) onto the I²C data bus.<br>Depending on the state of the ENA_SCL bit, the output may be also forced high, preventing the module from having control over the I²C clock line. | |
| 1 | ENA_SCL | | SCL output enable. | 0 |
| | | 0 | When this bit is cleared to '0', the SCL output will be forced high when the module is in monitor mode. As described above, this will prevent the module from having any control over the I²C clock line. | |
| | | 1 | When this bit is set, the I²C module may exercise the same control over the clock line that it would in normal operation. This means that, acting as a slave peripheral, the I²C module can "stretch" the clock line (hold it low) until it has had time to respond to an I²C interrupt.[1] | |
| 2 | MATCH_ALL | | Select interrupt register match. | 0 |
| | | 0 | When this bit is cleared, an interrupt will only be generated when a match occurs to one of the (up-to) four address registers described above. That is, the module will respond as a normal slave as far as address-recognition is concerned. | |
| | | 1 | When this bit is set to '1' and the I²C is in monitor mode, an interrupt will be generated on ANY address received. This will enable the part to monitor all traffic on the bus. | |
| 31:3 | - | - | Reserved. The value read from reserved bits is not defined. | |

[1] When the ENA_SCL bit is cleared and the I²C no longer has the ability to stall the bus, interrupt response time becomes important. To give the part more time to respond to an I²C interrupt under these conditions, a DATA _BUFFER register is used (Section 12.7.9) to hold received data for a full 9-bit word transmission time.

**Remark:** The ENA_SCL and MATCH_ALL bits have no effect if the MM_ENA is '0' (i.e. if the module is NOT in monitor mode).

### 12.7.7.1 Interrupt in Monitor mode

All interrupts will occur as normal when the module is in monitor mode. This means that the first interrupt will occur when an address-match is detected (any address received if the MATCH_ALL bit is set, otherwise an address matching one of the four address registers).

Subsequent to an address-match detection, interrupts will be generated after each data byte is received for a slave-write transfer, or after each byte that the module "thinks" it has transmitted for a slave-read transfer. In this second case, the data register will actually contain data transmitted by some other slave on the bus which was actually addressed by the master.

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

#### 12.7.7.2 Loss of arbitration in Monitor mode

In monitor mode, the $I^2C$ module will not be able to respond to a request for information by the bus master or issue an ACK). Some other slave on the bus will respond instead. This will most probably result in a lost-arbitration state as far as our module is concerned.

Software should be aware of the fact that the module is in monitor mode and should not respond to any loss of arbitration state that is detected. In addition, hardware may be designed into the module to block some/all loss of arbitration states from occurring if those state would either prevent a desired interrupt from occurring or cause an unwanted interrupt to occur. Whether any such hardware will be added is still to be determined.

### 12.7.8 $I^2C$ Slave Address registers (I2C0ADR[1, 2, 3] - 0x4000 00[20, 24, 28])

These registers are readable and writable and are only used when an $I^2C$ interface is set to slave mode. In master mode, this register has no effect. The LSB of the ADR register is the General Call bit. When this bit is set, the General Call address (0x00) is recognized.

If these registers contain 0x00, the $I^2C$ will not acknowledge any address on the bus. All four registers will be cleared to this disabled state on reset (also see Table 157).

**Table 163. $I^2C$ Slave Address registers (I2C0ADR[1, 2, 3]- 0x4000 00[20, 24, 28]) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | GC | General Call enable bit. | 0 |
| 7:1 | Address | The $I^2C$ device address for slave mode. | 0x00 |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | 0 |

### 12.7.9 $I^2C$ Data buffer register (I2C0DATA_BUFFER - 0x4000 002C)

In monitor mode, the $I^2C$ module may lose the ability to stretch the clock (stall the bus) if the ENA_SCL bit is not set. This means that the processor will have a limited amount of time to read the contents of the data received on the bus. If the processor reads the DAT shift register, as it ordinarily would, it could have only one bit-time to respond to the interrupt before the received data is overwritten by new data.

To give the processor more time to respond, a new 8-bit, read-only DATA_BUFFER register will be added. The contents of the 8 MSBs of the DAT shift register will be transferred to the DATA_BUFFER automatically after every nine bits (8 bits of data plus ACK or NACK) has been received on the bus. This means that the processor will have nine bit transmission times to respond to the interrupt and read the data before it is overwritten.

The processor will still have the ability to read the DAT register directly, as usual, and the behavior of DAT will not be altered in any way.

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

Table 164. I2C Data buffer register (I2C0DATA_BUFFER - 0x4000 002C) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | Data | This register holds contents of the 8 MSBs of the DAT shift register. | 0 |
| 31:8 | - | Reserved. The value read from a reserved bit is not defined. | 0 |

### 12.7.10 I2C Mask registers (I2C0MASK[0, 1, 2, 3] - 0x4000 00[30, 34, 38, 3C])

The four mask registers each contain seven active bits (7:1). Any bit in these registers which is set to '1' will cause an automatic compare on the corresponding bit of the received address when it is compared to the ADDRn register associated with that mask register. In other words, bits in an ADDRn register which are masked are not taken into account in determining an address match.

On reset, all mask register bits are cleared to '0'.

The mask register has no effect on comparison to the General Call address ("0000000").

Bits(31:8) and bit(0) of the mask registers are unused and should not be written to. These bits will always read back as zeros.

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

Table 165. I2C Mask registers (I2C0MASK[0, 1, 2, 3] - 0x4000 00[30, 34, 38, 3C]) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | - | Reserved. User software should not write ones to reserved bits. This bit reads always back as 0. | 0 |
| 7:1 | MASK | Mask bits. | 0x00 |
| 31:8 | - | Reserved. The value read from reserved bits is undefined. | 0 |

## 12.8 I2C operating modes

In a given application, the I2C block may operate as a master, a slave, or both. In the slave mode, the I2C hardware looks for any one of its four slave addresses and the General Call address. If one of these addresses is detected, an interrupt is requested. If the processor wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave operation is not interrupted. If bus arbitration is lost in the master mode, the I2C block switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

### 12.8.1 Master Transmitter mode

In this mode data is transmitted from master to slave. Before the master transmitter mode can be entered, the CONSET register must be initialized as shown in Table 166. I2EN must be set to 1 to enable the I2C function. If the AA bit is 0, the I2C interface will not acknowledge any address when another device is master of the bus, so it can not enter slave mode. The STA, STO and SI bits must be 0. The SI Bit is cleared by writing 1 to the SIC bit in the CONCLR register. THe STA bit should be cleared after writing the slave address.

**Table 166. I2C0CONSET and I2C1CONSET used to configure Master mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | 0 | - | - |

The first byte transmitted contains the slave address of the receiving device (7 bits) and the data direction bit. In this mode the data direction bit (R/W) should be 0 which means Write. The first byte transmitted contains the slave address and Write bit. Data is transmitted 8 bits at a time. After each byte is transmitted, an acknowledge bit is received. START and STOP conditions are output to indicate the beginning and the end of a serial transfer.

The $I^2C$ interface will enter master transmitter mode when software sets the STA bit. The $I^2C$ logic will send the START condition as soon as the bus is free. After the START condition is transmitted, the SI bit is set, and the status code in the STAT register is 0x08. This status code is used to vector to a state service routine which will load the slave address and Write bit to the DAT register, and then clear the SI bit. SI is cleared by writing a 1 to the SIC bit in the CONCLR register.

When the slave address and R/W bit have been transmitted and an acknowledgment bit has been received, the SI bit is set again, and the possible status codes now are 0x18, 0x20, or 0x38 for the master mode, or 0x68, 0x78, or 0xB0 if the slave mode was enabled (by setting AA to 1). The appropriate actions to be taken for each of these status codes are shown in Table 170 to Table 175.

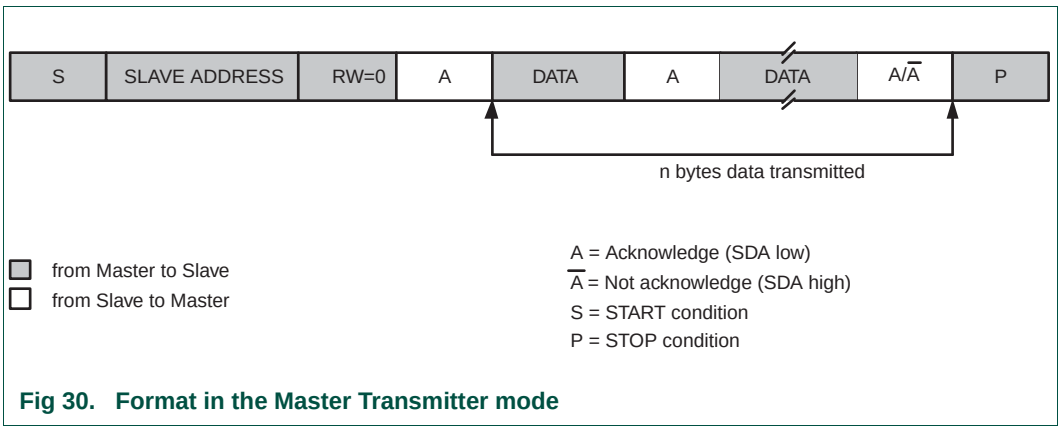



**Fig 30. Format in the Master Transmitter mode**

## 12.8.2 Master Receiver mode

In the master receiver mode, data is received from a slave transmitter. The transfer is initiated in the same way as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load the slave address and the data direction bit to the $I^2C$ Data register (DAT), and then clear the SI bit. In this case, the data direction bit (R/W) should be 1 to indicate a read.

When the slave address and data direction bit have been transmitted and an acknowledge bit has been received, the SI bit is set, and the Status Register will show the status code. For master mode, the possible status codes are 0x40, 0x48, or 0x38. For slave mode, the possible status codes are 0x68, 0x78, or 0xB0. For details, refer to Table 171.

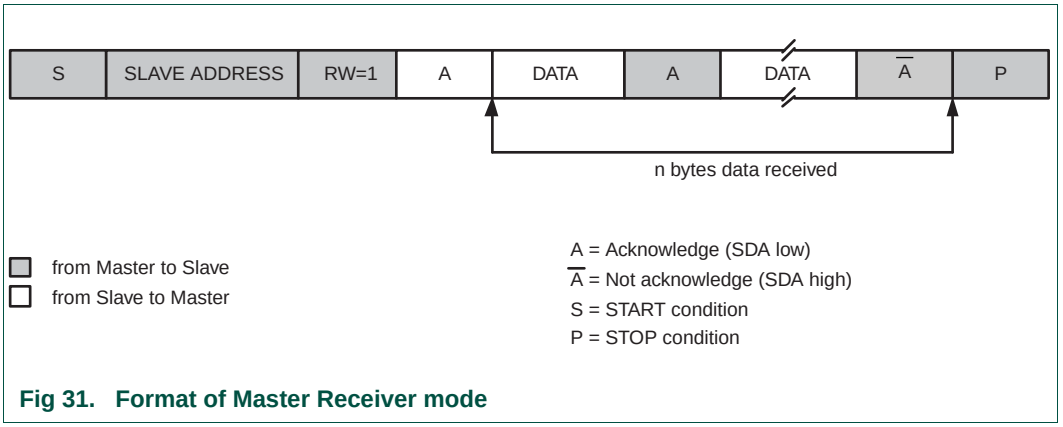



**Fig 31. Format of Master Receiver mode**

After a Repeated START condition, I²C may switch to the master transmitter mode.

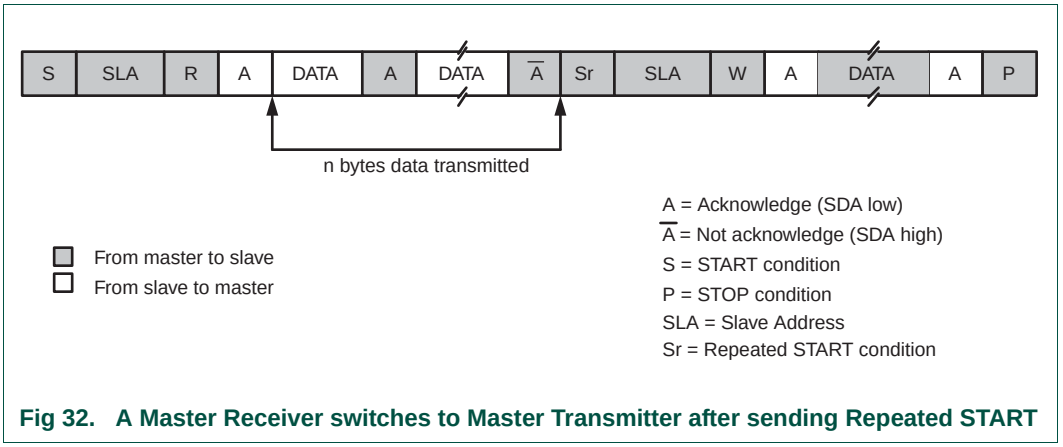



**Fig 32. A Master Receiver switches to Master Transmitter after sending Repeated START**

### 12.8.3 Slave Receiver mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, write any of the Slave Address registers (ADR0-3) and write the I²C Control Set register (CONSET) as shown in Table 167.

**Table 167. I2C0CONSET and I2C1CONSET used to configure Slave mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | 1 | - | - |

I2EN must be set to 1 to enable the I²C function. AA bit must be set to 1 to acknowledge its own slave address or the General Call address. The STA, STO and SI bits are set to 0.

After ADR and CONSET are initialized, the I²C interface waits until it is addressed by its own address or general address followed by the data direction bit. If the direction bit is 0 (W), it enters slave receiver mode. If the direction bit is 1 (R), it enters slave transmitter mode. After the address and direction bit have been received, the SI bit is set and a valid status code can be read from the Status register (STAT). Refer to Table 174 for the status codes and actions.

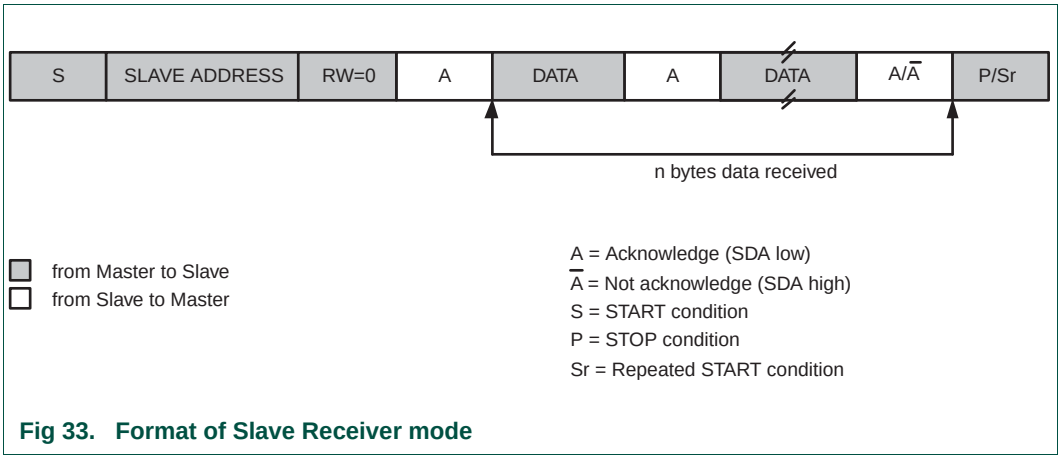



**Fig 33. Format of Slave Receiver mode**

### 12.8.4 Slave Transmitter mode

The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will be 1, indicating a read operation. Serial data is transmitted via SDA while the serial clock is input through SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. In a given application, $I^2C$ may operate as a master and as a slave. In the slave mode, the $I^2C$ hardware looks for its own slave address and the General Call address. If one of these addresses is detected, an interrupt is requested. When the microcontrollers wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave action is not interrupted. If bus arbitration is lost in the master mode, the $I^2C$ interface switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

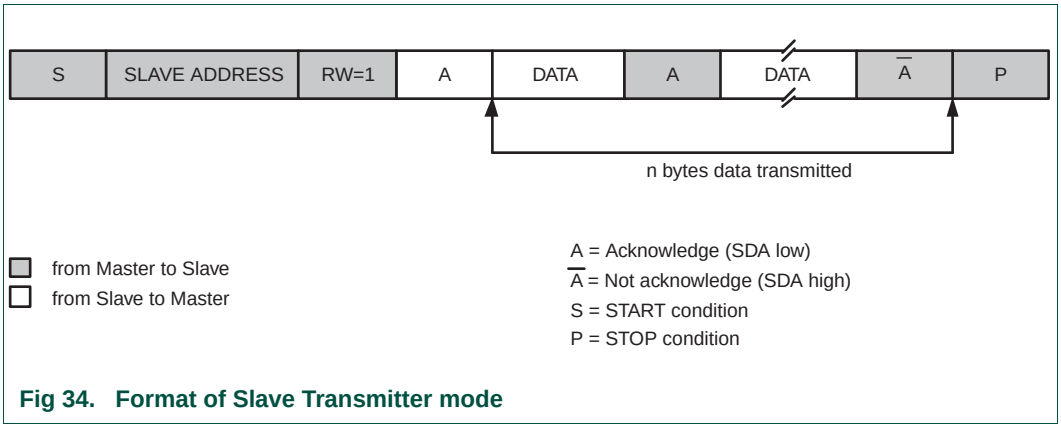



**Fig 34. Format of Slave Transmitter mode**

## 12.9 $I^2C$ implementation and operation

Figure 35 shows how the on-chip $I^2C$-bus interface is implemented, and the following text describes the individual blocks.

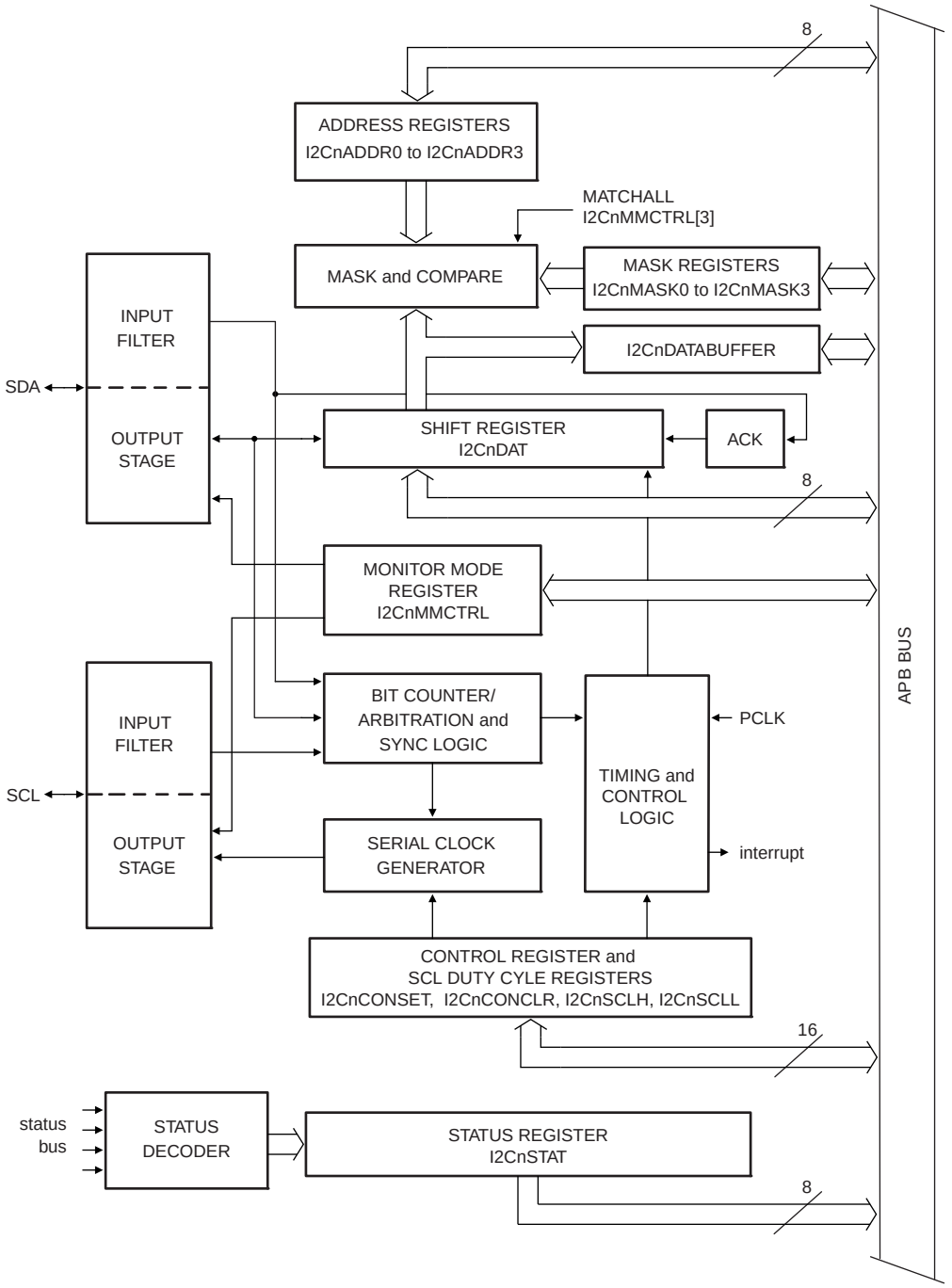



**Fig 35. I²C serial interface block diagram**

## 12.9.1 Input filters and output stages

Input signals are synchronized with the internal clock, and spikes shorter than three clocks are filtered out.

The output for I²C is a special pad designed to conform to the I²C specification.

### 12.9.2 Address Registers, ADDR0 to ADDR3

These registers may be loaded with the 7-bit slave address (7 most significant bits) to which the $I^2C$ block will respond when programmed as a slave transmitter or receiver. The LSB (GC) is used to enable General Call address (0x00) recognition. When multiple slave addresses are enabled, the actual address received may be read from the DAT register at the state where the own slave address has been received.

### 12.9.3 Address mask registers, MASK0 to MASK3

The four mask registers each contain seven active bits (7:1). Any bit in these registers which is set to '1' will cause an automatic compare on the corresponding bit of the received address when it is compared to the ADDRn register associated with that mask register. In other words, bits in an ADDRn register which are masked are not taken into account in determining an address match.

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

### 12.9.4 Comparator

The comparator compares the received 7-bit slave address with its own slave address (7 most significant bits in ADR). It also compares the first received 8-bit byte with the General Call address (0x00). If an equality is found, the appropriate status bits are set and an interrupt is requested.

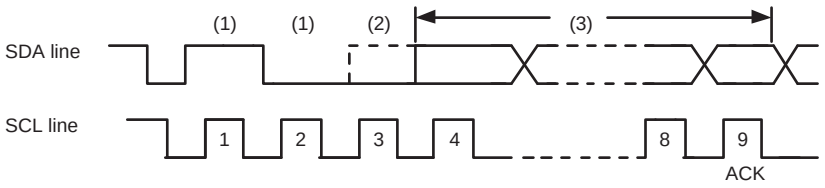
### 12.9.5 Shift register, DAT

This 8-bit register contains a byte of serial data to be transmitted or a byte which has just been received. Data in DAT is always shifted from right to left; the first bit to be transmitted is the MSB (bit 7) and, after a byte has been received, the first bit of received data is located at the MSB of DAT. While data is being shifted out, data on the bus is simultaneously being shifted in; DAT always contains the last byte present on the bus. Thus, in the event of lost arbitration, the transition from master transmitter to slave receiver is made with the correct data in DAT.

### 12.9.6 Arbitration and synchronization logic

In the master transmitter mode, the arbitration logic checks that every transmitted logic 1 actually appears as a logic 1 on the $I^2C$-bus. If another device on the bus overrules a logic 1 and pulls the SDA line low, arbitration is lost, and the $I^2C$ block immediately changes from master transmitter to slave receiver. The $I^2C$ block will continue to output clock pulses (on SCL) until transmission of the current serial byte is complete.

Arbitration may also be lost in the master receiver mode. Loss of arbitration in this mode can only occur while the $I^2C$ block is returning a "not acknowledge: (logic 1) to the bus. Arbitration is lost when another device on the bus pulls this signal low. Since this can occur only at the end of a serial byte, the $I^2C$ block generates no further clock pulses. Figure 36 shows the arbitration procedure.



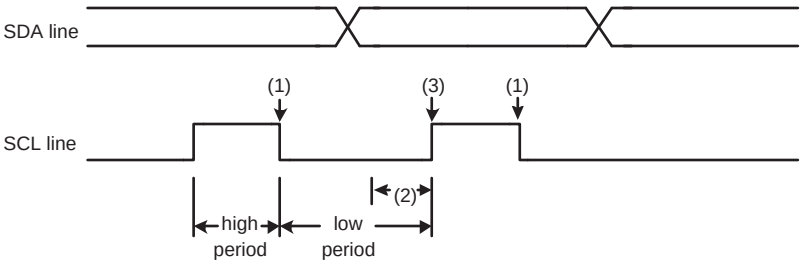

(1) Another device transmits serial data.

(2) Another device overrules a logic (dotted line) transmitted this I2C master by pulling the SDA line low. Arbitration is lost, and this I2C enters Slave Receiver mode.

(3) This I2C is in Slave Receiver mode but still generates clock pulses until the current byte has been transmitted. This I2C will not generate clock pulses for the next byte. Data on SDA originates from the new master once it has won arbitration.

**Fig 36. Arbitration procedure**

The synchronization logic will synchronize the serial clock generator with the clock pulses on the SCL line from another device. If two or more master devices generate clock pulses, the “mark” duration is determined by the device that generates the shortest “marks,” and the “space” duration is determined by the device that generates the longest “spaces”. Figure 37 shows the synchronization procedure.




(1) Another device pulls the SCL line low before this I2C has timed a complete high time. The other device effectively determines the (shorter) HIGH period.

(2) Another device continues to pull the SCL line low after this I2C has timed a complete low time and released SCL. The I2C clock generator is forced to wait until SCL goes HIGH. The other device effectively determines the (longer) LOW period.

(3) The SCL line is released , and the clock generator begins timing the HIGH time.

**Fig 37. Serial clock synchronization**

A slave may stretch the space duration to slow down the bus master. The space duration may also be stretched for handshaking purposes. This can be done after each bit or after a complete byte transfer. the I2C block will stretch the SCL space duration after a byte has been transmitted or received and the acknowledge bit has been transferred. The serial interrupt flag (SI) is set, and the stretching continues until the serial interrupt flag is cleared.

### 12.9.7 Serial clock generator

This programmable clock pulse generator provides the SCL clock pulses when the I2C block is in the master transmitter or master receiver mode. It is switched off when the I2C block is in slave mode. The I2C output clock frequency and duty cycle is programmable

via the I2C Clock Control Registers. See the description of the I2CSCLL and I2CSCLH registers for details. The output clock pulses have a duty cycle as programmed unless the bus is synchronizing with other SCL clock sources as described above.

### 12.9.8 Timing and control

The timing and control logic generates the timing and control signals for serial byte handling. This logic block provides the shift pulses for DAT, enables the comparator, generates and detects START and STOP conditions, receives and transmits acknowledge bits, controls the master and slave modes, contains interrupt request logic, and monitors the I2C-bus status.

### 12.9.9 Control register, CONSET and CONCLR

The I2C control register contains bits used to control the following I2C block functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.

The contents of the I2C control register may be read as CONSET. Writing to CONSET will set bits in the I2C control register that correspond to ones in the value written. Conversely, writing to CONCLR will clear bits in the I2C control register that correspond to ones in the value written.

### 12.9.10 Status decoder and status register

The status decoder takes all of the internal status bits and compresses them into a 5-bit code. This code is unique for each I2C-bus status. The 5-bit code may be used to generate vector addresses for fast processing of the various service routines. Each service routine processes a particular bus status. There are 26 possible bus states if all four modes of the I2C block are used. The 5-bit status code is latched into the five most significant bits of the status register when the serial interrupt flag is set (by hardware) and remains stable until the interrupt flag is cleared by software. The three least significant bits of the status register are always zero. If the status code is used as a vector to service routines, then the routines are displaced by eight address locations. Eight bytes of code is sufficient for most of the service routines (see the software example in this section).

## 12.10 Details of I2C operating modes

The four operating modes are:

- Master Transmitter
- Master Receiver
- Slave Receiver
- Slave Transmitter

Data transfers in each mode of operation are shown in Figure 38, Figure 39, Figure 40, Figure 41, and Figure 42. Table 168 lists abbreviations used in these figures when describing the I2C operating modes.

**Table 168. Abbreviations used to describe an $I^2C$ operation**

| Abbreviation | Explanation |
|---|---|
| S | START Condition |
| SLA | 7-bit slave address |
| R | Read bit (HIGH level at SDA) |
| W | Write bit (LOW level at SDA) |
| A | Acknowledge bit (LOW level at SDA) |
| $\overline{A}$ | Not acknowledge bit (HIGH level at SDA) |
| Data | 8-bit data byte |
| P | STOP condition |

In Figure 38 to Figure 42, circles are used to indicate when the serial interrupt flag is set. The numbers in the circles show the status code held in the STAT register. At these points, a service routine must be executed to continue or complete the serial transfer. These service routines are not critical since the serial transfer is suspended until the serial interrupt flag is cleared by software.

When a serial interrupt routine is entered, the status code in STAT is used to branch to the appropriate service routine. For each status code, the required software action and details of the following serial transfer are given in tables from Table 170 to Table 176.

### 12.10.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 38). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

**Table 169. I2C0CONSET used to initialize Master Transmitter mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | x | - | - |

The $I^2C$ rate must also be configured in the SCLL and SCLH registers. I2EN must be set to logic 1 to enable the $I^2C$ block. If the AA bit is reset, the $I^2C$ block will not acknowledge its own slave address or the General Call address in the event of another device becoming master of the bus. In other words, if AA is reset, the $I^2C$ interface cannot enter slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The $I^2C$ logic will now test the $I^2C$-bus and generate a START condition as soon as the bus becomes free. When a START condition is transmitted, the serial interrupt flag (SI) is set, and the status code in the status register (STAT) will be 0x08. This status code is used by the interrupt service routine to enter the appropriate state service routine that loads DAT with the slave address and the data direction bit (SLA+W). The SI bit in CON must then be reset before the serial transfer can continue.

When the slave address and the direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in STAT are possible. There are 0x18, 0x20, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = logic 1). The appropriate action to be taken for each of these status codes is detailed in Table 170. After a Repeated START condition (state 0x10). The $I^2C$ block may switch to the master receiver mode by loading DAT with SLA+R).

**Table 170. Master Transmitter mode**

| Status Code (I2CSTAT) | Status of the $I^2C$-bus and hardware | Application software response | | | | | Next action taken by $I^2C$ hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x08 | A START condition has been transmitted. | Load SLA+W; clear STA | X | 0 | 0 | X | SLA+W will be transmitted; ACK bit will be received. |
| 0x10 | A Repeated START condition has been transmitted. | Load SLA+W or | X | 0 | 0 | X | As above. |
| | | Load SLA+R; Clear STA | X | 0 | 0 | X | SLA+R will be transmitted; the $I^2C$ block will be switched to MST/REC mode. |
| 0x18 | SLA+W has been transmitted; ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x20 | SLA+W has been transmitted; NOT ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x28 | Data byte in DAT has been transmitted; ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x30 | Data byte in DAT has been transmitted; NOT ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x38 | Arbitration lost in SLA+R/W or Data bytes. | No DAT action or | 0 | 0 | 0 | X | $I^2C$-bus will be released; not addressed slave will be entered. |
| | | No DAT action | 1 | 0 | 0 | X | A START condition will be transmitted when the bus becomes free. |

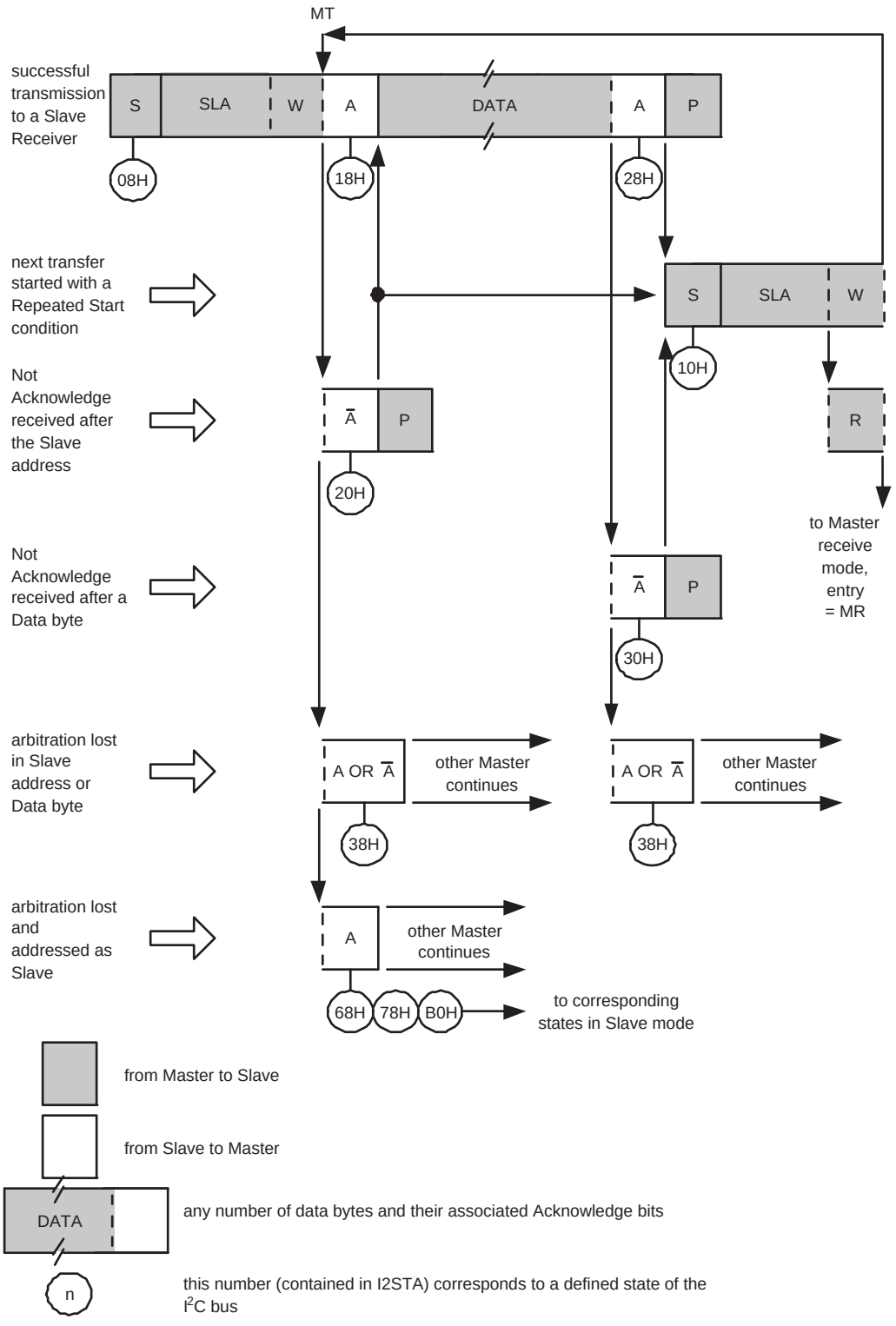



**Fig 38. Format and states in the Master Transmitter mode**

### 12.10.2 Master Receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see Figure 39). The transfer is initialized as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load DAT with the 7-bit slave address and the data direction bit (SLA+R). The SI bit in CON must then be cleared before the serial transfer can continue.

When the slave address and the data direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in STAT are possible. These are 0x40, 0x48, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = 1). The appropriate action to be taken for each of these status codes is detailed in Table 171. After a Repeated START condition (state 0x10), the $I^2C$ block may switch to the master transmitter mode by loading DAT with SLA+W.

**Table 171. Master Receiver mode**

| Status Code (STAT) | Status of the $I^2C$-bus and hardware | Application software response | | | | | Next action taken by $I^2C$ hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x08 | A START condition has been transmitted. | Load SLA+R | X | 0 | 0 | X | SLA+R will be transmitted; ACK bit will be received. |
| 0x10 | A Repeated START condition has been transmitted. | Load SLA+R or | X | 0 | 0 | X | As above. |
| | | Load SLA+W | X | 0 | 0 | X | SLA+W will be transmitted; the $I^2C$ block will be switched to MST/TRX mode. |
| 0x38 | Arbitration lost in NOT ACK bit. | No DAT action or | 0 | 0 | 0 | X | $I^2C$-bus will be released; the $I^2C$ block will enter slave mode. |
| | | No DAT action | 1 | 0 | 0 | X | A START condition will be transmitted when the bus becomes free. |
| 0x40 | SLA+R has been transmitted; ACK has been received. | No DAT action or | 0 | 0 | 0 | 0 | Data byte will be received; NOT ACK bit will be returned. |
| | | No DAT action | 0 | 0 | 0 | 1 | Data byte will be received; ACK bit will be returned. |
| 0x48 | SLA+R has been transmitted; NOT ACK has been received. | No DAT action or | 1 | 0 | 0 | X | Repeated START condition will be transmitted. |
| | | No DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x50 | Data byte has been received; ACK has been returned. | Read data byte or | 0 | 0 | 0 | 0 | Data byte will be received; NOT ACK bit will be returned. |
| | | Read data byte | 0 | 0 | 0 | 1 | Data byte will be received; ACK bit will be returned. |
| 0x58 | Data byte has been received; NOT ACK has been returned. | Read data byte or | 1 | 0 | 0 | X | Repeated START condition will be transmitted. |
| | | Read data byte or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | Read data byte | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |

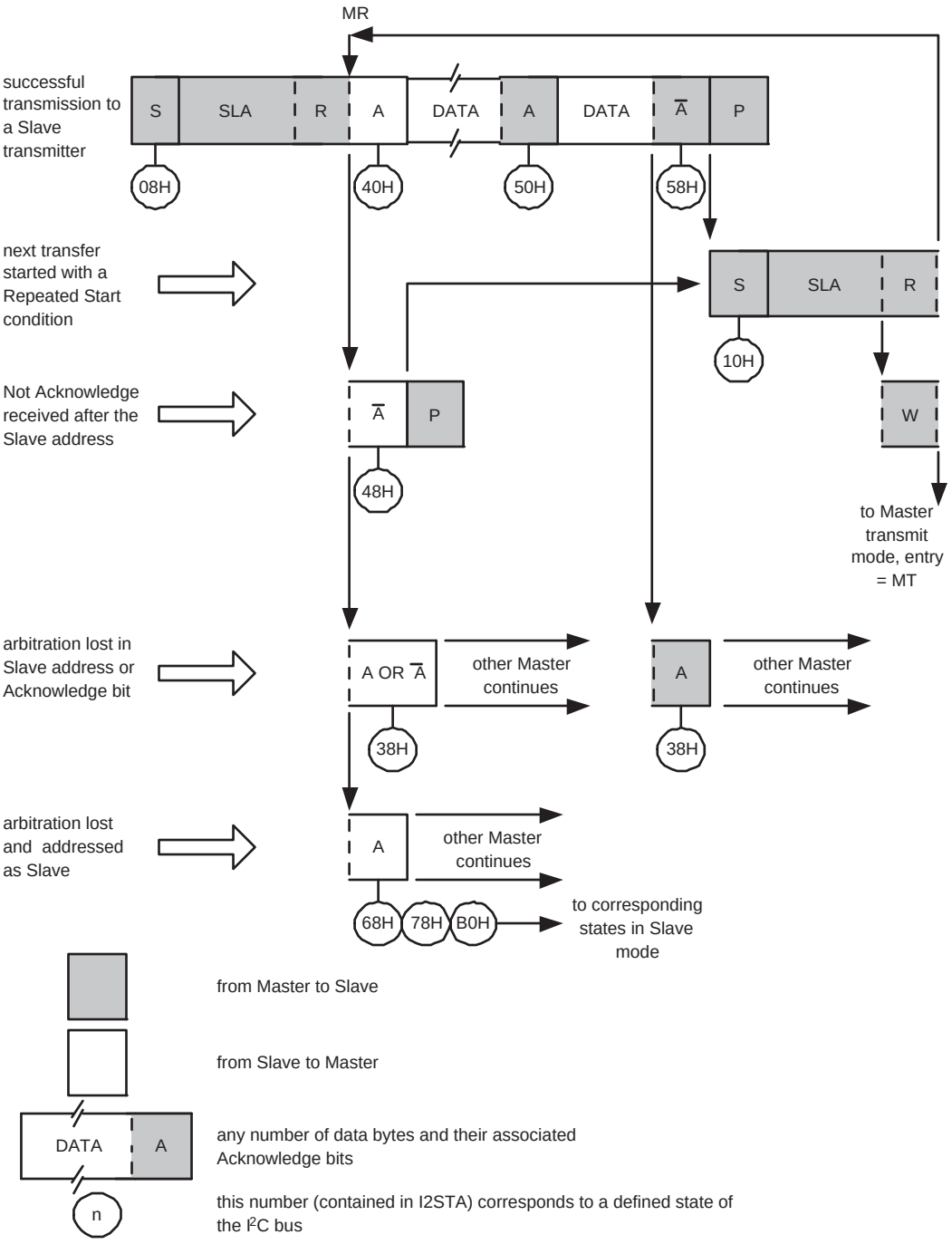



**Fig 39. Format and states in the Master Receiver mode**

### 12.10.3 Slave Receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 40). To initiate the slave receiver mode, ADR and CON must be loaded as follows:

**Table 172. I2C0ADR and I2C1ADR usage in Slave Receiver mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | own slave 7-bit address | | | | | | | GC |

The upper 7 bits are the address to which the $I^2C$ block will respond when addressed by a master. If the LSB (GC) is set, the $I^2C$ block will respond to the General Call address (0x00); otherwise it ignores the General Call address.

**Table 173. I2C0CONSET and I2C1CONSET used to initialize Slave Receiver mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | 1 | - | - |

The $I^2C$-bus rate settings do not affect the $I^2C$ block in the slave mode. I2EN must be set to logic 1 to enable the $I^2C$ block. The AA bit must be set to enable the $I^2C$ block to acknowledge its own slave address or the General Call address. STA, STO, and SI must be reset.

When ADR and CON have been initialized, the $I^2C$ block waits until it is addressed by its own slave address followed by the data direction bit which must be “0” (W) for the $I^2C$ block to operate in the slave receiver mode. After its own slave address and the W bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from STAT. This status code is used to vector to a state service routine. The appropriate action to be taken for each of these status codes is detailed in Table 174. The slave receiver mode may also be entered if arbitration is lost while the $I^2C$ block is in the master mode (see status 0x68 and 0x78).

If the AA bit is reset during a transfer, the $I^2C$ block will return a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the $I^2C$ block does not respond to its own slave address or a General Call address. However, the $I^2C$-bus is still monitored and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the $I^2C$ block from the $I^2C$-bus.

**Table 174. Slave Receiver mode**

| Status Code (STAT) | Status of the $I^2C$-bus and hardware | Application software response | | | | | Next action taken by $I^2C$ hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x60 | Own SLA+W has been received; ACK has been returned. | No DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x68 | Arbitration lost in SLA+R/W as master; Own SLA+W has been received, ACK returned. | No DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x70 | General call address (0x00) has been received; ACK has been returned. | No DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x78 | Arbitration lost in SLA+R/W as master; General call address has been received, ACK has been returned. | No DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x80 | Previously addressed with own SLV address; DATA has been received; ACK has been returned. | Read data byte or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | Read data byte | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x88 | Previously addressed with own SLA; DATA byte has been received; NOT ACK has been returned. | Read data byte or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | Read data byte or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. |
| | | Read data byte or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | Read data byte | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0x90 | Previously addressed with General Call; DATA byte has been received; ACK has been returned. | Read data byte or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | Read data byte | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |

**Table 174. Slave Receiver mode** *…continued*

| Status Code (STAT) | Status of the I²C-bus and hardware | Application software response | | | | | Next action taken by I²C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x98 | Previously addressed with General Call; DATA byte has been received; NOT ACK has been returned. | Read data byte or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | Read data byte or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. |
| | | Read data byte or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | Read data byte | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0xA0 | A STOP condition or Repeated START condition has been received while still addressed as SLV/REC or SLV/TRX. | No STDAT action or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | No STDAT action or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. |
| | | No STDAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No STDAT action | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |

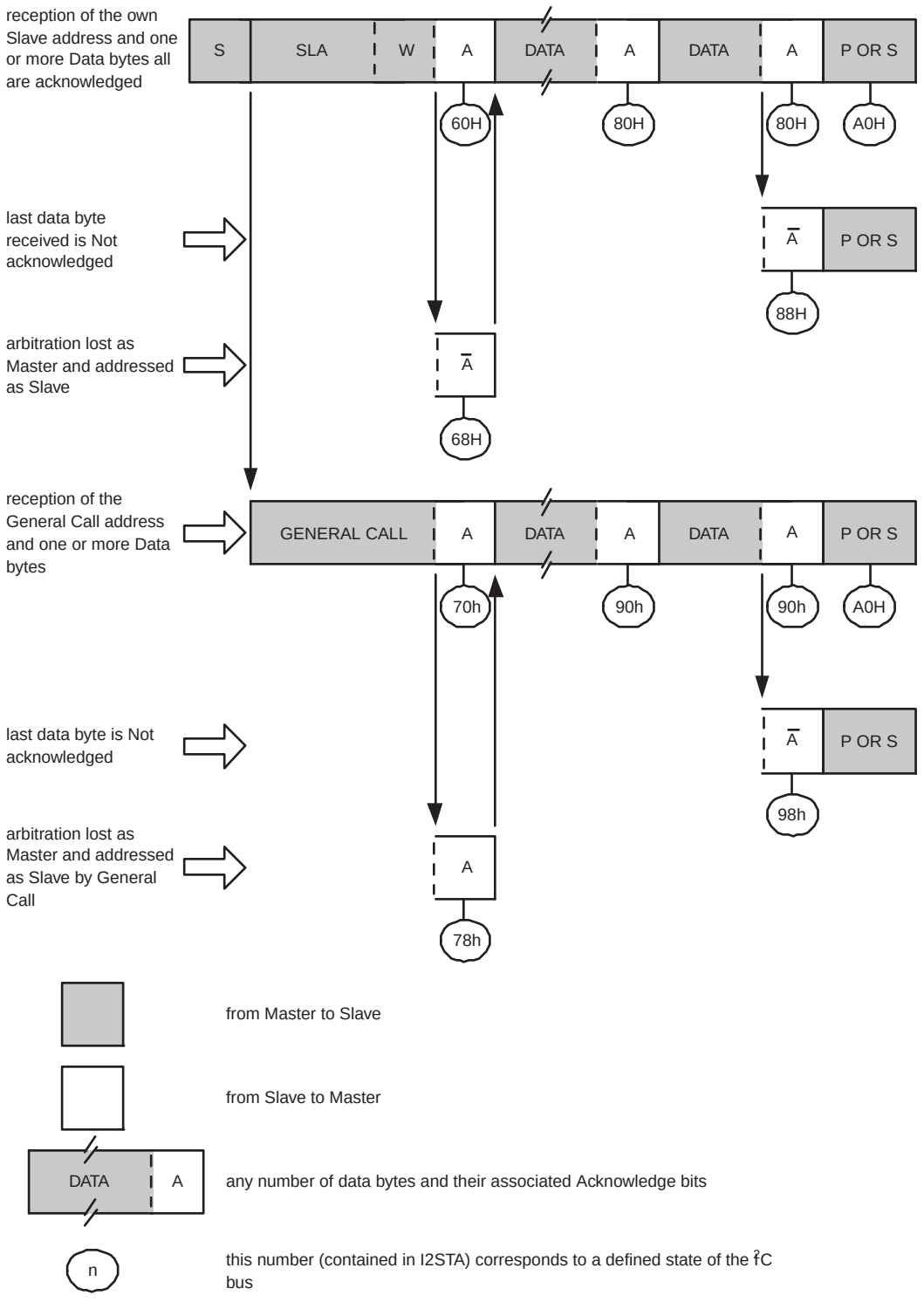



**Fig 40. Format and states in the Slave Receiver mode**

### 12.10.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 41). Data transfer is initialized as in the slave receiver mode. When ADR and CON have been initialized, the $I^2C$ block waits until it is addressed by its own slave address followed by the data direction bit which must be “1” (R) for the $I^2C$ block to operate in the slave transmitter mode. After its own slave address and the R bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from STAT. This status code is used to vector to a state service routine, and the appropriate action to be taken for each of these status codes is detailed in Table 175. The slave transmitter mode may also be entered if arbitration is lost while the $I^2C$ block is in the master mode (see state 0xB0).

If the AA bit is reset during a transfer, the $I^2C$ block will transmit the last byte of the transfer and enter state 0xC0 or 0xC8. The $I^2C$ block is switched to the not addressed slave mode and will ignore the master receiver if it continues the transfer. Thus the master receiver receives all 1s as serial data. While AA is reset, the $I^2C$ block does not respond to its own slave address or a General Call address. However, the $I^2C$-bus is still monitored, and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the $I^2C$ block from the $I^2C$-bus.

**Table 175. Slave Transmitter mode**

| Status Code (STAT) | Status of the I²C-bus and hardware | Application software response | | | | | Next action taken by I²C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0xA8 | Own SLA+R has been received; ACK has been returned. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK will be received. |
| 0xB0 | Arbitration lost in SLA+R/W as master; Own SLA+R has been received, ACK has been returned. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK bit will be received. |
| 0xB8 | Data byte in DAT has been transmitted; ACK has been received. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK bit will be received. |
| 0xC0 | Data byte in DAT has been transmitted; NOT ACK has been received. | No DAT action or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | No DAT action or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. |
| | | No DAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No DAT action | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0xC8 | Last data byte in DAT has been transmitted (AA = 0); ACK has been received. | No DAT action or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | No DAT action or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR[0] = logic 1. |
| | | No DAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No DAT action | 1 | 0 | 0 | 01 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if ADR.0 = logic 1. A START condition will be transmitted when the bus becomes free. |

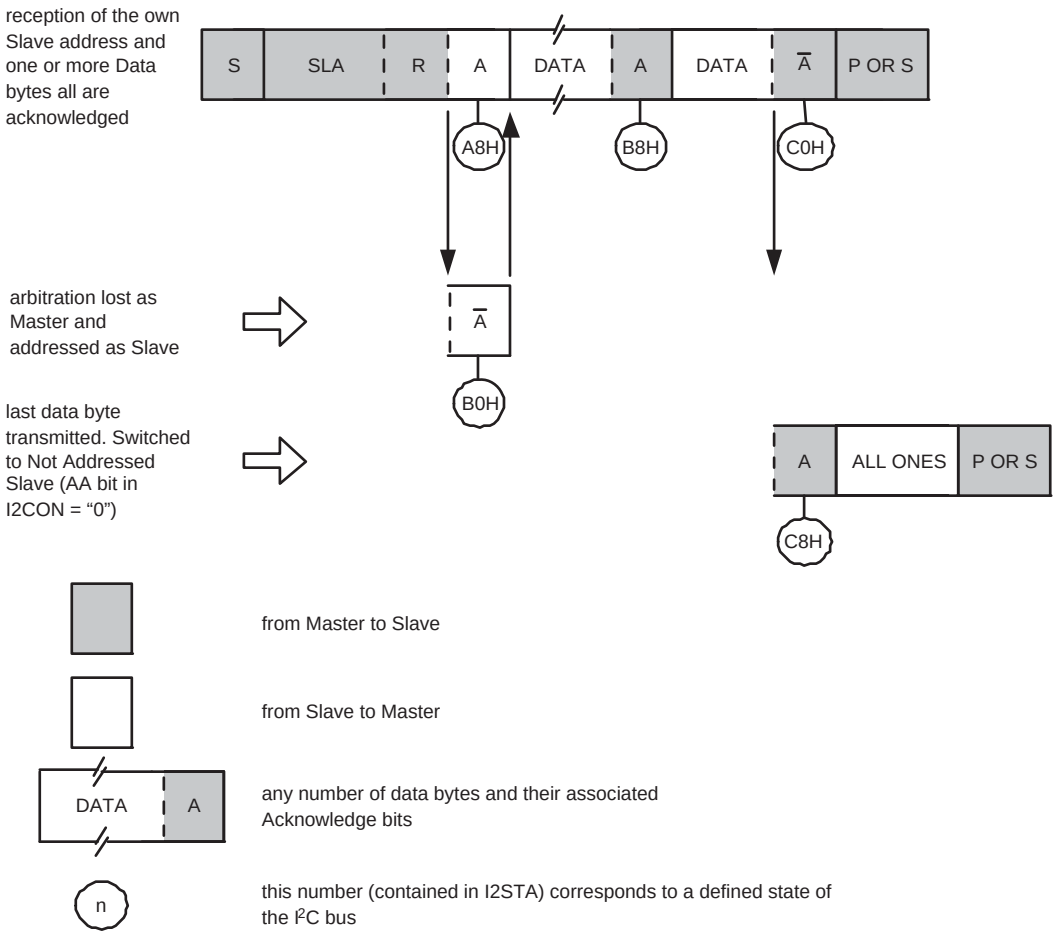



**Fig 41. Format and states in the Slave Transmitter mode**

### 12.10.5 Miscellaneous states

There are two STAT codes that do not correspond to a defined $I^2C$ hardware state (see Table 176). These are discussed below.

#### 12.10.5.1 STAT = 0xF8

This status code indicates that no relevant information is available because the serial interrupt flag, SI, is not yet set. This occurs between other states and when the $I^2C$ block is not involved in a serial transfer.

#### 12.10.5.2 STAT = 0x00

This status code indicates that a bus error has occurred during an $I^2C$ serial transfer. A bus error is caused when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. A bus error may also be caused when external interference disturbs the internal $I^2C$ block signals. When a bus error occurs, SI is set. To recover from a bus error, the STO flag must be set and SI must be cleared. This

causes the I²C block to enter the “not addressed” slave mode (a defined state) and to clear the STO flag (no other bits in CON are affected). The SDA and SCL lines are released (a STOP condition is not transmitted).

**Table 176. Miscellaneous States**

| Status Code (STAT) | Status of the I²C-bus and hardware | Application software response | | | | | Next action taken by I²C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From DAT | To CON | | | | |
| | | | STA | STO | SI | AA | |
| 0xF8 | No relevant state information available; SI = 0. | No DAT action | No CON action | | | | Wait or proceed current transfer. |
| 0x00 | Bus error during MST or selected slave modes, due to an illegal START or STOP condition. State 0x00 can also occur when interference causes the I²C block to enter an undefined state. | No DAT action | 0 | 1 | 0 | X | Only the internal hardware is affected in the MST or addressed SLV modes. In all cases, the bus is released and the I²C block is switched to the not addressed SLV mode. STO is reset. |

### 12.10.6 Some special cases

The I²C hardware has facilities to handle the following special cases that may occur during a serial transfer:

- Simultaneous Repeated START conditions from two masters
- Data transfer after loss of arbitration
- Forced access to the I²C-bus
- I²C-bus obstructed by a LOW level on SCL or SDA
- Bus error

#### 12.10.6.1 Simultaneous Repeated START conditions from two masters

A Repeated START condition may be generated in the master transmitter or master receiver modes. A special case occurs if another master simultaneously generates a Repeated START condition (see Figure 42). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

If the I²C hardware detects a Repeated START condition on the I²C-bus before generating a Repeated START condition itself, it will release the bus, and no interrupt request is generated. If another master frees the bus by generating a STOP condition, the I²C block will transmit a normal START condition (state 0x08), and a retry of the total serial data transfer can commence.

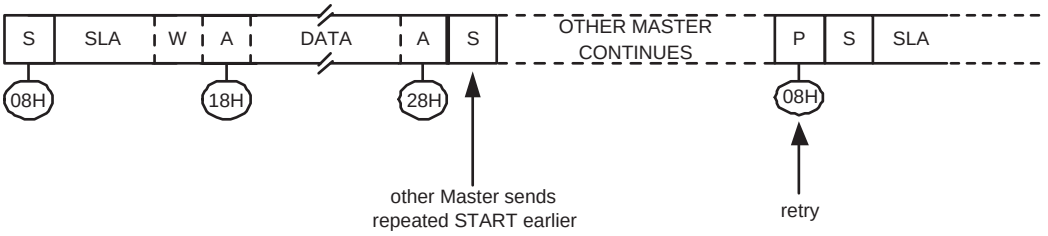



**Fig 42. Simultaneous Repeated START conditions from two masters**

### 12.10.6.2 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see Figure 36). Loss of arbitration is indicated by the following states in STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 38 and Figure 39).

If the STA flag in CON is set by the routines which service these states, then, if the bus is free again, a START condition (state 0x08) is transmitted without intervention by the CPU, and a retry of the total serial transfer can commence.

### 12.10.6.3 Forced access to the I²C-bus

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

If an uncontrolled source generates a superfluous START or masks a STOP condition, then the I²C-bus stays busy indefinitely. If the STA flag is set and bus access is not obtained within a reasonable amount of time, then a forced access to the I²C-bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I²C hardware behaves as if a STOP condition was received and is able to transmit a START condition. The STO flag is cleared by hardware (see Figure 43).

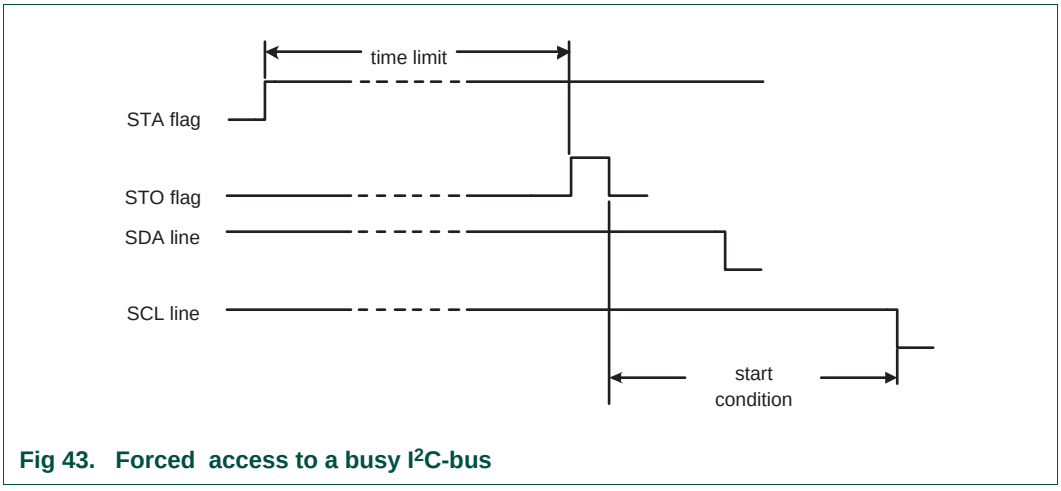
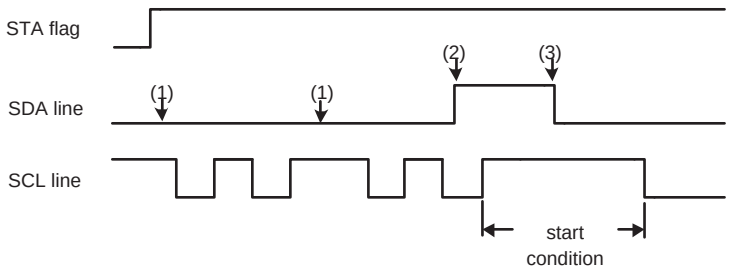



**Fig 43. Forced access to a busy I²C-bus**

#### 12.10.6.4 I2C-bus obstructed by a LOW level on SCL or SDA

An I2C-bus hang-up can occur if either the SDA or SCL line is held LOW by any device on the bus. If the SCL line is obstructed (pulled LOW) by a device on the bus, no further serial transfer is possible, and the problem must be resolved by the device that is pulling the SCL bus line LOW.

Typically, the SDA line may be obstructed by another device on the bus that has become out of synchronization with the current bus master by either missing a clock, or by sensing a noise pulse as a clock. In this case, the problem can be solved by transmitting additional clock pulses on the SCL line (see Figure 44). The I2C interface does not include a dedicated time-out timer to detect an obstructed bus, but this can be implemented using another timer in the system. When detected, software can force clocks (up to 9 may be required) on SCL until SDA is released by the offending device. At that point, the slave may still be out of synchronization, so a START should be generated to insure that all I2C peripherals are synchronized.




(1) Unsuccessful attempt to send a START condition.
(2) SDA line is released.
(3) Successful attempt to send a START condition. State 08H is entered.

**Fig 44. Recovering from a bus obstruction caused by a LOW level on SDA**

#### 12.10.6.5 Bus error

A bus error occurs when a START or STOP condition is detected at an illegal position in the format frame. Examples of illegal positions are during the serial transfer of an address byte, a data bit, or an acknowledge bit.

The I2C hardware only reacts to a bus error when it is involved in a serial transfer either as a master or an addressed slave. When a bus error is detected, the I2C block immediately switches to the not addressed slave mode, releases the SDA and SCL lines, sets the interrupt flag, and loads the status register with 0x00. This status code may be used to vector to a state service routine which either attempts the aborted serial transfer again or simply recovers from the error condition as shown in Table 176.

### 12.10.7 I2C state service routines

This section provides examples of operations that must be performed by various I2C state service routines. This includes:

- Initialization of the I2C block after a Reset.
- I2C Interrupt Service
- The 26 state service routines providing support for all four I2C operating modes.

### 12.10.8 Initialization

In the initialization example, the $I^2C$ block is enabled for both master and slave modes. For each mode, a buffer is used for transmission and reception. The initialization routine performs the following functions:

- ADR is loaded with the part's own slave address and the General Call bit (GC)
- The $I^2C$ interrupt enable and interrupt priority bits are set
- The slave mode is enabled by simultaneously setting the I2EN and AA bits in CON and the serial clock frequency (for master modes) is defined by is defined by loading the SCLH and SCLL registers. The master routines must be started in the main program.

The $I^2C$ hardware now begins checking the $I^2C$-bus for its own slave address and General Call. If the General Call or the own slave address is detected, an interrupt is requested and STAT is loaded with the appropriate state information.

### 12.10.9 $I^2C$ interrupt service

When the $I^2C$ interrupt is entered, STAT contains a status code which identifies one of the 26 state services to be executed.

### 12.10.10 The state service routines

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

### 12.10.11 Adapting state services to an application

The state service examples show the typical actions that must be performed in response to the 26 $I^2C$ state codes. If one or more of the four $I^2C$ operating modes are not used, the associated state services can be omitted, as long as care is taken that the those states can never occur.

In an application, it may be desirable to implement some kind of time-out during $I^2C$ operations, in order to trap an inoperative bus or a lost service routine.

## 12.11 Software example

### 12.11.1 Initialization routine

Example to initialize $I^2C$ Interface as a Slave and/or Master.

1. Load ADR with own Slave Address, enable General Call recognition if needed.
2. Enable $I^2C$ interrupt.
3. Write 0x44 to CONSET to set the I2EN and AA bits, enabling Slave functions. For Master only functions, write 0x40 to CONSET.

### 12.11.2 Start Master Transmit function

Begin a Master Transmit operation by setting up the buffer, pointer, and data count, then initiating a START.

1. Initialize Master data counter.

2. Set up the Slave Address to which data will be transmitted, and add the Write bit.
3. Write 0x20 to CONSET to set the STA bit.
4. Set up data to be transmitted in Master Transmit buffer.
5. Initialize the Master data counter to match the length of the message being sent.
6. Exit

### 12.11.3 Start Master Receive function

Begin a Master Receive operation by setting up the buffer, pointer, and data count, then initiating a START.

1. Initialize Master data counter.
2. Set up the Slave Address to which data will be transmitted, and add the Read bit.
3. Write 0x20 to CONSET to set the STA bit.
4. Set up the Master Receive buffer.
5. Initialize the Master data counter to match the length of the message to be received.
6. Exit

### 12.11.4 $I^2C$ interrupt routine

Determine the $I^2C$ state and which state routine will be used to handle it.

1. Read the $I^2C$ status from STA.
2. Use the status value to branch to one of 26 possible state routines.

### 12.11.5 Non mode specific states

#### 12.11.5.1 State: 0x00

Bus Error. Enter not addressed Slave mode and release bus.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.5.2 Master States

State 08 and State 10 are for both Master Transmit and Master Receive modes. The R/W bit decides whether the next state is within Master Transmit mode or Master Receive mode.

#### 12.11.5.3 State: 0x08

A START condition has been transmitted. The Slave Address + R/W bit will be transmitted, an ACK bit will be received.

1. Write Slave Address with R/W bit to DAT.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Master Transmit mode data buffer.

5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

#### 12.11.5.4 State: 0x10

A Repeated START condition has been transmitted. The Slave Address + R/W bit will be transmitted, an ACK bit will be received.

1. Write Slave Address with R/W bit to DAT.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Master Transmit mode data buffer.
5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

### 12.11.6 Master Transmitter states

#### 12.11.6.1 State: 0x18

Previous state was State 8 or State 10, Slave Address + Write has been transmitted, ACK has been received. The first data byte will be transmitted, an ACK bit will be received.

1. Load DAT with first data byte from Master Transmit buffer.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Increment Master Transmit buffer pointer.
5. Exit

#### 12.11.6.2 State: 0x20

Slave Address + Write has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.6.3 State: 0x28

Data has been transmitted, ACK has been received. If the transmitted data was the last data byte then transmit a STOP condition, otherwise transmit the next data byte.

1. Decrement the Master data counter, skip to step 5 if not the last data byte.
2. Write 0x14 to CONSET to set the STO and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Exit
5. Load DAT with next data byte from Master Transmit buffer.

6. Write 0x04 to CONSET to set the AA bit.
7. Write 0x08 to CONCLR to clear the SI flag.
8. Increment Master Transmit buffer pointer
9. Exit

#### 12.11.6.4 State: 0x30

Data has been transmitted, NOT ACK received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.6.5 State: 0x38

Arbitration has been lost during Slave Address + Write or data. The bus has been released and not addressed Slave mode is entered. A new START condition will be transmitted when the bus is free again.

1. Write 0x24 to CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

### 12.11.7 Master Receive states

#### 12.11.7.1 State: 0x40

Previous state was State 08 or State 10. Slave Address + Read has been transmitted, ACK has been received. Data will be received and ACK returned.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.7.2 State: 0x48

Slave Address + Read has been transmitted, NOT ACK has been received. A STOP condition will be transmitted.

1. Write 0x14 to CONSET to set the STO and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.7.3 State: 0x50

Data has been received, ACK has been returned. Data will be read from DAT. Additional data will be received. If this is the last data byte then NOT ACK will be returned, otherwise ACK will be returned.

1. Read data byte from DAT into Master Receive buffer.
2. Decrement the Master data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to CONCLR to clear the SI flag and the AA bit.

4. Exit
5. Write 0x04 to CONSET to set the AA bit.
6. Write 0x08 to CONCLR to clear the SI flag.
7. Increment Master Receive buffer pointer
8. Exit

#### 12.11.7.4 State: 0x58

Data has been received, NOT ACK has been returned. Data will be read from DAT. A STOP condition will be transmitted.

1. Read data byte from DAT into Master Receive buffer.
2. Write 0x14 to CONSET to set the STO and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Exit

### 12.11.8 Slave Receiver states

#### 12.11.8.1 State: 0x60

Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

#### 12.11.8.2 State: 0x68

Arbitration has been lost in Slave Address and R/W bit as bus Master. Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK will be returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit.

#### 12.11.8.3 State: 0x70

General call has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.

4. Initialize Slave data counter.
5. Exit

#### 12.11.8.4 State: 0x78

Arbitration has been lost in Slave Address + R/W bit as bus Master. General call has been received and ACK has been returned. Data will be received and ACK returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to CONSET to set the STA and AA bits.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

#### 12.11.8.5 State: 0x80

Previously addressed with own Slave Address. Data has been received and ACK has been returned. Additional data will be read.

1. Read data byte from DAT into the Slave Receive buffer.
2. Decrement the Slave data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to CONCLR to clear the SI flag and the AA bit.
4. Exit.
5. Write 0x04 to CONSET to set the AA bit.
6. Write 0x08 to CONCLR to clear the SI flag.
7. Increment Slave Receive buffer pointer.
8. Exit

#### 12.11.8.6 State: 0x88

Previously addressed with own Slave Address. Data has been received and NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.8.7 State: 0x90

Previously addressed with General Call. Data has been received, ACK has been returned. Received data will be saved. Only the first data byte will be received with ACK. Additional data will be received with NOT ACK.

1. Read data byte from DAT into the Slave Receive buffer.
2. Write 0x0C to CONCLR to clear the SI flag and the AA bit.
3. Exit

#### 12.11.8.8 State: 0x98

Previously addressed with General Call. Data has been received, NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

#### 12.11.8.9 State: 0xA0

A STOP condition or Repeated START has been received, while still addressed as a Slave. Data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

### 12.11.9 Slave Transmitter states

#### 12.11.9.1 State: 0xA8

Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received.

1. Load DAT from Slave Transmit buffer with first data byte.
2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

#### 12.11.9.2 State: 0xB0

Arbitration lost in Slave Address and R/W bit as bus Master. Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received. STA is set to restart Master mode after the bus is free again.

1. Load DAT from Slave Transmit buffer with first data byte.
2. Write 0x24 to CONSET to set the STA and AA bits.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

#### 12.11.9.3 State: 0xB8

Data has been transmitted, ACK has been received. Data will be transmitted, ACK bit will be received.

1. Load DAT from Slave Transmit buffer with data byte.

2. Write 0x04 to CONSET to set the AA bit.
3. Write 0x08 to CONCLR to clear the SI flag.
4. Increment Slave Transmit buffer pointer.
5. Exit

#### 12.11.9.4 State: 0xC0

Data has been transmitted, NOT ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit.

#### 12.11.9.5 State: 0xC8

The last data byte has been transmitted, ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to CONSET to set the AA bit.
2. Write 0x08 to CONCLR to clear the SI flag.
3. Exit

## 13.1 How to read this chapter

The C_CAN block is available in LPC11Cxx parts only (LPC11C00 series).

The LPC11C22 and LPC11C24 parts include an on-chip, high-speed transceiver. For these parts, the CAN_RXD and CAN_TXD signals are connected internally to the on-chip transceiver, and the transceiver signals are pinned out (see Table 178).

## 13.2 Basic configuration

The C_CAN is configured using the following registers:

1. Power: In the SYSAHBCLKCTRL register, set bit 17 (Table 20).
2. Clocking: For an accurate peripheral clock to the C_CAN block, select the system oscillator either as the main clock (Table 17) or as input to the system PLL (Table 15). Do not select the IRC if C_CAN baud rates above 100 kbit/s are required.
3. Reset: Before accessing the C_CAN block, ensure that the CAN_RST_N bit (bit 3) in the PRESETCTRL register (Table 8) is set to 1. This de-asserts the reset signal to the C_CAN block.

The peripheral clock to the C_CAN (the C_CAN system clock) and to the programmable C_CAN clock divider (see Table 209) is provided by the system clock (seeTable 20). This clock can be disabled through bit 17 in the SYSAHBCLKCTRL register for power savings.

**Remark:** If C_CAN 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.

## 13.3 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.

## 13.4 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.

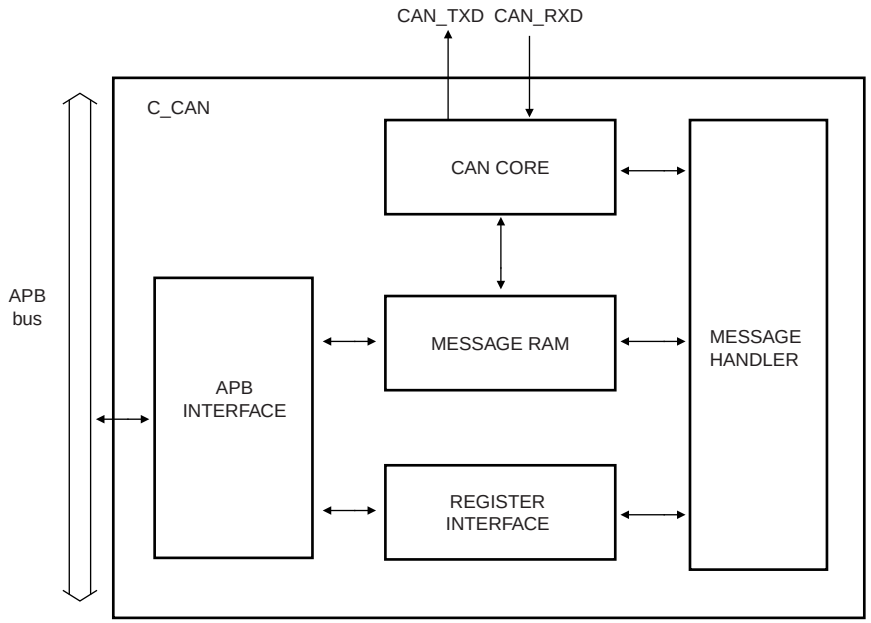



**Fig 45. C_CAN block diagram**

## 13.5 Pin description

**Table 177. CAN pin description (LPC11C12/C14)**

| Pin | Type | Description |
|---|---|---|
| CAN_TXD | O | C_CAN transmit output |
| CAN_RXD | I | C_CAN receive input |

**Table 178. CAN pin description (LPC11C22/C24)**

| Pin | Type | Description |
|---|---|---|
| CANL | I/O | LOW-level CAN bus line. |
| CANH | I/O | HIGH-level CAN bus line. |
| STB | I | Silent mode control input for CAN transceiver (LOW = Normal mode, HIGH = silent mode). |
| VDD_CAN | - | Supply voltage for I/O level of CAN transceiver. |
| $V_{CC}$ | - | Supply voltage for CAN transceiver. |
| GND | - | Ground for CAN transceiver. |

## 13.6 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 179. Register overview: CCAN (base address 0x4005 0000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| CANCNTL | R/W | 0x000 | CAN control | 0x0001 |
| CANSTAT | R/W | 0x004 | Status register | 0x0000 |
| CANEC | RO | 0x008 | Error counter | 0x0000 |
| CANBT | R/W | 0x00C | Bit timing register | 0x2301 |
| CANINT | RO | 0x010 | Interrupt register | 0x0000 |
| CANTEST | R/W | 0x014 | Test register | - |
| CANBRPE | R/W | 0x018 | Baud rate prescaler extension register | 0x0000 |
| - | - | 0x01C | Reserved | - |
| CANIF1_CMDREQ | R/W | 0x020 | Message interface 1 command request | 0x0001 |
| CANIF1_CMDMSK | R/W | 0x024 | Message interface 1 command mask (write direction) | 0x0000 |
| CANIF1_CMDMSK | R/W | 0x024 | Message interface 1 command mask (read direction) | 0x0000 |
| CANIF1_MSK1 | R/W | 0x028 | Message interface 1 mask 1 | 0xFFFF |
| CANIF1_MSK2 | R/W | 0x02C | Message interface 1 mask 2 | 0xFFFF |
| CANIF1_ARB1 | R/W | 0x030 | Message interface 1 arbitration 1 | 0x0000 |
| CANIF1_ARB2 | R/W | 0x034 | Message interface 1 arbitration 2 | 0x0000 |
| CANIF1_MCTRL | R/W | 0x038 | Message interface 1 message control | 0x0000 |

**Table 179. Register overview: CCAN (base address 0x4005 0000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| CANIF1_DA1 | R/W | 0x03C | Message interface 1 data A1 | 0x0000 |
| CANIF1_DA2 | R/W | 0x040 | Message interface 1 data A2 | 0x0000 |
| CANIF1_DB1 | R/W | 0x044 | Message interface 1 data B1 | 0x0000 |
| CANIF1_DB2 | R/W | 0x048 | Message interface 1 data B2 | 0x0000 |
| - | - | 0x04C - 0x07C | Reserved | - |
| CANIF2_CMDREQ | R/W | 0x080 | Message interface 2 command request (write direction) | 0x0001 |
| CANIF2_CMDREQ | R/W | 0x080 | Message interface 2 command request (read direction) | 0x0001 |
| CANIF2_CMDMSK | R/W | 0x084 | Message interface 2 command mask | 0x0000 |
| CANIF2_MSK1 | R/W | 0x088 | Message interface 2 mask 1 | 0xFFFF |
| CANIF2_MSK2 | R/W | 0x08C | Message interface 2 mask 2 | 0xFFFF |
| CANIF2_ARB1 | R/W | 0x090 | Message interface 2 arbitration 1 | 0x0000 |
| CANIF2_ARB2 | R/W | 0x094 | Message interface 2 arbitration 2 | 0x0000 |
| CANIF2_MCTRL | R/W | 0x098 | Message interface 2 message control | 0x0000 |
| CANIF2_DA1 | R/W | 0x09C | Message interface 2 data A1 | 0x0000 |
| CANIF2_DA2 | R/W | 0x0A0 | Message interface 2 data A2 | 0x0000 |
| CANIF2_DB1 | R/W | 0x0A4 | Message interface 2 data B1 | 0x0000 |
| CANIF2_DB2 | R/W | 0x0A8 | Message interface 2 data B2 | 0x0000 |
| - | - | 0x0AC - 0x0FC | Reserved | - |
| CANTXREQ1 | RO | 0x100 | Transmission request 1 | 0x0000 |
| CANTXREQ2 | RO | 0x104 | Transmission request 2 | 0x0000 |
| - | - | 0x108 - 0x11C | Reserved | - |
| CANND1 | RO | 0x120 | New data 1 | 0x0000 |
| CANND2 | RO | 0x124 | New data 2 | 0x0000 |
| - | - | 0x128 - 0x13C | Reserved | - |
| CANIR1 | RO | 0x140 | Interrupt pending 1 | 0x0000 |
| CANIR2 | RO | 0x144 | Interrupt pending 2 | 0x0000 |
| - | - | 0x148 - 0x15C | Reserved | - |
| CANMSGV1 | RO | 0x160 | Message valid 1 | 0x0000 |
| CANMSGV2 | RO | 0x164 | Message valid 2 | 0x0000 |
| - | - | 0x168 - 0x17C | Reserved | - |
| CANCLKDIV | R/W | 0x180 | Can clock divider register | 0x0000 |

### 13.6.1 CAN protocol registers

#### 13.6.1.1 CAN control register

The reset value 0x0001 of the CANCTRL 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 180. CAN control registers (CANCNTL, address 0x4005 0000) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 0 | INIT | | Initialization | 1 | R/W |
| | | 0 | Normal operation. | | |
| | | 1 | 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. | | |
| 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 | Automatic retransmission of disturbed messages enabled. | | |
| | | 1 | Automatic retransmission disabled. | | |
| 6 | CCE | | Configuration change enable | 0 | R/W |
| | | 0 | The CPU has no write access to the bit timing register. | | |
| | | 1 | 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.

#### 13.6.1.2 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 CANSTAT register will clear the Status Interrupt value (0x8000) in the CANIR register.

**Table 181. CAN status register (CANSTAT, address 0x4005 0004) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 2:0 | LEC | | Last error code<br>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.<br>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<br>This bit is reset by the CPU. It is never reset by the CAN controller. | 0 | R/W |
| | | 0 | Since this bit was reset by the CPU, no message has been successfully transmitted. | | |
| | | 1 | 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<br>This bit is reset by the CPU. It is never reset by the CAN controller. | 0 | R/W |
| | | 0 | Since this bit was last reset by the CPU, no message has been successfully transmitted. | | |
| | | 1 | Since this bit was last set to zero by the CPU, a message has been successfully received independent of the result of acceptance filtering. | | |
| 5 | EPASS | | Error passive | 0 | RO |
| | | 0 | The CAN controller is in the error active state. | | |
| | | 1 | The CAN controller is in the error passive state as defined in the *CAN 2.0 specification*. | | |

**Table 181. CAN status register (CANSTAT, address 0x4005 0004) bit description**

*…continued*

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 6 | EWARN | | Warning status | 0 | RO |
| | | 0 | Both error counters are below the error warning limit of 96. | | |
| | | 1 | At least one of the error counters in the EML 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 | | |

### 13.6.1.3 CAN error counter

**Table 182. CAN error counter (CANEC, address 0x4005 0008) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 7:0 | TEC[7:0] | | Transmit error counter<br>Current value of the transmit error counter (maximum value 255) | 0 | RO |
| 14:8 | REC[6:0] | | Receive error counter<br>Current value of the receive error counter (maximum value 127). | - | RO |
| 15 | RP | | Receive error passive | - | RO |
| | | 0 | The receive counter is below the error passive level. | | |
| | | 1 | The receive counter has reached the error passive level as defined in the *CAN2.0 specification*. | | |
| 31:16 | - | - | Reserved | - | - |

#### 13.6.1.4 CAN bit timing register

**Table 183. CAN bit timing register (CANBT, address 0x4005 000C) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 5:0 | BRP | Baud rate prescaler<br>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.[1] | 000001 | R/W |
| 7:6 | SJW | (Re)synchronization jump width<br>Valid programmed values are 0 to 3.[1] | 00 | R/W |
| 11:8 | TSEG1 | Time segment before the sample point<br>Valid values are 1 to 15.[1] | 0011 | R/W |
| 14:12 | TSEG2 | Time segment after the sample point<br>Valid values are 0 to 7.[1] | 010 | R/W |
| 31:15 | - | Reserved | - | - |

[1] Hardware interprets the value programmed into these bits as the bit value + 1.

For example, with a LPC11Cx 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 13.7.5 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 LPC11Cx 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:

$t_{SJW} = t_q \times (SJW + 1)$

### 13.6.1.5 CAN interrupt register

**Table 184. CAN interrupt register (CANINT, address 0x4005 0010) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | INTID | 0x0000 = No interrupt is pending.<br>0x0001 - 0x0020 = Number of message object which caused the interrupt.<br>0x0021 - 0x7FFF = Unused<br>0x8000 = Status interrupt<br>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.

### 13.6.1.6 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 185. CAN test register (CANTEST, address 0x4005 0014) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 1:0 | - | - | Reserved | | - |
| 2 | BASIC | | Basic mode | 0 | R/W |
| | | 0 | Basic mode disabled. | | |
| | | 1 | IF1 registers used as TX buffer, IF2 registers used as RX buffer. | | |
| 3 | SILENT | | Silent mode | 0 | R/W |
| | | 0 | Normal operation. | | |
| | | 1 | The module is in silent mode. | | |
| 4 | LBACK | | Loop back mode | 0 | R/W |
| | | 0 | Loop back mode is disabled. | | |
| | | 1 | Loop back mode is enabled. | | |

**Table 185. CAN test register (CANTEST, address 0x4005 0014) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 6:5 | TX | | Control of CAN_TXD pins | 00 | R/W |
| | | 0x0 | Level at the CAN_TXD pin is controlled by the CAN controller. This is the value at reset. | | |
| | | 0x1 | The sample point can be monitored at the CAN_TXD pin. | | |
| | | 0x2 | CAN_TXD pin is driven LOW/dominant. | | |
| | | 0x3 | CAN_TXD pin is driven HIGH/recessive. | | |
| 7 | RX | | Monitors the actual value of the CAN_RXD pin. | 0 | R |
| | | 0 | The CAN bus is recessive (CAN_RXD = '1'). | | |
| | | 1 | The CAN bus is dominant (CAN_RXD = '0'). | | |
| 31:8 | - | | R/W | | - |

#### 13.6.1.7 CAN baud rate prescaler extension register

**Table 186. CAN baud rate prescaler extension register (CANBRPE, address 0x4005 0018) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 3:0 | BRPE | Baud rate prescaler extension<br>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 | - | - |

### 13.6.2 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 13.6.2.1) 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 187. Message interface registers**

| IF1 register names | IF1 register set | IF2 register names | IF2 register set |
|---|---|---|---|
| CANIF1_CMDREQ | IF1 command request | CANIF2_CMDREQ | IF2 command request |
| CANIF1_CMDMASK | IF1 command mask | CANIF2_CMDMASK | IF2 command mask |
| CANIF1_MASK1 | IF1 mask 1 | CANIF2_MSK1 | IF2 mask 1 |
| CANIF1_MASK2 | IF1 mask 2 | CANIF2_MSK2 | IF2 mask 2 |
| CANIF1_ARB1 | IF1 arbitration 1 | CANIF2_ARB1 | IF2 arbitration 1 |
| CANIF1_ARB2 | IF1 arbitration 2 | CANIF2_ARB2 | IF2 arbitration 2 |
| CANIF1_MCTRL | IF1 message control | CANIF2_MCTRL | IF2 message control |
| CANIF1_DA1 | IF1 data A1 | CANIF2_DA1 | IF2 data A1 |
| CANIF1_DA2 | IF1 data A2 | CANIF2_DA2 | IF2 data A2 |
| CANIF1_DB1 | IF1 data B1 | CANIF2_DB1 | IF2 data B1 |
| CANIF1_DB2 | IF1 data B2 | CANIF2_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 13.7.3.

### 13.6.2.1 Message objects

A message object contains the information from the various bits in the message interface registers. Table 188 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 188. 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 | | | |

### 13.6.2.2 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 189. CAN message interface command request registers (CANIF1_CMDREQ, address 0x4005 0020 and CANIF2_CMDREQ, address 0x4005 0080) bit description**

| Bit | Symbol | Value | Description | Reset Value | Access |
|---|---|---|---|---|---|
| 5:0 | MN | | Message number<br>0x01 - 0x20 = Valid message numbers. The message object in the message RAM is selected for data transfer.<br>0x00 = Not a valid message number. This value is interpreted as 0x20.[1]<br>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 | Set to zero by hardware when read/write action to this Command request register has finished. | | |
| | | 1 | 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.

### 13.6.2.3 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 190. CAN message interface command mask registers (CANIF1_CMDMSK, address 0x4005 0024 and CANIF2_CMDMSK, address 0x4005 0084) bit description - write direction**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 0 | DATA_B | | Access data bytes 4-7 | 0 | R/W |
| | | 0 | Data bytes 4-7 unchanged. | | |
| | | 1 | Transfer data bytes 4-7 to message object. | | |
| 1 | DATA_A | | Access data bytes 0-3 | 0 | R/W |
| | | 0 | Data bytes 0-3 unchanged. | | |
| | | 1 | Transfer data bytes 0-3 to message object. | | |

**Table 190. CAN message interface command mask registers (CANIF1_CMDMSK, address 0x4005 0024 and CANIF2_CMDMSK, address 0x4005 0084) bit description - write direction** *...continued*

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 2 | TXRQST | | Access transmission request bit | 0 | R/W |
| | | 0 | No transmission request. TXRQSRT bit unchanged in IF1/2_MCTRL.<br>**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 | Control bits unchanged. | | |
| | | 1 | Transfer control bits to message object | | |
| 5 | ARB | | Access arbitration bits | 0 | R/W |
| | | 0 | Arbitration bits unchanged. | | |
| | | 1 | Transfer Identifier, DIR, XTD, and MSGVAL bits to message object. | | |
| 6 | MASK | | Access mask bits | 0 | R/W |
| | | 0 | Mask bits unchanged. | | |
| | | 1 | Transfer Identifier MASK + MDIR + MXTD to message object. | | |
| 7 | WR/RD | 1 | Write transfer<br>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 191. CAN message interface command mask registers (CANIF1_CMDMSK, address 0x4005 0024 and CANIF2_CMDMSK, address 0x4005 0084) bit description - read direction**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 0 | DATA_B | | Access data bytes 4-7 | 0 | R/W |
| | | 0 | Data bytes 4-7 unchanged. | | |
| | | 1 | Transfer data bytes 4-7 to IFx message buffer register. | | |
| 1 | DATA_A | | Access data bytes 0-3 | 0 | R/W |
| | | 0 | Data bytes 0-3 unchanged. | | |
| | | 1 | Transfer data bytes 0-3 to IFx message buffer. | | |

**Table 191. CAN message interface command mask registers (CANIF1_CMDMSK, address 0x4005 0024 and CANIF2_CMDMSK, address 0x4005 0084) bit description - read direction** *…continued*

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 2 | NEWDAT | | Access new data bit | 0 | R/W |
| | | 0 | NEWDAT bit remains unchanged.<br>**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 NEWDAT bit in the message object. | | |
| 3 | CLRINTPND | | Clear interrupt pending bit. | 0 | R/W |
| | | 0 | INTPND bit remains unchanged. | | |
| | | 1 | Clear INTPND bit in the message object. | | |
| 4 | CTRL | | Access control bits | 0 | R/W |
| | | 0 | Control bits unchanged. | | |
| | | 1 | Transfer control bits to IFx message buffer. | | |
| 5 | ARB | | Access arbitration bits | 0 | R/W |
| | | 0 | Arbitration bits unchanged. | | |
| | | 1 | Transfer Identifier, DIR, XTD, and MSGVAL bits to IFx message buffer register. | | |
| 6 | MASK | | Access mask bits | 0 | R/W |
| | | 0 | Mask bits unchanged. | | |
| | | 1 | Transfer Identifier MASK + MDIR + MXTD to IFx message buffer register. | | |
| 7 | WR/RD | 0 | Read transfer<br>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 | - |

### 13.6.2.4 IF1 and IF2 message buffer registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM.

#### 13.6.2.4.1 CAN message interface command mask 1 registers

**Table 192. CAN message interface command mask 1 registers (CANIF1_MSK1, address 0x4005 0028 and CANIF2_MASK1, address 0x4005 0088) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 15:0 | MSK[15:0] | | Identifier mask | 0xFFFF | R/W |
| | | 0 | The corresponding bit in the identifier of the message can not inhibit the match in the acceptance filtering. | | |
| | | 1 | The corresponding identifier bit is used for acceptance filtering. | | |
| 31:16 | - | - | reserved | 0 | - |

#### 13.6.2.4.2 CAN message interface command mask 2 registers

**Table 193. CAN message interface command mask 2 registers (CANIF1_MSK2, address 0x4005 002C and CANIF2_MASK2, address 0x4005 008C) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 12:0 | MSK[28:16] | | Identifier mask | 0xFFF | R/W |
| | | 0 | The corresponding bit in the identifier of the message can not inhibit the match in the acceptance filtering. | | |
| | | 1 | The corresponding identifier bit is used for acceptance filtering. | | |
| 13 | - | | Reserved | 1 | - |
| 14 | MDIR | | Mask message direction | 1 | R/W |
| | | 0 | The message direction bit (DIR) has no effect on acceptance filtering. | | |
| | | 1 | The message direction bit (DIR) is used for acceptance filtering. | | |
| 15 | MXTD | | Mask extend identifier | 1 | R/W |
| | | 0 | The extended identifier bit (XTD) has no effect on acceptance filtering. | | |
| | | 1 | The extended identifier bit (XTD) is used for acceptance filtering. | | |
| 31:16 | - | - | Reserved | 0 | - |

#### 13.6.2.4.3 CAN message interface command arbitration 1 registers

**Table 194. CAN message interface command arbitration 1 registers (CANIF1_ARB1, address 0x4005 0030 and CANIF2_ARB1, address 0x4005 0090) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | ID[15:0] | Message identifier<br>29-bit identifier (extended frame)<br>11-bit identifier (standard frame) | 0x00 | R/W |
| 31:16 | - | Reserved | 0 | - |

#### 13.6.2.4.4 CAN message interface command arbitration 2 registers

**Table 195. CAN message interface command arbitration 2 registers (CANIF1_ARB2, address 0x4005 0034 and CANIF2_ARB2, address 0x4005 0094) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 12:0 | ID[28:16] | | Message identifier<br>29-bit identifier (extended frame)<br>11-bit identifier (standard frame) | 0x00 | R/W |
| 13 | DIR | | Message direction | 0x00 | R/W |
| | | 0 | Direction = receive.<br>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 | Direction = transmit.<br>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 | The 11-bit standard identifier will be used for this message object. | | |
| | | 1 | The 29-bit extended identifier will be used for this message object. | | |
| 15 | MSGVAL | | Message valid<br>**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 | The message object is ignored by the message handler. | | |
| | | 1 | The message object is configured and should be considered by the message handler. | | |
| 31:16 | - | - | Reserved | 0 | - |

#### 13.6.2.4.5 CAN message interface message control registers

**Table 196. CAN message interface message control registers (CANIF1_MCTRL, address 0x4005 0038 and CANIF2_MCTRL, address 0x4005 0098) bit description**

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 3:0 | DLC[3:0] | | Data length code<br>**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.<br>0000 - 1000 = Data frame has 0 - 8 data bytes.<br>1001 - 1111 = Data frame has 8 data bytes. | 0000 | R/W |
| 6:4 | - | | reserved | - | - |
| 7 | EOB | | End of buffer | 0 | R/W |
| | | 0 | Message object belongs to a FIFO buffer and is not the last message object of that FIFO buffer. | | |
| | | 1 | Single message object or last message object of a FIFO buffer. | | |

**Table 196. CAN message interface message control registers (CANIF1_MCTRL, address 0x4005 0038 and CANIF2_MCTRL, address 0x4005 0098) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value | Access |
|---|---|---|---|---|---|
| 8 | TXRQST | | Transmit request | 0 | R/W |
| | | 0 | This message object is not waiting for transmission. | | |
| | | 1 | The transmission of this message object is requested and is not yet done | | |
| 9 | RMTEN | | Remote enable | 0 | R/W |
| | | 0 | At the reception of a remote frame, TXRQST is left unchanged. | | |
| | | 1 | At the reception of a remote frame, TXRQST is set. | | |
| 10 | RXIE | | Receive interrupt enable | 0 | R/W |
| | | 0 | INTPND will be left unchanged after successful reception of a frame. | | |
| | | 1 | INTPND will be set after successful reception of a frame. | | |
| 11 | TXIE | | Transmit interrupt enable | 0 | R/W |
| | | 0 | The INTPND bit will be left unchanged after a successful transmission of a frame. | | |
| | | 1 | INTPND will be set after a successful transmission of a frame. | | |
| 12 | UMASK | | Use acceptance mask<br>**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 | Mask ignored. | | |
| | | 1 | Use mask (MSK[28:0], MXTD, and MDIR) for acceptance filtering. | | |
| 13 | INTPND | | Interrupt pending | 0 | R/W |
| | | 0 | This message object is not the source of an interrupt. | | |
| | | 1 | 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. | | |
| 14 | MSGLST | | Message lost (only valid for message objects in the direction receive). | 0 | R/W |
| | | 0 | No message lost since this bit was reset last by the CPU. | | |
| | | 1 | 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 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 | The message handler or the CPU has written new data into the data portion of this message object. | | |
| 31:16 | - | - | Reserved | 0 | - |

#### 13.6.2.4.6 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 197. CAN message interface data A1 registers (CANIF1_DA1, address 0x4005 003C and CANIF2_DA1, address 0x4005 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 | - | - |

#### 13.6.2.4.7 CAN message interface data A2 registers

**Table 198. CAN message interface data A2 registers (CANIF1_DA2, address 0x4005 0040 and CANIF2_DA2, address 0x4005 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 | - | - |

#### 13.6.2.4.8 CAN message interface data B1 registers

**Table 199. CAN message interface data B1 registers (CANIF1_DB1, address 0x4005 0044 and CANIF2_DB1, address 0x4005 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 | - | - |

#### 13.6.2.4.9 CAN message interface data B2 registers

**Table 200. CAN message interface data B2 registers (CANIF1_DB2, address 0x4005 0048 and CANIF2_DB2, address 0x4005 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 | - | - |

### 13.6.3 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.

### 13.6.3.1 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 201. CAN transmission request 1 register (CANTXREQ1, address 0x4005 0100) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | TXRQST[16:1] | Transmission request bit of message objects 16 to 1.<br>0 = This message object is not waiting for transmission.<br>1 = The transmission of this message object is requested and not yet done. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.2 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 202. CAN transmission request 2 register (CANTXREQ2, address 0x4005 0104) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | TXRQST[32:17] | Transmission request bit of message objects 32 to 17.<br>0 = This message object is not waiting for transmission.<br>1 = The transmission of this message object is requested and not yet done. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.3 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 203. CAN new data 1 register (CANND1, address 0x4005 0120) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | NEWDAT[16:1] | New data bits of message objects 16 to 1.<br>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.<br>1 = The Message Handler or the CPU has written new data into the data portion of this Message Object. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.4 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 204. CAN new data 2 register (CANND2, address 0x4005 0124) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | NEWDAT[32:17] | New data bits of message objects 32 to 17.<br>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.<br>1 = The Message Handler or the CPU has written new data into the data portion of this Message Object. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.5 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 205. CAN interrupt pending 1 register (CANIR1, address 0x4005 0140) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | INTPND[16:1] | Interrupt pending bits of message objects 16 to 1.<br>0 = This message object is ignored by the message handler.<br>1 = This message object is the source of an interrupt. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.6 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 206. CAN interrupt pending 2 register (CANIR2, addresses 0x4005 0144) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | INTPND[32:17] | Interrupt pending bits of message objects 32 to 17.<br>0 = This message object is ignored by the message handler.<br>1 = This message object is the source of an interrupt. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.7 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 207. CAN message valid 1 register (CANMSGV1, addresses 0x4005 0160) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 15:0 | MSGVAL[16:1] | Message valid bits of message objects 16 to 1.<br>0 = This message object is ignored by the message handler.<br>1 = This message object is configured and should be considered by the message handler. | 0x00 | R |
| 31:16 | - | Reserved | - | - |

### 13.6.3.8 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 208. CAN message valid 2 register (CANMSGV2, address 0x4005 0164) bit description**

| Bit | Symbol | Description | Access | Reset value |
|---|---|---|---|---|
| 15:0 | MSGVAL[32:17] | Message valid bits of message objects 32 to 17.<br>0 = This message object is ignored by the message handler.<br>1 = This message object is configured and should be considered by the message handler. | R | 0x00 |
| 31:16 | - | Reserved | - | - |

### 13.6.4 CAN timing register

#### 13.6.4.1 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.

**Table 209. CAN clock divider register (CANCLKDIV, address 0x4005 0180) bit description**

| Bit | Symbol | Description | Reset value | Access |
|---|---|---|---|---|
| 3:0 | CLKDIVVAL | Clock divider value. CAN_CLK = PCLK/(CLKDIVVAL +1)<br>0000: CAN_CLK = PCLK divided by 1.<br>0001: CAN_CLK = PCLK divided by 2.<br>0010: CAN_CLK = PCLK divided by 3<br>0010: CAN_CLK = PCLK divided by 4.<br>...<br>1111: CAN_CLK = PCLK divided by 16. | 0000 | R/W |
| 31:4 | - | reserved | - | - |

## 13.7 Functional description

### 13.7.1 C_CAN controller state after reset

After a hardware reset, the registers hold the values described in Table 179. 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.

### 13.7.2 C_CAN operating modes

#### 13.7.2.1 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 EML 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.

#### 13.7.2.2 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.

#### 13.7.2.3 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.

#### 13.7.2.4 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.

##### 13.7.2.4.1 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).

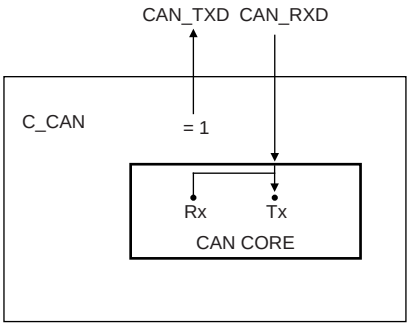



**Fig 46. CAN core in Silent mode**

##### 13.7.2.4.2 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.

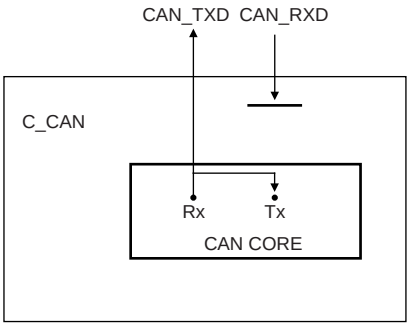



**Fig 47. CAN core in Loop-back mode**

#### 13.7.2.4.3 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.

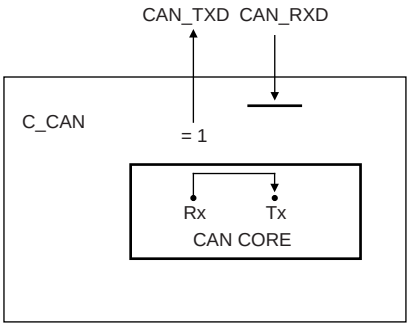



**Fig 48. CAN core in Loop-back mode combined with Silent mode**

#### 13.7.2.4.4 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')

#### 13.7.2.4.5 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 13.6.1.6.

**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.

### 13.7.3 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 49.

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

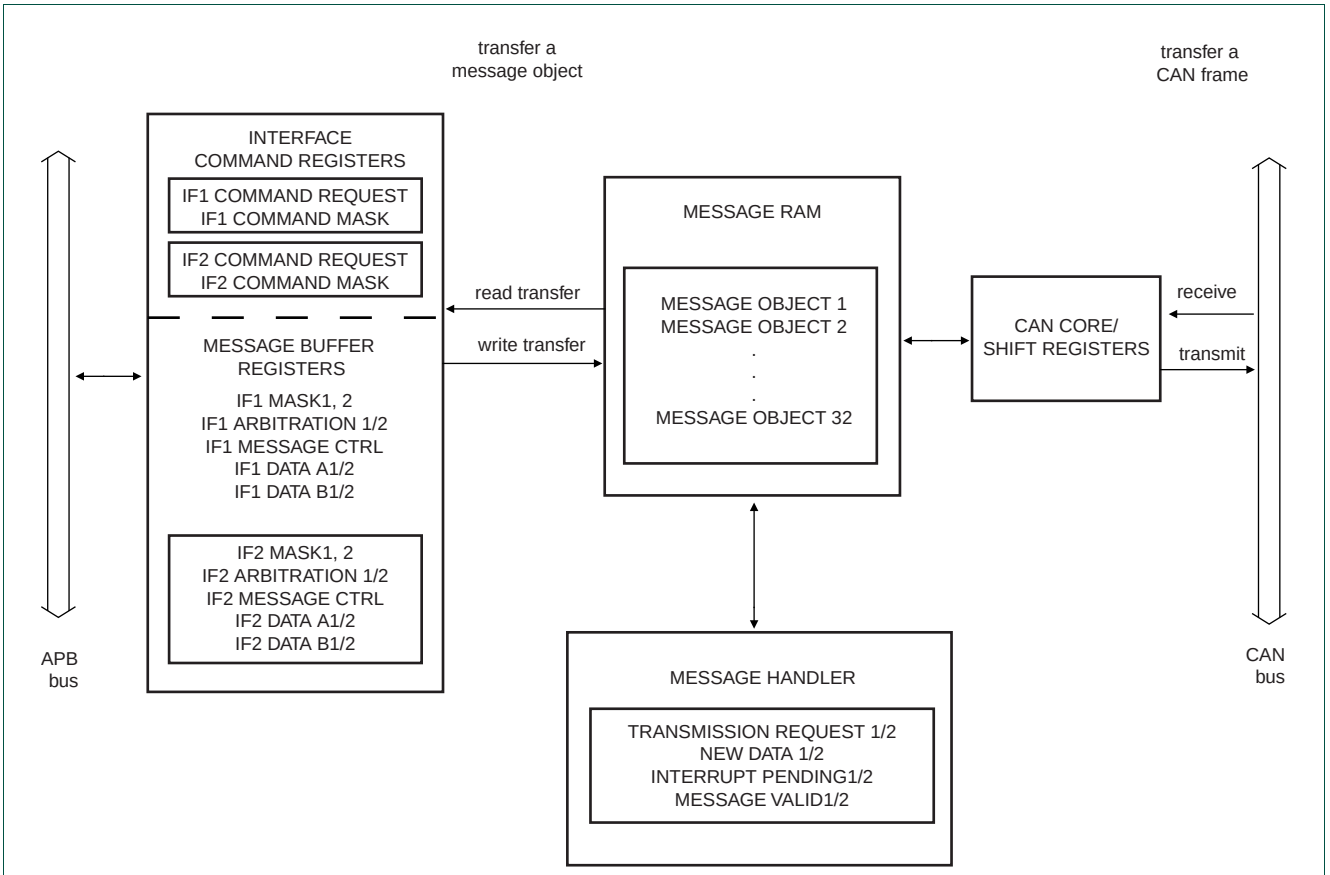



**Fig 49. Block diagram of a message object transfer**

### 13.7.3.1 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.

#### 13.7.3.2 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.

#### 13.7.3.3 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.

#### 13.7.3.4 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.

##### 13.7.3.4.1 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.

##### 13.7.3.4.2 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.

### 13.7.3.5 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.

### 13.7.3.6 Configuration of a transmit object

Table 210 shows how a transmit object should be initialized by software (see also Table 188):

**Table 210. 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 13.7.3.4.2. The DIR bit should not be masked.

### 13.7.3.7 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 13.7.3.3.

When NEWDAT is set together with TXRQST, NEWDAT will be reset as soon as the new transmission has started.

### 13.7.3.8 Configuration of a receive object

Table 211 shows how a receive object should be initialized by software (see also Table 188)

**Table 211. 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 13.7.3.4.1. The DIR bit should not be masked in typical applications.

### 13.7.3.9 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.

#### 13.7.3.10 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 13.7.3.8.

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.

##### 13.7.3.10.1 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.

##### 13.7.3.10.2 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.

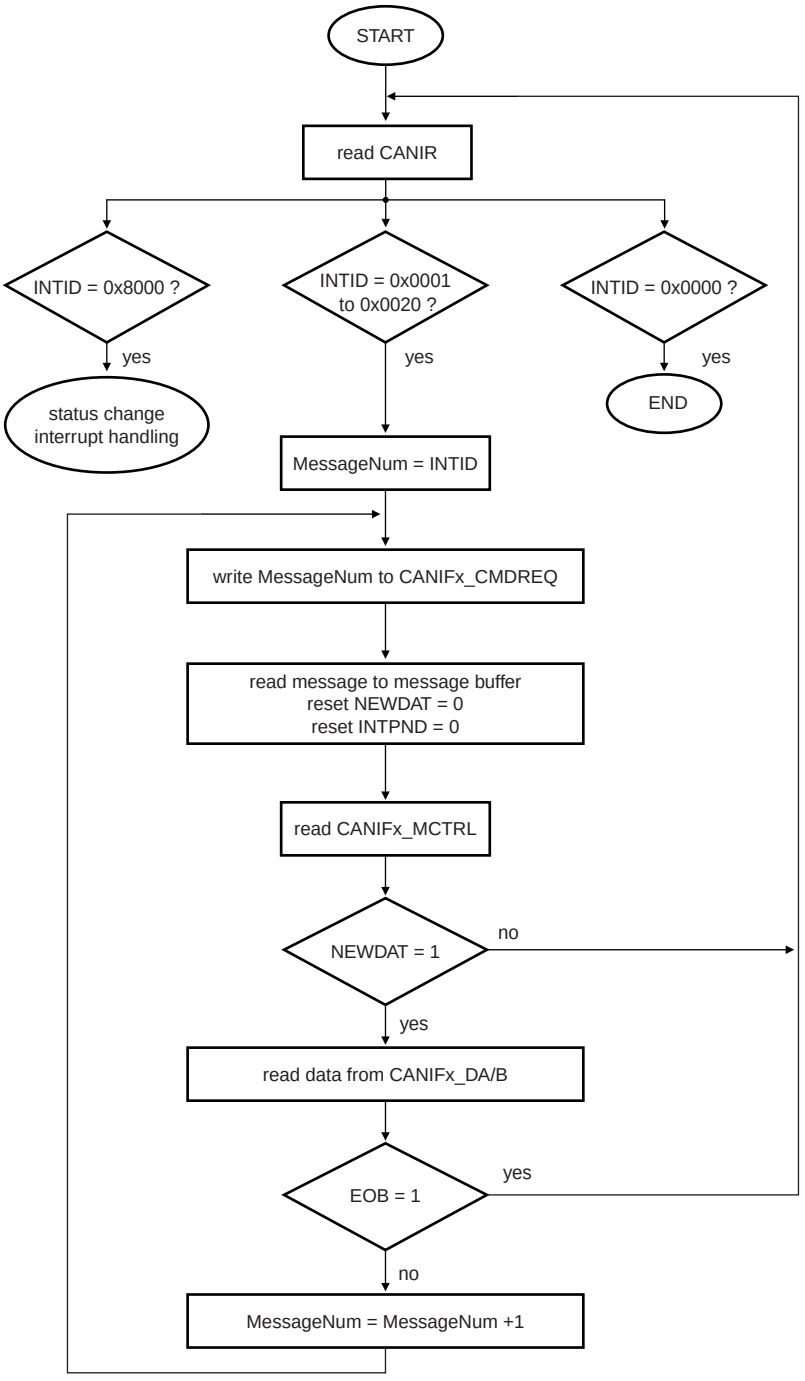



**Fig 50. Reading a message from the FIFO buffer to the message buffer**

### 13.7.4 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 13.6.3.5.

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.

### 13.7.5 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.

### 13.7.5.1 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 51). 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 212). 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 LPC11Cx system clock (see Section 13.2).

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 212 describes the minimum programmable ranges required by the CAN protocol. Bit time parameters are programmed through the CANBT register, Table 183. For details on bit timing and examples, see the *C_CAN user's manual, revision 1.2*.

**Table 212. 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. |

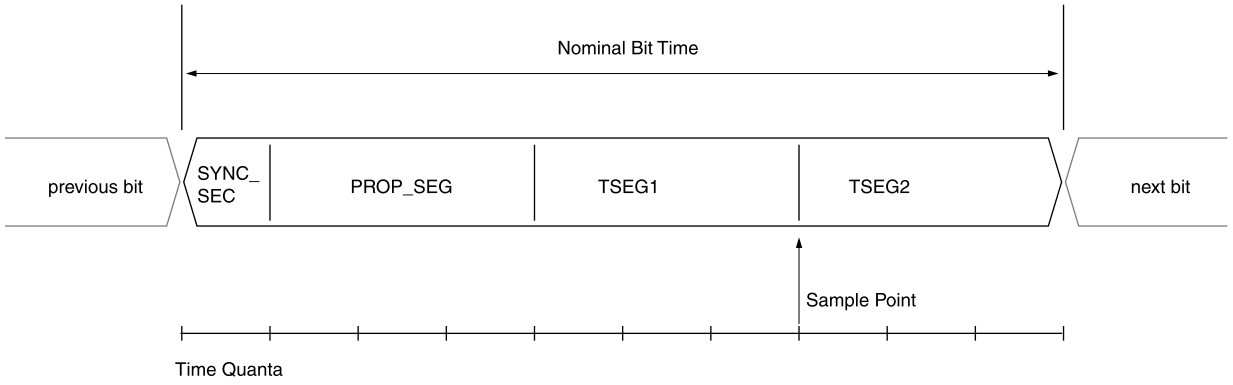



**Fig 51. Bit timing**

## 14.1 How to read this chapter

The C_CAN block is available in LPC11Cxx parts only (LPC11C00 series).

## 14.2 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

## 14.3 General description

In addition to the CAN ISP, the boot ROM provides a CAN and CANopen API to simplify CAN application development. It covers initialization, configuration, basic CAN send/receive as well as a CANopen SDO interface. Callback functions are available to process receive events.

### 14.3.1 Differences to fully-compliant CANopen

While the bootloader uses the SDO communication protocol and the Object Dictionary data organization method, it is not a fully CiA 301 standard compliant CANopen node. In particular, the following features are not available or different to the standard:

- No Network Management (NMT) message processing.
- No Heartbeat Message, no entry 0x1017.
- Uses proprietary SDO Abort Codes to indicate device errors
- “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. This to speed up the communication.
- Entry [1018h,1] Vendor ID reads 0x0000 0000 rather than an official CiA-assigned unique Vendor ID. This in particular because the chip will be incorporated into designs of customers who will become the “vendor” of the whole device. The host will have to use a different method to identify the CAN ISP devices.

## 14.4 API description

### 14.4.1 Calling the C_CAN API

A fixed location in ROM contains a pointer to the ROM driver table i.e. 0x1FFF 1FF8. This location is the same for all LPC11Cxx parts. The ROM driver table contains pointer to the CAN API table. Pointers to the various CAN API functions are stored in this table. CAN API functions can be called by using a C structure.

Figure 52 illustrates the pointer mechanism used to access the on-chip CAN API. On-chip RAM from address 0x1000 0050 to 0x1000 00B8 is used by the CAN API. This address range should not be used by the application. For applications using the on-chip CAN API, the linker control file should be modified appropriately to prevent usage of this area for application's variable storage.

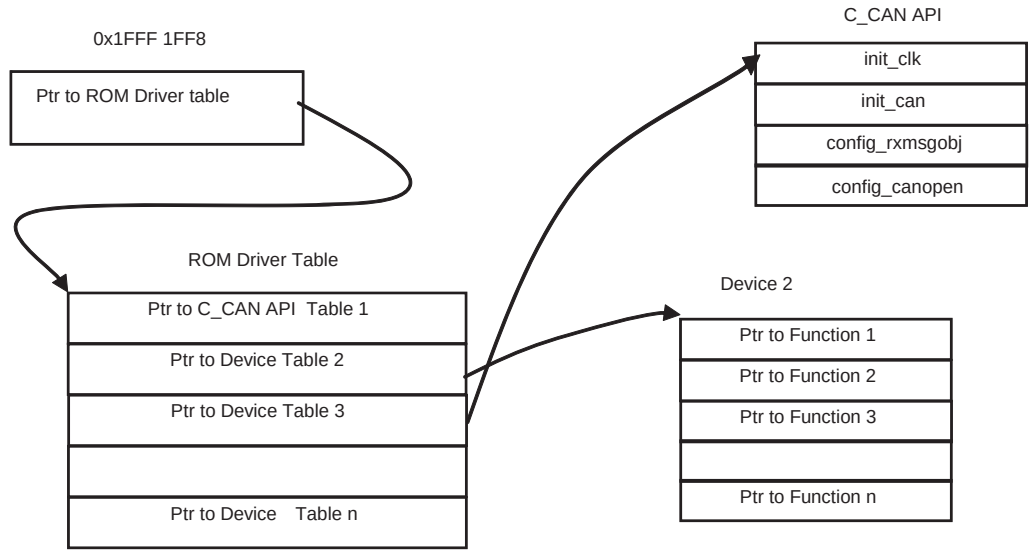



**Fig 52. CAN API pointer structure**

In C, the structure with the function list that is referenced to call the API functions looks as follows:

```
typedef struct _CAND {
  void    (*init_can)        (uint32_t * can_cfg);
  void    (*isr)             (void);
  void    (*config_rxmsgobj) (CAN_MSG_OBJ * msg_obj);
  uint8_t (*can_receive)     (CAN_MSG_OBJ * msg_obj);
  void    (*can_transmit)    (CAN_MSG_OBJ * msg_obj);
  void    (*config_canopen)  (CAN_CANOPENCFG * canopen_cfg);
  void    (*canopen_handler) (void);
  void    (*config_calb)     (CAN_CALLBACKS * callback_cfg);
}  CAND;
```

### 14.4.2 CAN initialization

The CAN controller clock divider, the CAN bit rate is set, and the CAN controller is initialized based on an array of register values that are passed on via a pointer.

```
void init_can (uint32_t * can_cfg, uint8_t isr_ena)
```

The first 32-bit value in the array is applied to the CANCLKDIV register, the second value is applied to the CAN_BTR register.

The second parameter enables interrupts on the CAN controller level. Set to FALSE for polled communication.

Example call:

```
ROM **rom = (ROM **)0x1fff1ff8;

uint32_t CanApiClkInitTable[2] = {
  0x00000000UL, // CANCLKDIV
  0x00004DC5UL // CAN_BTR
};

(*rom)->pCANAPI->init_can(&CanApiCanInitTable[0]);
```

### 14.4.3 CAN interrupt handler

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 CAN interrupt. In order to process CAN events and call the callback functions the application must call the CAN API interrupt handler directly from the interrupt handler routine. The CAN API interrupt handler takes appropriate action according to the data received and the status detected on the CAN bus.

```
void isr (void)
```

The CAN interrupt handler does not process CANopen messages.

Example call:

```
(*rom)->pCAND->isr();
```

For polled communication, the interrupt handler may be called manually as often as required. The callback functions for receive, transmit, and error will be executed as described and on the same level the interrupt handler was called from.

### 14.4.4 CAN Rx message object configuration

The CAN API supports and uses the full CAN model with 32 message objects. Any of the message objects can be used for receive or transmit of either 11-bit or 29-bit CAN messages. CAN messages that have their RTR-bit set (remote transmit) are also supported. For receive objects, a mask pattern for the message identifier allows to receive ranges of messages, up to receiving all CAN messages on the bus in a single message object. See also Section 13.7.3.4.

Transmit message objects are automatically configured when used.

```
// control bits for CAN_MSG_OBJ.mode_id
#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

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;


void config_rxmsgobj (CAN_MSG_OBJ * msg_obj)
```

Example call:

```
// Configure message object 1 to receive all 11-bit messages 0x000-0x00F
msg_obj.msgobj  = 1;
msg_obj.mode_id = 0x000;
msg_obj.mask    = 0x7F0;
(*rom)->pCAND-> config_rxmsgobj(&msg_obj);
```

### 14.4.5 CAN receive

The CAN receive function allows reading messages that have been received by an Rx message object. A pointer to a message object structure is passed to the receive function. Before calling, the number of the message object that is to be read has to be set in the structure.

```
void config_rxmsgobj (CAN_MSG_OBJ * msg_obj)
```

Example call:

```
// Read out received message
msg_obj.msgobj = 5;
(*rom)->pCAND->can_receive(&msg_obj);
```

### 14.4.6 CAN transmit

The CAN transmit function allows setting up a message object and triggering the transmission of a CAN message on the bus. 11-bit standard and 29-bit extended messages are supported as well as both standard data and remote-transmit (RTR) messages.

```
void config_txmsgobj (CAN_MSG_OBJ * msg_obj)
```

Example call:

```
msg_obj.msgobj = 3;
msg_obj.mode_id = 0x123UL;
msg_obj.mask = 0x0UL;
msg_obj.dlc = 1;
msg_obj.data[0] = 0x00;
(*rom)->pCAND->can_transmit(&msg_obj);
```

### 14.4.7 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. One table contains all read-only, constant entries of four bytes or less. The second table contains all variable and writable entries as well as SDO segmented entries.

```
typedef struct _CAN_ODCONSTENTRY {
  uint16_t index;
  uint8_t  subindex;
  uint8_t  len;
  uint32_t val;
}  CAN_ODCONSTENTRY;

// upper-nibble values for CAN_ODENTRY.entrytype_len
#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

typedef struct _CAN_ODENTRY {
  uint16_t index;
  uint8_t subindex;
  uint8_t entrytype_len;
  unint8_t isr_handled;
  uint8_t *val;
} CAN_ODENTRY;

typedef struct _CAN_CANOPENCFG {
  uint8_t node_id;
  uint8_t msgobj_rx;
  uint8_t msgobj_tx;
  uint32_t od_const_num;
  CAN_ODCONSTENTRY *od_const_table;
  uint32_t od_num;
  CAN_ODENTRY *od_table;
} CAN_CANOPENCFG;
```

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);
```

### 14.4.8 CANopen handler

The CANopen handler processes the CANopen SDO messages to access the Object Dictionary and calls the CANopen callback functions when initialized. It can either be called by the interrupt handler automatically (isr_handled == TRUE in CANopen initialization structure) or manually via the CANopen handler API function. If called manually, the CANopen handler has to be called cyclically as often as needed for the application.

In a typical CANopen application, SDO handling has the lowest priority and is done in the foreground rather than through interrupt processing.

Example call:

```
// Call CANopen handler
(*rom)->pCAND->canopen_handler();
```

### 14.4.9 CAN/CANopen callback functions

The CAN API supports callback functions for various events. The callback functions are published via an API function.

```
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);
  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);
}  CAN_CALLBACKS;
```

Example callback table definition:

```
// List of callback function pointers
const CAN_CALLBACKS callbacks = {
  CAN_rx,
  CAN_tx,
  CAN_error,
  CANOPEN_sdo_exp_read,
  CANOPEN_sdo_exp_write,
  CANOPEN_sdo_seg_read,
  CANOPEN_sdo_seg_write,
  CANOPEN_sdo_req,
};
```

Example call:

```
// Publish callbacks
(*rom)->pCAND->config_calb((CAN_CALLBACKS *)&callbacks);
```

### 14.4.10 CAN message received callback

The CAN message received callback function is called on the interrupt level by the CAN interrupt handler.

Example call:

```
// CAN receive handler
void CAN_rx(uint8_t msgobj_num)
{
  // Read out received message
  msg_obj.msgobj = msgobj_num;
  (*rom)->pCAND->can_receive(&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.

### 14.4.11 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.

### 14.4.12 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;
}
```

### 14.4.13 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.

### 14.4.14 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.

### 14.4.15 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.

### 14.4.16 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.

### 14.4.17 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.

## 15.1 How to read this chapter

The 16-bit timer blocks are identical for all LPC111x and LPC11Cxx parts.

The Match0 output of timer 1 (CT16B1_MAT0) is not pinned out on parts LPC11C22 and LPC11C24.

## 15.2 Basic configuration

The CT16B0/1 are configured using the following registers:

1. Pins: The CT16B0/1 pins must be configured in the IOCONFIG register block (Section 7.4).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 7 and bit 8 (Table 20). The peripheral clock is provided by the system clock (see Table 19).

## 15.3 Features

- Two 16-bit counter/timers with a programmable 16-bit prescaler.
- Counter or timer operation.
- One 16-bit capture channel that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 16-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to three (CT16B0) or two (CT16B1) external outputs corresponding to match registers with the following capabilities:
  - Set LOW on match.
  - Set HIGH on match.
  - Toggle on match.
  - Do nothing on match.
- For each timer, up to four match registers can be configured as PWM allowing to use up to three match outputs as single edge controlled PWM outputs.

## 15.4 Applications

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

- Pulse Width Modulator via match outputs

## 15.5 Description

Each Counter/timer is designed to count cycles of the peripheral clock (PCLK) or an externally supplied clock and can optionally generate interrupts or perform other actions at specified timer values based on four match registers. The peripheral clock is provided by the system clock (see Figure 3). Each counter/timer also includes one capture input to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, three match registers on CT16B0 and two match registers on CT16B1 can be used to provide a single-edge controlled PWM output on the match output pins. It is recommended to use the match registers that are not pinned out to control the PWM cycle length.

**Remark:** The 16-bit counter/timer0 (CT16B0) and the 16-bit counter/timer1 (CT16B1) are functionally identical except for the peripheral base address.

## 15.6 Pin description

Table 213 gives a brief summary of each of the counter/timer related pins.

**Table 213. Counter/timer pin description**

| Pin | Type | Description |
|---|---|---|
| CT16B0_CAP0<br>CT16B1_CAP0 | Input | Capture Signal:<br>A transition on a capture pin can be configured to load the Capture Register with the value in the counter/timer and optionally generate an interrupt.<br>Counter/Timer block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 15.7.11. |
| CT16B0_MAT[2:0]<br>CT16B1_MAT[1:0] | Output | External Match Outputs of CT16B0/1:<br>When a match register of CT16B0/1 (MR3:0) equals the timer counter (TC), this output can either toggle, go LOW, go HIGH, or do nothing. The External Match Register (EMR) and the PWM Control Register (PWMCON) control the functionality of this output. |

## 15.7 Register description

The 16-bit counter/timer0 contains the registers shown in Table 214 and the 16-bit counter/timer1 contains the registers shown in Table 215. More detailed descriptions follow.

**Table 214. Register overview: 16-bit counter/timer 0 CT16B0 (base address 0x4000 C000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR16B0IR | R/W | 0x000 | Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending. | 0 |
| TMR16B0TCR | R/W | 0x004 | Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR. | 0 |
| TMR16B0TC | R/W | 0x008 | Timer Counter (TC). The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR. | 0 |
| TMR16B0PR | R/W | 0x00C | Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC. | 0 |
| TMR16B0PC | R/W | 0x010 | Prescale Counter (PC). The 16-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface. | 0 |
| TMR16B0MCR | R/W | 0x014 | Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs. | 0 |
| TMR16B0MR0 | R/W | 0x018 | Match Register 0 (MR0). MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC. | 0 |
| TMR16B0MR1 | R/W | 0x01C | Match Register 1 (MR1). See MR0 description. | 0 |
| TMR16B0MR2 | R/W | 0x020 | Match Register 2 (MR2). See MR0 description. | 0 |
| TMR16B0MR3 | R/W | 0x024 | Match Register 3 (MR3). See MR0 description. | 0 |
| TMR16B0CCR | R/W | 0x028 | Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place. | 0 |
| TMR16B0CR0 | RO | 0x02C | Capture Register 0 (CR0). CR0 is loaded with the value of TC when there is an event on the CT16B0_CAP0 input. | 0 |
| TMR16B0EMR | R/W | 0x03C | External Match Register (EMR). The EMR controls the match function and the external match pins CT16B0_MAT[2:0]. | 0 |
| - | - | 0x040 - 0x06C | reserved | - |
| TMR16B0CTCR | R/W | 0x070 | Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting. | 0 |
| TMR16B0PWMC | R/W | 0x074 | PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT16B0_MAT[2:0]. | 0 |

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

**Table 215. Register overview: 16-bit counter/timer 1 CT16B1 (base address 0x4001 0000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR16B1IR | R/W | 0x000 | Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending. | 0 |
| TMR16B1TCR | R/W | 0x004 | Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR. | 0 |
| TMR16B1TC | R/W | 0x008 | Timer Counter (TC). The 16-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR. | 0 |
| TMR16B1PR | R/W | 0x00C | Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC. | 0 |
| TMR16B1PC | R/W | 0x010 | Prescale Counter (PC). The 16-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface. | 0 |
| TMR16B1MCR | R/W | 0x014 | Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs. | 0 |
| TMR16B1MR0 | R/W | 0x018 | Match Register 0 (MR0). MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC. | 0 |
| TMR16B1MR1 | R/W | 0x01C | Match Register 1 (MR1). See MR0 description. | 0 |
| TMR16B1MR2 | R/W | 0x020 | Match Register 2 (MR2). See MR0 description. | 0 |
| TMR16B1MR3 | R/W | 0x024 | Match Register 3 (MR3). See MR0 description. | 0 |
| TMR16B1CCR | R/W | 0x028 | Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place. | 0 |
| TMR16B1CR0 | RO | 0x02C | Capture Register 0 (CR0). CR0 is loaded with the value of TC when there is an event on the CT16B1_CAP0 input. | 0 |
| TMR16B1EMR | R/W | 0x03C | External Match Register (EMR). The EMR controls the match function and the external match pins CT16B1_MAT[1:0]. | 0 |
| - | - | 0x040 - 0x06C | reserved | - |
| TMR16B1CTCR | R/W | 0x070 | Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting. | 0 |
| TMR16B1PWMC | R/W | 0x074 | PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT16B1_MAT[1:0]. | 0 |

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

### 15.7.1 Interrupt Register (TMR16B0IR and TMR16B1IR)

The Interrupt Register (IR) consists of four bits for the match interrupts and one bit for the capture interrupt. If an interrupt is generated then the corresponding bit in the IR will be HIGH. Otherwise, the bit will be LOW. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

**Table 216. Interrupt Register (TMR16B0IR - address 0x4000 C000 and TMR16B1IR - address 0x4001 0000) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | MR0 Interrupt | Interrupt flag for match channel 0. | 0 |
| 1 | MR1 Interrupt | Interrupt flag for match channel 1. | 0 |
| 2 | MR2 Interrupt | Interrupt flag for match channel 2. | 0 |
| 3 | MR3 Interrupt | Interrupt flag for match channel 3. | 0 |
| 4 | CR0 Interrupt | Interrupt flag for capture channel 0 event. | 0 |
| 31:5 | - | Reserved | - |

### 15.7.2 Timer Control Register (TMR16B0TCR and TMR16B1TCR)

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

**Table 217. Timer Control Register (TMR16B0TCR - address 0x4000 C004 and TMR16B1TCR - address 0x4001 0004) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | CEn | Counter Enable. When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled. | 0 |
| 1 | CRst | Counter Reset. When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero. | 0 |
| 31:2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 15.7.3 Timer Counter (TMR16B0TC - address 0x4000 C008 and TMR16B1TC - address 0x4001 0008)

The 16-bit Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC will count up through the value 0x0000 FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

**Table 218: Timer counter registers (TMR16B0TC, address 0x4000 C008 and TMR16B1TC 0x4001 0008) bit description**

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

### 15.7.4 Prescale Register (TMR16B0PR - address 0x4000 C00C and TMR16B1PR - address 0x4001 000C)

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

Table 219: Prescale registers (TMR16B0PR, address 0x4000 C00C and TMR16B1PR 0x4001 000C) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 15:0 | PR | Prescale max value. | 0 |
| 31:16 | - | Reserved. | - |

### 15.7.5 Prescale Counter register (TMR16B0PC - address 0x4000 C010 and TMR16B1PC - address 0x4001 0010)

The 16-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter. This allows control of the relationship between the resolution of the timer and the maximum time before the timer overflows. The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale Register, the Timer Counter is incremented, and the Prescale Counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, etc.

Table 220: Prescale counter registers (TMR16B0PC, address 0x4001 C010 and TMR16B1PC 0x4000 0010) bit description

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

### 15.7.6 Match Control Register (TMR16B0MCR and TMR16B1MCR)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter. The function of each of the bits is shown in Table 221.

Table 221. Match Control Register (TMR16B0MCR - address 0x4000 C014 and TMR16B1MCR - address 0x4001 0014) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | MR0I | | Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 1 | MR0R | | Reset on MR0: the TC will be reset if MR0 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 2 | MR0S | | Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 3 | MR1I | | Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |

**Table 221. Match Control Register (TMR16B0MCR - address 0x4000 C014 and TMR16B1MCR - address 0x4001 0014) bit description** *…continued*

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4 | MR1R | | Reset on MR1: the TC will be reset if MR1 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 5 | MR1S | | Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 6 | MR2I | | Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 7 | MR2R | | Reset on MR2: the TC will be reset if MR2 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 8 | MR2S | | Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 9 | MR3I | | Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 10 | MR3R | | Reset on MR3: the TC will be reset if MR3 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 11 | MR3S | | Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 31:12 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 15.7.7 Match Registers (TMR16B0MR0/1/2/3 - addresses 0x4000 C018/1C/20/24 and TMR16B1MR0/1/2/3 - addresses 0x4001 0018/1C/20/24)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

**Table 222: Match registers (TMR16B0MR0 to 3, addresses 0x4000 C018 to 24 and TMR16B1MR0 to 3, addresses 0x4001 0018 to 24) bit description**

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

## 15.7.8 Capture Control Register (TMR16B0CCR and TMR16B1CCR)

The Capture Control Register is used to control whether the Capture Register is loaded with the value in the Counter/timer when the capture event occurs, and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the description below, "n" represents the Timer number, 0 or 1.

**Table 223. Capture Control Register (TMR16B0CCR - address 0x4000 C028 and TMR16B1CCR - address 0x4001 0028) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | CAP0RE | | Capture on CT16Bn_CAP0 rising edge: a sequence of 0 then 1 on CT16Bn_CAP0 will cause CR0 to be loaded with the contents of TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 1 | CAP0FE | | Capture on CT16Bn_CAP0 falling edge: a sequence of 1 then 0 on CT16Bn_CAP0 will cause CR0 to be loaded with the contents of TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 2 | CAP0I | | Interrupt on CT16Bn_CAP0 event: a CR0 load due to a CT16Bn_CAP0 event will generate an interrupt. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 31:3 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 15.7.9 Capture Register (CT16B0CR0 - address 0x4000 C02C and CT16B1CR0 - address 0x4001 002C)

Each Capture register is associated with a device pin and may be loaded with the counter/timer value when a specified event occurs on that pin. The settings in the Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

**Table 224: Capture registers (TMR16B0CR0, address 0x4000 C02C and TMR16B1CR0, address 0x4001 002C) bit description**

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

### 15.7.10 External Match Register (TMR16B0EMR and TMR16B1EMR)

The External Match Register provides both control and status of the external match channels and external match pins CT16B0_MAT[2:0] and CT16B1_MAT[1:0].

If the match outputs are configured as PWM output in the PWMCON registers (Section 15.7.12), the function of the external match registers is determined by the PWM rules (Section 15.7.13 “Rules for single edge controlled PWM outputs” on page 259).

**Table 225. External Match Register (TMR16B0EMR - address 0x4000 C03C and TMR16B1EMR - address 0x4001 003C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | EM0 | | External Match 0. This bit reflects the state of output CT16B0_MAT0/CT16B1_MAT0, whether or not this output is connected to its pin. When a match occurs between the TC and MR0, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[5:4] control the functionality of this output. This bit is driven to the CT16B0_MAT0/CT16B1_MAT0 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 1 | EM1 | | External Match 1. This bit reflects the state of output CT16B0_MAT1/CT16B1_MAT1, whether or not this output is connected to its pin. When a match occurs between the TC and MR1, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[7:6] control the functionality of this output. This bit is driven to the CT16B0_MAT1/CT16B1_MAT1 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 2 | EM2 | | External Match 2. This bit reflects the state of output match channel 2, whether or not this output is connected to its pin. When a match occurs between the TC and MR2, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[9:8] control the functionality of this output. Note that on counter/timer 0 this match channel is not pinned out. This bit is driven to the CT16B1_MAT2 pin if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 3 | EM3 | | External Match 3. This bit reflects the state of output of match channel 3. When a match occurs between the TC and MR3, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[11:10] control the functionality of this output. There is no output pin available for this channel on either of the 16-bit timers. | 0 |
| 5:4 | EMC0 | | External Match Control 0. Determines the functionality of External Match 0. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT16Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 7:6 | EMC1 | | External Match Control 1. Determines the functionality of External Match 1. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT16Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |

**Table 225. External Match Register (TMR16B0EMR - address 0x4000 C03C and TMR16B1EMR - address 0x4001 003C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 9:8 | EMC2 | | External Match Control 2. Determines the functionality of External Match 2. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT16Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 11:10 | EMC3 | | External Match Control 3. Determines the functionality of External Match 3. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT16Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 31:12 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

**Table 226. External match control**

| EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4] | Function |
|---|---|
| 00 | Do Nothing. |
| 01 | Clear the corresponding External Match bit/output to 0 (CT16Bn_MATm pin is LOW if pinned out). |
| 10 | Set the corresponding External Match bit/output to 1 (CT16Bn_MATm pin is HIGH if pinned out). |
| 11 | Toggle the corresponding External Match bit/output. |

### 15.7.11 Count Control Register (TMR16B0CTCR and TMR16B1CTCR)

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

When Counter Mode is chosen as a mode of operation, the CAP input (selected by the CTCR bits 3:2) is sampled on every rising edge of the PCLK clock. After comparing two consecutive samples of this CAP input, one of the following four events is recognized: rising edge, falling edge, either of edges or no changes in the level of the selected CAP input. Only if the identified event occurs, and the event corresponds to the one selected by bits 1:0 in the CTCR register, will the Timer Counter register be incremented.

Effective processing of the externally supplied clock to the counter has some limitations. Since two successive rising edges of the PCLK clock are used to identify only one edge on the CAP selected input, the frequency of the CAP input can not exceed one half of the PCLK clock. Consequently, duration of the HIGH/LOW levels on the same CAP input in this case can not be shorter than $1/(2 \times \text{PCLK})$.

**Table 227. Count Control Register (TMR16B0CTCR - address 0x4000 C070 and TMR16B1CTCR - address 0x4001 0070) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | CTM | | Counter/Timer Mode. This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC). | 00 |
| | | 0x0 | Timer Mode: every rising PCLK edge | |
| | | 0x1 | Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2. | |
| | | 0x2 | Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2. | |
| | | 0x3 | Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2. | |
| 3:2 | CIS | | Count Input Select. In counter mode (when bits 1:0 in this register are not 00), these bits select which CAP pin is sampled for clocking. **Note:** If Counter mode is selected in the CTCR register, bits 2:0 in the Capture Control Register (CCR) must be programmed as 000. | 00 |
| | | 0x0 | CT16Bn_CAP0 | |
| | | 0x1 | Reserved. | |
| | | 0x2 | Reserved. | |
| | | 0x0 | Reserved. | |
| 31:4 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 15.7.12 PWM Control register (TMR16B0PWMC and TMR16B1PWMC)

The PWM Control Register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the External Match Register (EMR).

For timer 0, three single-edge controlled PWM outputs can be selected on the CT16B0_MAT[2:0] outputs. For timer 1, two single-edged PWM outputs can be selected on the CT16B1_Mat[1:0] outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

**Table 228. PWM Control Register (TMR16B0PWMC - address 0x4000 C074 and TMR16B1PWMC- address 0x4001 0074) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | PWMEN0 | | PWM channel0 enable | 0 |
| | | 0 | CT16Bn_MAT0 is controlled by EM0. | |
| | | 1 | PWM mode is enabled for CT16Bn_MAT0. | |

**Table 228. PWM Control Register (TMR16B0PWMC - address 0x4000 C074 and TMR16B1PWMC- address 0x4001 0074) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1 | PWMEN1 | | PWM channel1 enable | 0 |
| | | 0 | CT16Bn_MAT1 is controlled by EM1. | |
| | | 1 | PWM mode is enabled for CT16Bn_MAT1. | |
| 2 | PWMEN2 | | PWM channel2 enable | 0 |
| | | 0 | Match channel 2 or pin CT16B0_MAT2 is controlled by EM2. Match channel 2 is not pinned out on timer 1. | |
| | | 1 | PWM mode is enabled for match channel 2 or pin CT16B0_MAT2. | |
| 3 | PWMEN3 | | PWM channel3 enable<br>**Note:** It is recommended to use match channel 3 to set the PWM cycle because it is not pinned out. | 0 |
| | | 0 | Match channel 3 match channel 3 is controlled by EM3. | |
| | | 1 | PWM mode is enabled for match channel 3match channel 3. | |
| 31:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 15.7.13 Rules for single edge controlled PWM outputs

1. All single edge controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
2. Each PWM output will go HIGH when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
3. If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal will be cleared on the next start of the next PWM cycle.
4. If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output will be reset to LOW on the next clock tick. Therefore, the PWM output will always consist of a one clock tick wide positive pulse with a period determined by the PWM cycle length (i.e. the timer reload value).
5. If a match register is set to zero, then the PWM output will go to HIGH the first time the timer goes back to zero and will stay HIGH continuously.

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

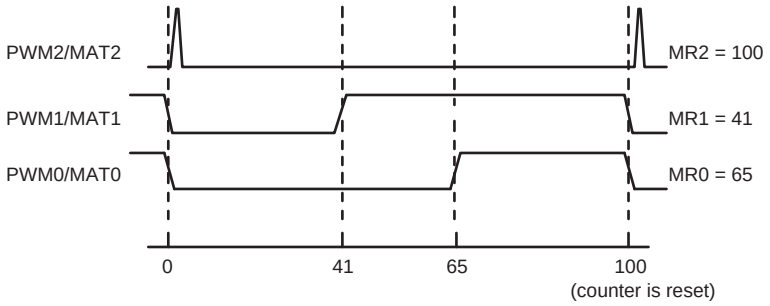



**Fig 53. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.**

## 15.8 Example timer operation

Figure 54 shows a timer configured to reset the count and generate an interrupt on match. The prescaler is set to 2 and the match register set to 6. At the end of the timer cycle where the match occurs, the timer count is reset. This gives a full length cycle to the match value. The interrupt indicating that a match occurred is generated in the next clock after the timer reached the match value.

Figure 55 shows a timer configured to stop and generate an interrupt on match. The prescaler is again set to 2 and the match register set to 6. In the next clock after the timer reaches the match value, the timer enable bit in TCR is cleared, and the interrupt indicating that a match occurred is generated.

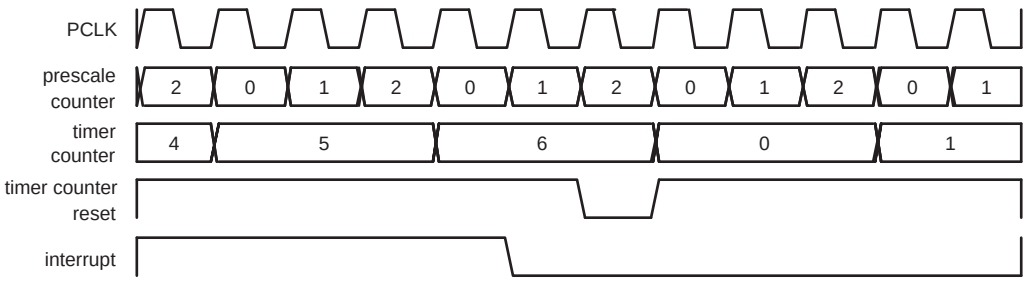



**Fig 54. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled**

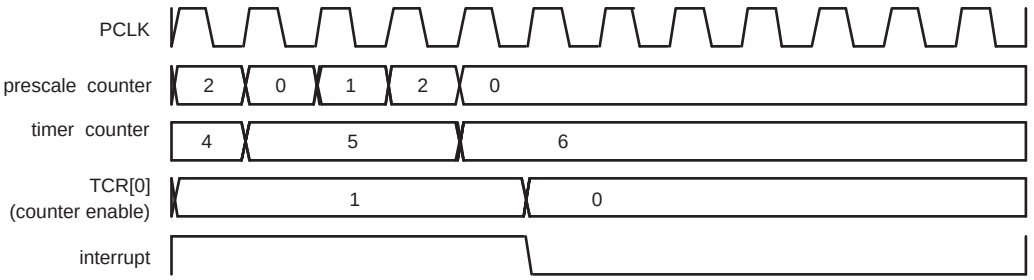



**Fig 55. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled**

## 15.9 Architecture

The block diagram for counter/timer0 and counter/timer1 is shown in Figure 56.

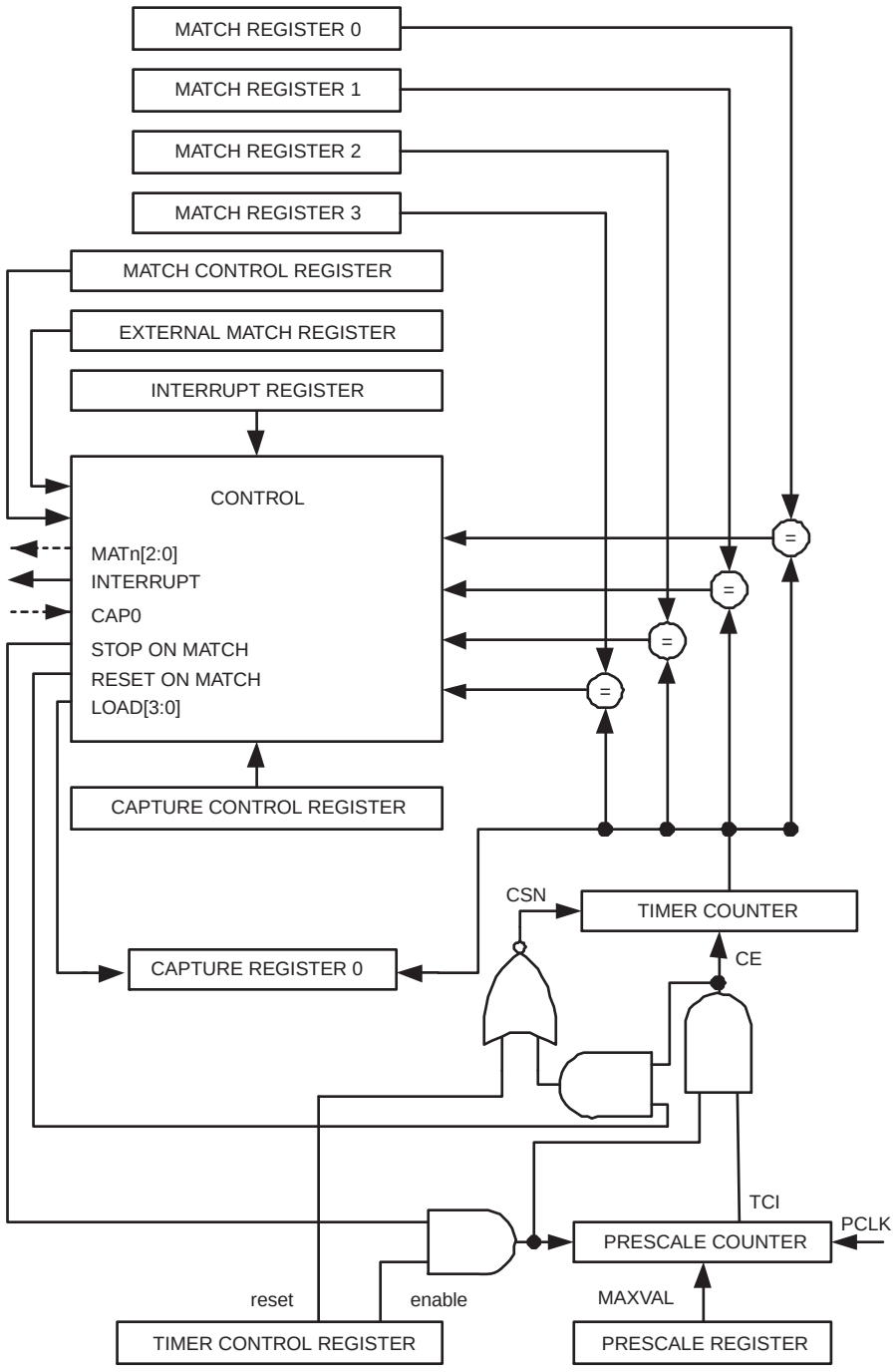



**Fig 56. 16-bit counter/timer block diagram**

## 16.1 How to read this chapter

The 32-bit timer blocks are identical for all LPC111x and LPC11Cxx parts.

## 16.2 Basic configuration

The CT32B0/1 are configured using the following registers:

1. Pins: The CT32B0/1 pins must be configured in the IOCONFIG register block (Section 7.4).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 9 and bit 10 (Table 20). The peripheral clock (PCLK) is provided by the system clock (see Table 19).

## 16.3 Features

- Two 32-bit counter/timers with a programmable 32-bit prescaler.
- Counter or Timer operation.
- One 32-bit capture channel that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Four external outputs corresponding to match registers with the following capabilities:
  - Set LOW on match.
  - Set HIGH on match.
  - Toggle on match.
  - Do nothing on match.
- For each timer, up to four match registers can be configured as PWM allowing to use up to three match outputs as single edge controlled PWM outputs.

## 16.4 Applications

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

## 16.5 Description

Each Counter/timer is designed to count cycles of the peripheral clock (PCLK) or an externally supplied clock and can optionally generate interrupts or perform other actions at specified timer values based on four match registers. The peripheral clock is provided by the system clock (see Figure 3). Each counter/timer also includes one capture input to trap the timer value when an input signal transitions, optionally generating an interrupt.

In PWM mode, three match registers can be used to provide a single-edge controlled PWM output on the match output pins. One match register is used to control the PWM cycle length.

**Remark:** 32-bit counter/timer0 (CT32B0) and 32-bit counter/timer1 (CT32B1) are functionally identical except for the peripheral base address.

## 16.6 Pin description

Table 229 gives a brief summary of each of the counter/timer related pins.

**Table 229. Counter/timer pin description**

| Pin | Type | Description |
|---|---|---|
| CT32B0_CAP0<br>CT32B1_CAP0 | Input | Capture Signals:<br>A transition on a capture pin can be configured to load one of the Capture Registers with the value in the Timer Counter and optionally generate an interrupt.<br>The counter/timer block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 16.7.11 “Count Control Register (TMR32B0CTCR and TMR32B1TCR)” on page 271. |
| CT32B0_MAT[3:0]<br>CT32B1_MAT[3:0] | Output | External Match Output of CT32B0/1:<br>When a match register TMR32B0/1MR3:0 equals the timer counter (TC), this output can either toggle, go LOW, go HIGH, or do nothing. The External Match Register (EMR) and the PWM Control register (PWMCON) control the functionality of this output. |

## 16.7 Register description

32-bit counter/timer0 contains the registers shown in Table 230 and 32-bit counter/timer1 contains the registers shown in Table 231. More detailed descriptions follow.

**Table 230. Register overview: 32-bit counter/timer 0 CT32B0 (base address 0x4001 4000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR32B0IR | R/W | 0x000 | Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending. | 0 |
| TMR32B0TCR | R/W | 0x004 | Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR. | 0 |
| TMR32B0TC | R/W | 0x008 | Timer Counter (TC). The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR. | 0 |
| TMR32B0PR | R/W | 0x00C | Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC. | 0 |

**Table 230. Register overview: 32-bit counter/timer 0 CT32B0 (base address 0x4001 4000)** *…continued*

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR32B0PC | R/W | 0x010 | Prescale Counter (PC). The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface. | 0 |
| TMR32B0MCR | R/W | 0x014 | Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs. | 0 |
| TMR32B0MR0 | R/W | 0x018 | Match Register 0 (MR0). MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC. | 0 |
| TMR32B0MR1 | R/W | 0x01C | Match Register 1 (MR1). See MR0 description. | 0 |
| TMR32B0MR2 | R/W | 0x020 | Match Register 2 (MR2). See MR0 description. | 0 |
| TMR32B0MR3 | R/W | 0x024 | Match Register 3 (MR3). See MR0 description. | 0 |
| TMR32B0CCR | R/W | 0x028 | Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place. | 0 |
| TMR32B0CR0 | RO | 0x02C | Capture Register 0 (CR0). CR0 is loaded with the value of TC when there is an event on the CT32B0_CAP0 input. | 0 |
| TMR32B0EMR | R/W | 0x03C | External Match Register (EMR). The EMR controls the match function and the external match pins CT32B0_MAT[3:0]. | 0 |
| - | - | 0x040 - 0x06C | reserved | - |
| TMR32B0CTCR | R/W | 0x070 | Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting. | 0 |
| TMR32B0PWMC | R/W | 0x074 | PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT32B0_MAT[3:0]. | 0 |

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

**Table 231. Register overview: 32-bit counter/timer 1 CT32B1 (base address 0x4001 8000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR32B1IR | R/W | 0x000 | Interrupt Register (IR). The IR can be written to clear interrupts. The IR can be read to identify which of five possible interrupt sources are pending. | 0 |
| TMR32B1TCR | R/W | 0x004 | Timer Control Register (TCR). The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR. | 0 |
| TMR32B1TC | R/W | 0x008 | Timer Counter (TC). The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR. | 0 |
| TMR32B1PR | R/W | 0x00C | Prescale Register (PR). When the Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC. | 0 |
| TMR32B1PC | R/W | 0x010 | Prescale Counter (PC). The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface. | 0 |

**Table 231. Register overview: 32-bit counter/timer 1 CT32B1 (base address 0x4001 8000)** *…continued*

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| TMR32B1MCR | R/W | 0x014 | Match Control Register (MCR). The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs. | 0 |
| TMR32B1MR0 | R/W | 0x018 | Match Register 0 (MR0). MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC. | 0 |
| TMR32B1MR1 | R/W | 0x01C | Match Register 1 (MR1). See MR0 description. | 0 |
| TMR32B1MR2 | R/W | 0x020 | Match Register 2 (MR2). See MR0 description. | 0 |
| TMR32B1MR3 | R/W | 0x024 | Match Register 3 (MR3). See MR0 description. | 0 |
| TMR32B1CCR | R/W | 0x028 | Capture Control Register (CCR). The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place. | 0 |
| TMR32B1CR0 | RO | 0x02C | Capture Register 0 (CR0). CR0 is loaded with the value of TC when there is an event on the CT32B1_CAP0 input. | 0 |
| TMR32B1EMR | R/W | 0x03C | External Match Register (EMR). The EMR controls the match function and the external match pins CT32B1_MAT[3:0]. | 0 |
| - | - | 0x040 - 0x06C | reserved | - |
| TMR32B1CTCR | R/W | 0x070 | Count Control Register (CTCR). The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting. | 0 |
| TMR32B1PWMC | R/W | 0x074 | PWM Control Register (PWMCON). The PWMCON enables PWM mode for the external match pins CT32B1_MAT[3:0]. | 0 |

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

### 16.7.1 Interrupt Register (TMR32B0IR and TMR32B1IR)

The Interrupt Register consists of four bits for the match interrupts and one bit for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be HIGH. Otherwise, the bit will be LOW. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

**Table 232: Interrupt Register (TMR32B0IR - address 0x4001 4000 and TMR32B1IR - address 0x4001 8000) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | MR0 Interrupt | Interrupt flag for match channel 0. | 0 |
| 1 | MR1 Interrupt | Interrupt flag for match channel 1. | 0 |
| 2 | MR2 Interrupt | Interrupt flag for match channel 2. | 0 |
| 3 | MR3 Interrupt | Interrupt flag for match channel 3. | 0 |
| 4 | CR0 Interrupt | Interrupt flag for capture channel 0 event. | 0 |
| 31:5 | - | Reserved | - |

### 16.7.2 Timer Control Register (TMR32B0TCR and TMR32B1TCR)

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

Table 233: Timer Control Register (TMR32B0TCR - address 0x4001 4004 and TMR32B1TCR - address 0x4001 8004) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | CEn | When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled. | 0 |
| 1 | CRst | When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero. | 0 |
| 31:2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 16.7.3 Timer Counter (TMR32B0TC - address 0x4001 4008 and TMR32B1TC - address 0x4001 8008)

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

Table 234: Timer counter registers (TMR32B0TC, address 0x4001 4008 and TMR32B1TC 0x4001 8008) bit description

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

### 16.7.4 Prescale Register (TMR32B0PR - address 0x4001 400C and TMR32B1PR - address 0x4001 800C)

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

Table 235: Prescale registers (TMR32B0PR, address 0x4001 400C and TMR32B1PR 0x4001 800C) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | PR | Prescale value. | 0 |

### 16.7.5 Prescale Counter Register (TMR32B0PC - address 0x4001 4010 and TMR32B1PC - address 0x4001 8010)

The 32-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter. This allows control of the relationship between the resolution of the timer and the maximum time before the timer overflows. The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale Register, the Timer Counter is incremented, and the Prescale Counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, etc.

**Table 236: Prescale counter registers (TMR32B0PC, address 0x4001 4010 and TMR32B1PC 0x4001 8010) bit description**

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

### 16.7.6 Match Control Register (TMR32B0MCR and TMR32B1MCR)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter. The function of each of the bits is shown in Table 237.

**Table 237: Match Control Register (TMR32B0MCR - address 0x4001 4014 and TMR32B1MCR - address 0x4001 8014) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | MR0I | | Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 1 | MR0R | | Reset on MR0: the TC will be reset if MR0 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 2 | MR0S | | Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 3 | MR1I | | Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 4 | MR1R | | Reset on MR1: the TC will be reset if MR1 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 5 | MR1S | | Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 6 | MR2I | | Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 7 | MR2R | | Reset on MR2: the TC will be reset if MR2 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 8 | MR2S | | Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |

**Table 237: Match Control Register (TMR32B0MCR - address 0x4001 4014 and TMR32B1MCR - address 0x4001 8014) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 9 | MR3I | | Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 10 | MR3R | | Reset on MR3: the TC will be reset if MR3 matches it. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 11 | MR3S | | Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 31:12 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 16.7.7 Match Registers (TMR32B0MR0/1/2/3 - addresses 0x4001 4018/1C/20/24 and TMR32B1MR0/1/2/3 addresses 0x4001 8018/1C/20/24)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

**Table 238: Match registers (TMR32B0MR0 to 3, addresses 0x4001 4018 to 24 and TMR32B1MR0 to 3, addresses 0x4001 8018 to 24) bit description**

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

### 16.7.8 Capture Control Register (TMR32B0CCR and TMR32B1CCR)

The Capture Control Register is used to control whether the Capture Register is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the description below, “n” represents the Timer number, 0 or 1.

**Table 239: Capture Control Register (TMR32B0CCR - address 0x4001 4028 and TMR32B1CCR - address 0x4001 8028) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | CAP0RE | | Capture on CT32Bn_CAP0 rising edge: a sequence of 0 then 1 on CT32Bn_CAP0 will cause CR0 to be loaded with the contents of TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |

**Table 239: Capture Control Register (TMR32B0CCR - address 0x4001 4028 and TMR32B1CCR - address 0x4001 8028) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1 | CAP0FE | | Capture on CT32Bn_CAP0 falling edge: a sequence of 1 then 0 on CT32Bn_CAP0 will cause CR0 to be loaded with the contents of TC. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 2 | CAP0I | | Interrupt on CT32Bn_CAP0 event: a CR0 load due to a CT32Bn_CAP0 event will generate an interrupt. | 0 |
| | | 1 | Enabled | |
| | | 0 | Disabled | |
| 31:3 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 16.7.9 Capture Register (TMR32B0CR0 - address 0x4001 402C and TMR32B1CR0 - address 0x4001 802C)

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin. The settings in the Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

**Table 240: Capture registers (TMR32B0CR0, addresses 0x4001 402C and TMR32B1CR0, addresses 0x4001 802C) bit description**

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

### 16.7.10 External Match Register (TMR32B0EMR and TMR32B1EMR)

The External Match Register provides both control and status of the external match pins CAP32Bn_MAT[3:0].

If the match outputs are configured as PWM output, the function of the external match registers is determined by the PWM rules (Section 16.7.13 “Rules for single edge controlled PWM outputs” on page 273).

**Table 241: External Match Register (TMR32B0EMR - address 0x4001 403C and TMR32B1EMR - address0x4001 803C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | EM0 | | External Match 0. This bit reflects the state of output CT32Bn_MAT0, whether or not this output is connected to its pin. When a match occurs between the TC and MR0, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[5:4] control the functionality of this output. This bit is driven to the CT32B0_MAT0/CT16B1_MAT0 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 1 | EM1 | | External Match 1. This bit reflects the state of output CT32Bn_MAT1, whether or not this output is connected to its pin. When a match occurs between the TC and MR1, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[7:6] control the functionality of this output. This bit is driven to the CT32B0_MAT1/CT16B1_MAT1 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 2 | EM2 | | External Match 2. This bit reflects the state of output CT32Bn_MAT2, whether or not this output is connected to its pin. When a match occurs between the TC and MR2, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[9:8] control the functionality of this output. This bit is driven to the CT32B0_MAT2/CT16B1_MAT2 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 3 | EM3 | | External Match 3. This bit reflects the state of output CT32Bn_MAT3, whether or not this output is connected to its pin. When a match occurs between the TC and MR3, this bit can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[11:10] control the functionality of this output. This bit is driven to the CT32B0_MAT3/CT16B1_MAT3 pins if the match function is selected in the IOCON registers (0 = LOW, 1 = HIGH). | 0 |
| 5:4 | EMC0 | | External Match Control 0. Determines the functionality of External Match 0. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT32Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 7:6 | EMC1 | | External Match Control 1. Determines the functionality of External Match 1. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT32Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 9:8 | EMC2 | | External Match Control 2. Determines the functionality of External Match 2. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT32Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |

**Table 241: External Match Register (TMR32B0EMR - address 0x4001 403C and TMR32B1EMR - address0x4001 803C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 11:10 | EMC3 | | External Match Control 3. Determines the functionality of External Match 3. | 00 |
| | | 0x0 | Do Nothing. | |
| | | 0x1 | Clear the corresponding External Match bit/output to 0 (CT32Bn_MATm pin is LOW if pinned out). | |
| | | 0x2 | Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out). | |
| | | 0x3 | Toggle the corresponding External Match bit/output. | |
| 31:12 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

**Table 242. External match control**

| EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4] | Function |
|---|---|
| 00 | Do Nothing. |
| 01 | Clear the corresponding External Match bit/output to 0 (CT32Bn_MATm pin is LOW if pinned out). |
| 10 | Set the corresponding External Match bit/output to 1 (CT32Bn_MATm pin is HIGH if pinned out). |
| 11 | Toggle the corresponding External Match bit/output. |

### 16.7.11 Count Control Register (TMR32B0CTCR and TMR32B1TCR)

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

When Counter Mode is chosen as a mode of operation, the CAP input (selected by the CTCR bits 3:2) is sampled on every rising edge of the PCLK clock. After comparing two consecutive samples of this CAP input, one of the following four events is recognized: rising edge, falling edge, either of edges or no changes in the level of the selected CAP input. Only if the identified event occurs, and the event corresponds to the one selected by bits 1:0 in the CTCR register, will the Timer Counter register be incremented.

Effective processing of the externally supplied clock to the counter has some limitations. Since two successive rising edges of the PCLK clock are used to identify only one edge on the CAP selected input, the frequency of the CAP input can not exceed one half of the PCLK clock. Consequently, duration of the HIGH/LOW levels on the same CAP input in this case can not be shorter than 1/(2 $\times$ PCLK).

**Table 243: Count Control Register (TMR32B0CTCR - address 0x4001 4070 and TMR32B1TCR - address 0x4001 8070) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | CTM | | Counter/Timer Mode. This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC).<br>Timer Mode: every rising PCLK edge | 00 |
| | | 0x0 | Timer Mode: every rising PCLK edge | |
| | | 0x1 | Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2. | |
| | | 0x2 | Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2. | |
| | | 0x3 | Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2. | |
| 3:2 | CIS | | Count Input Select. When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking: | 00 |
| | | 0x0 | CT32Bn_CAP0 | |
| | | 0x1 | Reserved | |
| | | 0x2 | Reserved | |
| | | 0x3 | Reserved<br>**Note:** If Counter mode is selected in the TnCTCR, the 3 bits for that input in the Capture Control Register (TnCCR) must be programmed as 000. | |
| 31:4 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 16.7.12 PWM Control Register (TMR32B0PWMC and TMR32B1PWMC)

The PWM Control Register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the External Match Register (EMR).

For each timer, a maximum of three-single edge controlled PWM outputs can be selected on the MATn[2:0] outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

**Table 244: PWM Control Register (TMR32B0PWMC - 0x4001 4074 and TMR32B1PWMC - 0x4001 8074) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | PWMEN0 | | PWM channel 0 enable | 0 |
| | | 0 | CT32Bn_MAT0 is controlled by EM0. | |
| | | 1 | PWM mode is enabled for CT32Bn_MAT0. | |

**Table 244: PWM Control Register (TMR32B0PWMC - 0x4001 4074 and TMR32B1PWMC - 0x4001 8074) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1 | PWMEN1 | | PWM channel 1 enable | 0 |
| | | 0 | CT32Bn_MAT1 is controlled by EM1. | |
| | | 1 | PWM mode is enabled for CT32Bn_MAT1. | |
| 2 | PWMEN2 | | PWM channel 2 enable | 0 |
| | | 0 | CT32Bn_MAT2 is controlled by EM2. | |
| | | 1 | PWM mode is enabled for CT32Bn_MAT2. | |
| 3 | PWMEN3 | | PWM channel 3 enable<br>**Note:** It is recommended to use match channel 3 to set the PWM cycle. | 0 |
| | | 0 | CT32Bn_MAT3 is controlled by EM3. | |
| | | 1 | PWM mode is enabled for CT32Bn_MAT3. | |
| 31:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 16.7.13 Rules for single edge controlled PWM outputs

1. All single edge controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
2. Each PWM output will go HIGH when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
3. If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal will be cleared with the start of the next PWM cycle.
4. If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output will be reset to LOW on the next clock tick after the timer reaches the match value. Therefore, the PWM output will always consist of a one clock tick wide positive pulse with a period determined by the PWM cycle length (i.e. the timer reload value).
5. If a match register is set to zero, then the PWM output will go to HIGH the first time the timer goes back to zero and will stay HIGH continuously.

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

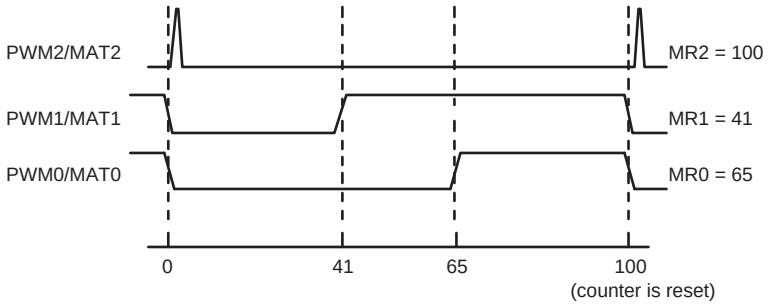



**Fig 57. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by the PWCON register.**

## 16.8 Example timer operation

Figure 58 shows a timer configured to reset the count and generate an interrupt on match. The prescaler is set to 2 and the match register set to 6. At the end of the timer cycle where the match occurs, the timer count is reset. This gives a full length cycle to the match value. The interrupt indicating that a match occurred is generated in the next clock after the timer reached the match value.

Figure 59 shows a timer configured to stop and generate an interrupt on match. The prescaler is again set to 2 and the match register set to 6. In the next clock after the timer reaches the match value, the timer enable bit in TCR is cleared, and the interrupt indicating that a match occurred is generated.

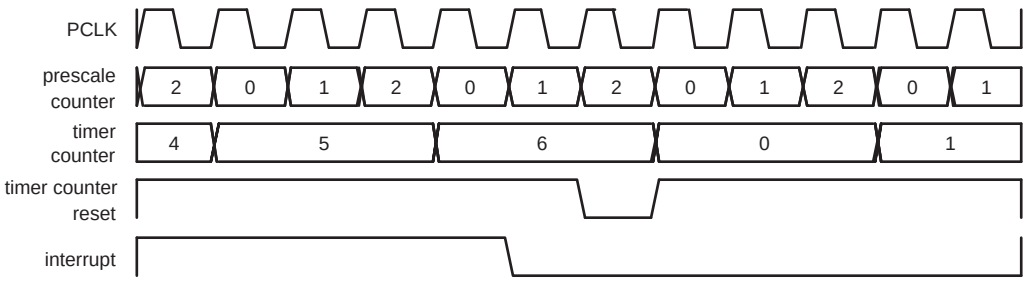



**Fig 58. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled**

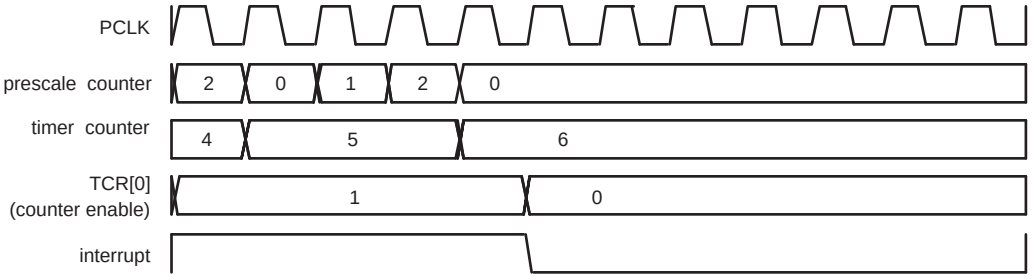



**Fig 59. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled**

## 16.9 Architecture

The block diagram for 32-bit counter/timer0 and 32-bit counter/timer1 is shown in Figure 60.

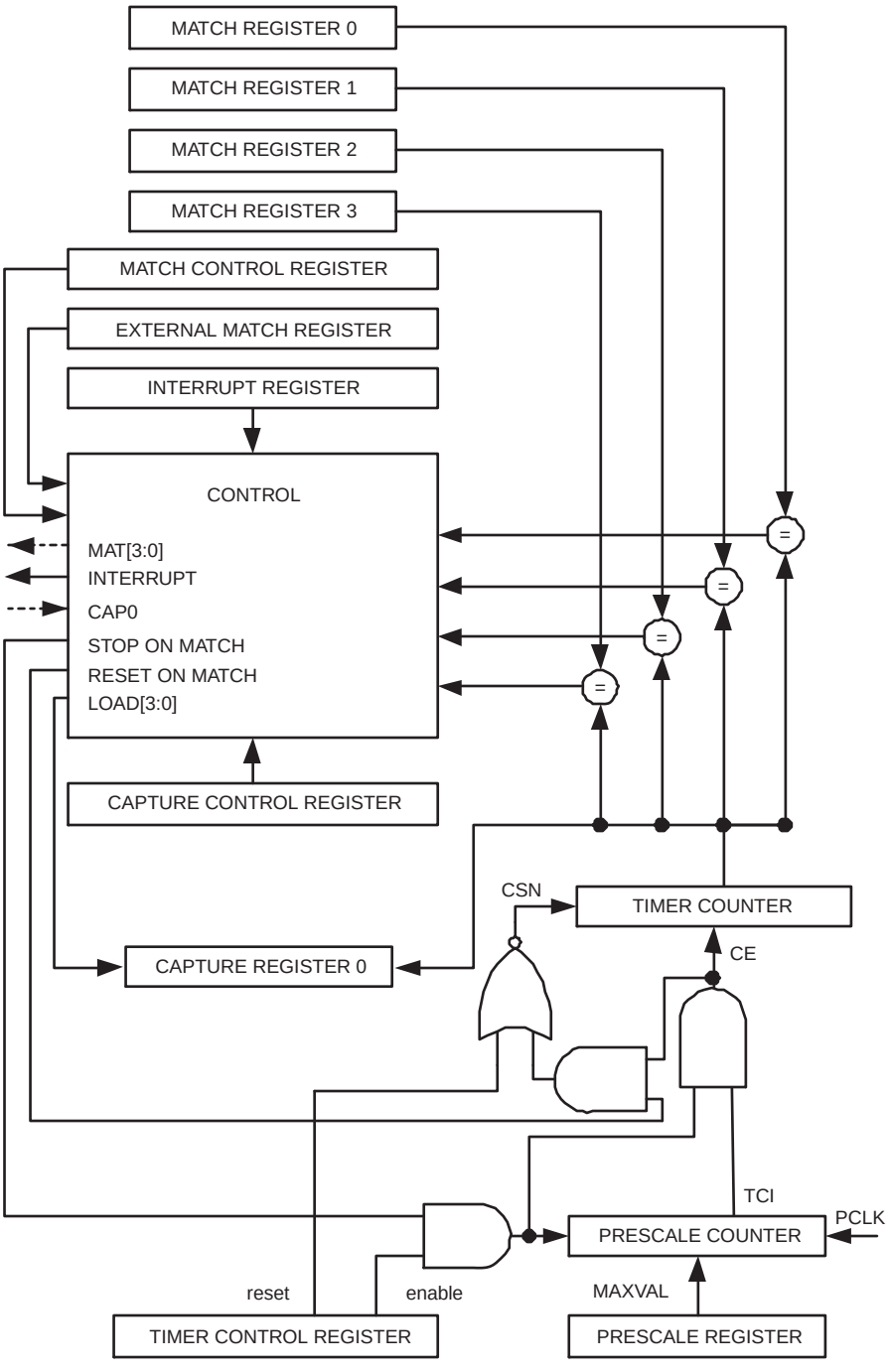



**Fig 60. 32-bit counter/timer block diagram**

## 17.1 How to read this chapter

The WDT block is identical for all LPC111x and LPC11Cxx parts.

## 17.2 Basic configuration

The WDT is configured using the following registers:

1. Pins: The WDT uses no external pins.
2. Power: In the SYSAHBCLKCTRL register, set bit 15 (Table 20).
3. Peripheral clock: Select the watchdog clock source Table 24) and enable the WDT peripheral clock by writing to the WDTCLKDIV register (Table 26).

   **Remark:** The frequency of the watchdog oscillator is undefined after reset. The watchdog oscillator frequency must be programmed by writing to the WDTOSCCTRL register (see Table 12) before using the watchdog oscillator for the WDT.

## 17.3 Features

- Internally resets chip if not periodically reloaded.
- Debug mode.
- Enabled by software but requires a hardware reset or a Watchdog reset/interrupt to be disabled.
- Incorrect/Incomplete feed sequence causes reset/interrupt if enabled.
- Flag to indicate Watchdog reset.
- Programmable 24 bit timer with internal pre-scaler.
- Selectable time period from ($T_{WDCLK} \times 256 \times 4$) to ($T_{WDCLK} \times 2^{24} \times 4$) in multiples of $T_{WDCLK} \times 4$.
- The Watchdog clock (WDCLK) source is selected in the syscon block from the Internal RC oscillator (IRC), the main clock, or the Watchdog oscillator, see Table 24. This gives a wide range of potential timing choices for Watchdog operation under different power reduction conditions. For increased reliability, it also provides the ability to run the Watchdog timer from an entirely internal source that is not dependent on an external crystal and its associated components and wiring.

## 17.4 Applications

The purpose of the Watchdog is to reset the microcontroller within a reasonable amount of time if it enters an erroneous state. When enabled, the Watchdog will generate a system reset if the user program fails to feed (or reload) the Watchdog within a predetermined amount of time.

## 17.5 Description

The Watchdog consists of a divide by 4 fixed pre-scaler and a 24-bit counter. The clock is fed to the timer via a pre-scaler. The timer 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:

1. Set the Watchdog timer constant reload value in WDTC register.
2. Setup the Watchdog timer operating mode in WDMOD register.
3. Enable the Watchdog by writing 0xAA followed by 0x55 to the WDFEED register.
4. The Watchdog should be fed again before the Watchdog counter underflows to prevent reset/interrupt.

When the Watchdog is in the reset mode and the counter underflows, the CPU will be reset, loading the stack pointer and program counter from the vector table as in the case of external reset. The Watchdog time-out flag (WDTOF) can be examined to determine if the Watchdog has caused the reset condition. The WDTOF flag must be cleared by software.

## 17.6 WDT clocking

The watchdog timer block uses two clocks: PCLK and WDCLK. PCLK is used for the APB accesses to the watchdog registers and is derived from the system clock (see Figure 3). The WDCLK is used for the watchdog timer counting and is derived from the wdt_clk in Figure 3. Several clocks can be used as a clock source for wdt_clk clock: the IRC, the watchdog oscillator, and the main clock. The clock source is selected in the syscon block (see Table 24). The WDCLK has its own clock divider (Section 3.5.20), which can also disable this clock.

There is some synchronization logic between these two clock domains. When the WDMOD and WDTC registers are updated by APB operations, the new value will take effect 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 the PCLK for reading as the WDTV register by the CPU.

**Remark:** The frequency of the watchdog oscillator is undefined after reset. The watchdog oscillator frequency must be programmed by writing to the WDTOSCCTRL register (see Table 12) before using the watchdog oscillator for the WDT.

## 17.7 Register description

The Watchdog contains four registers as shown in Table 245 below.

**Table 245. Register overview: Watchdog timer (base address 0x4000 4000)**

| Name | Access | Address offset | Description | Reset Value[1] |
|---|---|---|---|---|
| WDMOD | R/W | 0x000 | Watchdog mode register. This register contains the basic mode and status of the Watchdog Timer. | 0 |
| WDTC | R/W | 0x004 | Watchdog timer constant register. This register determines the time-out value. | 0xFF |
| WDFEED | WO | 0x008 | Watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer with the value contained in WDTC. | NA |
| WDTV | RO | 0x00C | Watchdog timer value register. This register reads out the current value of the Watchdog timer. | 0xFF |

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

### 17.7.1 Watchdog Mode register (WDMOD - 0x4000 0000)

The WDMOD register controls the operation of the Watchdog through the combination of WDEN and RESET bits. Note that a watchdog feed must be performed before any changes to the WDMOD register take effect.

**Table 246. Watchdog Mode register (WDMOD - address 0x4000 4000) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | WDEN | WDEN Watchdog enable bit (Set Only). When 1, the watchdog timer is running. | 0 |
| 1 | WDRESET | WDRESET Watchdog reset enable bit (Set Only). When 1, a watchdog time-out will cause a chip reset. | 0 |
| 2 | WDTOF | WDTOF Watchdog time-out flag. Set when the watchdog timer times out, cleared by software. | 0 (Only after POR and BOD reset) |
| 3 | WDINT | WDINT Watchdog interrupt flag (Read Only, not clearable by software). | 0 |
| 7:4 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 31:8 | - | reserved | - |

Once the **WDEN** and/or **WDRESET** bits are set, they can not be cleared by software. Both flags are cleared by a reset or a Watchdog timer underflow.

**WDTOF** The Watchdog time-out flag is set when the Watchdog times out. This flag is cleared by software or a POR or Brown-Out-Detect reset.

**WDINT** The Watchdog interrupt flag is set when the Watchdog times out. This flag is cleared when any reset occurs. Once the watchdog interrupt is serviced, it can be disabled in the NVIC or the watchdog interrupt request will be generated indefinitely. The intent of the watchdog interrupt is to allow debugging watchdog activity without resetting the device when the watchdog overflows.

Watchdog reset or interrupt will occur any time the watchdog is running and has an operating clock source. Any clock source works in Sleep mode, and if a watchdog interrupt occurs in Sleep mode, it will wake up the device.

**Table 247. 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: debug with the Watchdog interrupt but no WDRESET enabled.<br>When this mode is selected, a watchdog counter underflow will set the WDINT flag and the Watchdog interrupt request will be generated.<br>**Remark:** In interrupt mode, check the WDINT flag. If this flag is set, the interrupt is true and can be serviced by the interrupt routine. If this flag is not set, the interrupt should be ignored. |
| 1 | 1 | Watchdog reset mode: operate with the Watchdog interrupt and WDRESET enabled.<br>When this mode is selected, a watchdog counter underflow will reset the microcontroller. Although the Watchdog interrupt is also enabled in this case (WDEN = 1) it will not be recognized since the watchdog reset will clear the WDINT flag. |

### 17.7.2 Watchdog Timer Constant register (WDTC - 0x4000 4004)

The WDTC register determines the time-out value. Every time a feed sequence occurs the WDTC content is reloaded in to the Watchdog timer. It's a 32-bit register with 8 LSB set to 1 on reset. Writing values below 0xFF will cause 0x0000 00FF to be loaded to the WDTC. Thus the minimum time-out interval is $T_{WDCLK} \times 256 \times 4$.

**Table 248. Watchdog Constant register (WDTC - address 0x4000 4004) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 23:0 | Count | Watchdog time-out interval. | 0x0000 00FF |
| 31:25 | - | Reserved | - |

### 17.7.3 Watchdog Feed register (WDFEED - 0x4000 4008)

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. The reset will be generated during the second PCLK following an incorrect access to a Watchdog register during a feed sequence.

Interrupts should be disabled during the feed sequence. An abort condition will occur if an interrupt happens during the feed sequence.

**Table 249. Watchdog Feed register (WDFEED - address 0x4000 4008) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | Feed | Feed value should be 0xAA followed by 0x55. | NA |
| 31:8 | - | Reserved | - |

### 17.7.4 Watchdog Timer Value register (WDTV - 0x4000 400C)

The WDTV register is used to read the current value of Watchdog timer.

When reading the value of the 24-bit timer, 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 250. Watchdog Timer Value register (WDTV - address 0x4000 000C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 23:0 | Count | Counter timer value. | 0x0000 00FF |
| 31:24 | - | Reserved | - |

## 17.8 Block diagram

The block diagram of the Watchdog is shown below in the Figure 61. The synchronization logic (PCLK/WDCLK) is not shown in the block diagram.

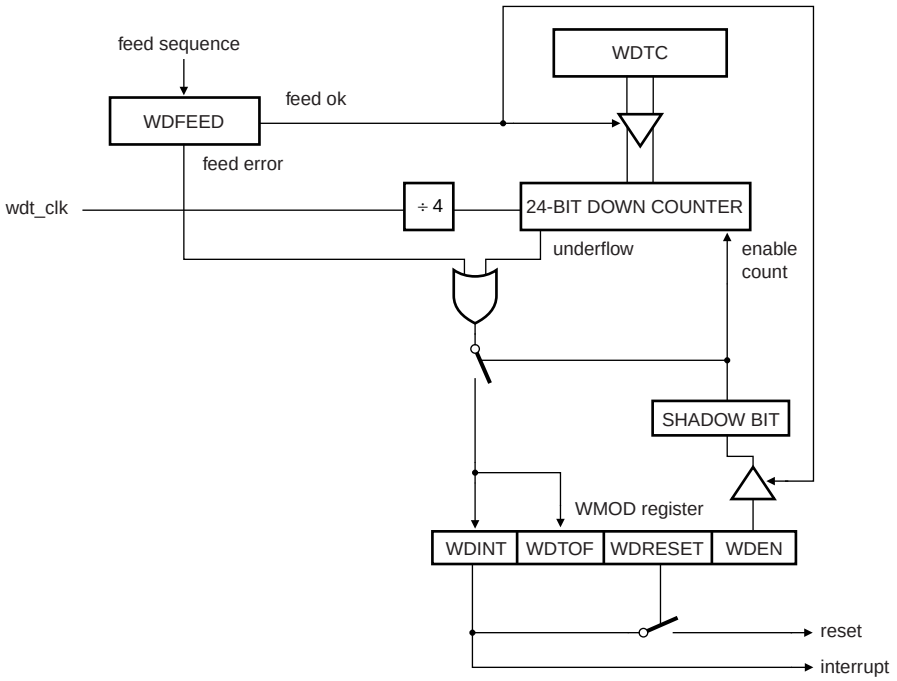



**Fig 61. Watchdog block diagram**

## 18.1 How to read this chapter

The system tick timer (SysTick timer) is part of the ARM Cortex-M0 core and is identical for all LPC111x and LPC11Cxx parts.

## 18.2 Basic configuration

The system tick timer is configured using the following registers:

1. Pins: The system tick timer uses no external pins.
2. Power: The system tick timer is enabled through the SysTick control register (Section 22.5.4.1). The system tick timer clock is fixed to half the frequency of the system clock.
3. Enable the clock source for the SysTick timer in the SYST_CSR register (Table 252).

## 18.3 Features

- Simple 24-bit timer.
- Uses dedicated exception vector.
- Clocked internally by the system clock or the system clock/2.

## 18.4 General description

The block diagram of the SysTick timer is shown below in the Figure 62.

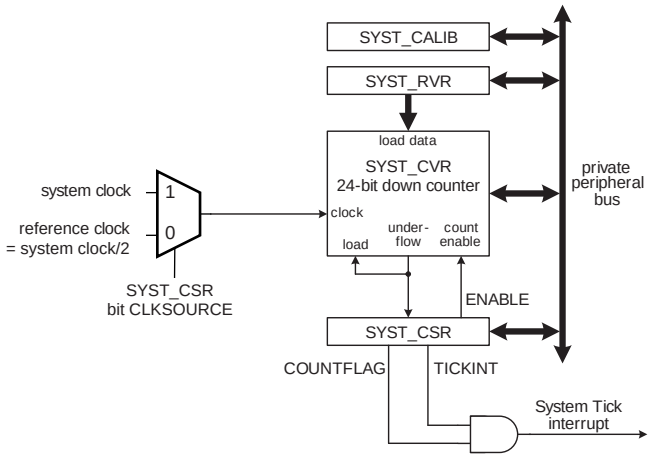



**Fig 62. System tick timer block diagram**

The SysTick timer is an integral part of the Cortex-M0. The SysTick timer is intended to generate a fixed 10 millisecond interrupt for use by an operating system or other system management software.

Since the SysTick timer is a part of the Cortex-M0, it facilitates porting of software by providing a standard timer that is available on Cortex-M0 based devices. The SysTick timer can be used for:

- An RTOS tick timer which fires at a programmable rate (for example 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the core clock.
- A simple counter. Software can use this to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNTFLAG bit-field in the control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

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

## 18.5 Register description

The systick timer registers are located on the ARM Cortex-M0 private peripheral bus (see Figure 70), and are part of the ARM Cortex-M0 core peripherals. For details, see Section 22.5.4.

**Table 251. Register overview: SysTick timer (base address 0xE000 E000)**

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| SYST_CSR | R/W | 0x010 | System Timer Control and status register | 0x000 0000 |
| SYST_RVR | R/W | 0x014 | System Timer Reload value register | 0 |
| SYST_CVR | R/W | 0x018 | System Timer Current value register | 0 |
| SYST_CALIB | R/W | 0x01C | System Timer Calibration value register | 0x4 |

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

### 18.5.1 System Timer Control and status register

The SYST_CSR register contains control information for the SysTick timer and provides a status flag. This register is part of the ARM Cortex-M0 core system timer register block. For a bit description of this register, see Section 22.5.4 “System timer, SysTick”.

This register determines the clock source for the system tick timer.

**Table 252. SysTick Timer Control and status register (SYST_CSR - 0xE000 E010) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | ENABLE | System Tick counter enable. When 1, the counter is enabled. When 0, the counter is disabled. | 0 |
| 1 | TICKINT | System Tick interrupt enable. When 1, the System Tick interrupt is enabled. When 0, the System Tick interrupt is disabled. When enabled, the interrupt is generated when the System Tick counter counts down to 0. | 0 |
| 2 | CLKSOURCE | System Tick clock source selection. When 1, the system clock (CPU) clock is selected. When 0, the system clock/2 is selected as the reference clock. | 0 |
| 15:3 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 16 | COUNTFLAG | Returns 1 if the SysTick timer counted to 0 since the last read of this register. | 0 |
| 31:17 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 18.5.2 System Timer Reload value register

The SYST_RVR register is set to the value that will be loaded into the SysTick timer whenever it counts down to zero. This register is loaded by software as part of timer initialization. The SYST_CALIB register may be read and used as the value for SYST_RVR register if the CPU is running at the frequency intended for use with the SYST_CALIB value.

**Table 253. System Timer Reload value register (SYST_RVR - 0xE000 E014) bit description**

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

## 18.5.3 System Timer Current value register

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

**Table 254. System Timer Current value register (SYST_CVR - 0xE000 E018) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 23:0 | CURRENT | Reading this register returns the current value of the System Tick counter. Writing any value clears the System Tick counter and the COUNTFLAG bit in STCTRL. | 0 |
| 31:24 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 18.5.4 System Timer Calibration value register (SYST_CALIB - 0xE000 E01C)

The value of the SYST_CALIB register is driven by the value of the SYSTCKCAL register in the system configuration block (see Table 33).

**Table 255. System Timer Calibration value register (SYST_CALIB - 0xE000 E01C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 23:0 | TENMS | | See Table 354. | 0x4 |
| 29:24 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 30 | SKEW | | See Table 354. | 0 |
| 31 | NOREF | | See Table 354. | 0 |

## 18.6 Functional description

The SysTick timer is a 24-bit timer that counts down to zero and generates an interrupt. The intent is to provide a fixed 10 millisecond time interval between interrupts. The SysTick timer is clocked from the CPU clock (the system clock, see Figure 3) or from the reference clock, which is fixed to half the frequency of the CPU clock. In order to generate recurring interrupts at a specific interval, the SYST_RVR register must be initialized with the correct value for the desired interval. A default value is provided in the SYST_CALIB register and may be changed by software. The default value gives a 10 millisecond interrupt rate if the CPU clock is set to 50 MHz.

## 18.7 Example timer calculations

To use the system tick timer, do the following:

1. Program the SYST_RVR register with the reload value RELOAD to obtain the desired time interval.
2. Clear the SYST_CVR register by writing to it. This ensures that the timer will count from the SYST_RVR value rather than an arbitrary value when the timer is enabled.
3. Program the SYST_SCR register with the value 0x7 which enables the SysTick timer and the SysTick timer interrupt.

The following example illustrates selecting the SysTick timer reload value to obtain a 10 ms time interval with the LPC111x/LPC11Cxx system clock set to 50 MHz.

#### Example (system clock = 50 MHz)

The system tick clock = system clock = 50 MHz. Bit CLKSOURCE in the SYST_CSR register set to 1 (system clock).

RELOAD = (system tick clock frequency $\times$ 10 ms) $-1$ = (50 MHz $\times$ 10 ms) $-1$ = 500000 $-1$ = 499999 = 0x0007A11F.

## 19.1 How to read this chapter

The ADC block is identical for all LPC111x and LPC11Cxx parts.

## 19.2 Basic configuration

The ADC is configured using the following registers:

1. Pins: The ADC pin functions are configured in the IOCONFIG register block (Section 7.4).
2. Power and peripheral clock: In the SYSAHBCLKCTRL register, set bit 13 (Table 20). Power to the ADC at run-time is controlled through the PDRUNCFG register (Table 41).

**Remark:** Basic clocking for the A/D converters is determined by the APB clock (PCLK). A programmable divider is included in the A/D converter to scale this clock to the 4.5 MHz (max) clock needed by the successive approximation process. An accurate conversion requires 11 clock cycles.

## 19.3 Features

- 10-bit successive approximation Analog-to-Digital Converter (ADC).
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range 0 to 3.6 V. Do not exceed the $V_{DD}$ voltage level.
- 10-bit conversion time $\geq$ 2.44 μs.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.
- Individual result registers for each A/D channel to reduce interrupt overhead.

## 19.4 Pin description

Table 256 gives a brief summary of the ADC related pins.

**Table 256. ADC pin description**

| Pin | Type | Description |
|---|---|---|
| AD[7:0] | Input | **Analog Inputs.** The A/D converter cell can measure the voltage on any of these input signals.<br>**Remark:** While the pins are 5 V tolerant in digital mode, the maximum input voltage must not exceed $V_{DD}$ when the pins are configured as analog inputs. |
| $V_{DD}$ | Input | $V_{REF}$; Reference voltage. |

The ADC function must be selected via the IOCON registers in order to get accurate voltage readings on the monitored pin. For a pin hosting an ADC input, it is not possible to have a have a digital function selected and yet get valid ADC readings. An inside circuit disconnects ADC hardware from the associated pin whenever a digital function is selected on that pin.

## 19.5 Register description

The ADC contains registers organized as shown in Table 257.

**Table 257. Register overview: ADC (base address 0x4001 C000)**

| Name | Access | Address offset | Description | Reset Value[1] |
|---|---|---|---|---|
| AD0CR | R/W | 0x000 | A/D Control Register. The AD0CR register must be written to select the operating mode before A/D conversion can occur. | 0x0000 0000 |
| AD0GDR | R/W | 0x004 | A/D Global Data Register. Contains the result of the most recent A/D conversion. | NA |
| - | - | 0x008 | Reserved. | - |
| AD0INTEN | R/W | 0x00C | A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt. | 0x0000 0100 |
| AD0DR0 | R/W | 0x010 | A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0 | NA |
| AD0DR1 | R/W | 0x014 | A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1. | NA |
| AD0DR2 | R/W | 0x018 | A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2. | NA |
| AD0DR3 | R/W | 0x01C | A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3. | NA |
| AD0DR4 | R/W | 0x020 | A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4. | NA |
| AD0DR5 | R/W | 0x024 | A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5. | NA |
| AD0DR6 | R/W | 0x028 | A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6. | NA |
| AD0DR7 | R/W | 0x02C | A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7. | NA |
| AD0STAT | RO | 0x030 | A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt flag. | 0 |

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

### 19.5.1 A/D Control Register (AD0CR - 0x4001 C000)

The A/D Control Register provides bits to select A/D channels to be converted, A/D timing, A/D modes, and the A/D start trigger.

**Table 258. A/D Control Register (AD0CR - address 0x4001 C000) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 7:0 | SEL | | Selects which of the AD7:0 pins is (are) to be sampled and converted. Bit 0 selects Pin AD0, bit 1 selects pin AD1,..., and bit 7 selects pin AD7.<br>In software-controlled mode (BURST = 0), only one channel can be selected, i.e. only one of these bits should be 1.<br>In hardware scan mode (BURST = 1), any numbers of channels can be selected, i.e any or all bits can be set to 1. If all bits are set to 0, channel 0 is selected automatically (SEL = 0x01). | 0x00 |
| 15:8 | CLKDIV | | The APB clock (PCLK) is divided by CLKDIV +1 to produce the clock for the ADC, which should be less than or equal to 4.5 MHz. Typically, software should program the smallest value in this field that yields a clock of 4.5 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable. | 0 |
| 16 | BURST | | Burst mode<br>**Remark:** If BURST is set to 1, the ADGINTEN bit in the AD0INTEN register (Table 261) must be set to 0. | 0 |
| | | 0 | Software-controlled mode: Conversions are software-controlled and require 11 clocks. | |
| | | 1 | Hardware scan mode: The AD converter does repeated conversions at the rate selected by the CLKS field, scanning (if necessary) through the pins selected by 1s in the SEL field. The first conversion after the start corresponds to the least-significant bit set to 1 in the SEL field, then the next higher bits (pins) set to 1 are scanned if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion in progress when this bit is cleared will be completed.<br>**Important:** START bits must be 000 when BURST = 1 or conversions will not start. | |
| 19:17 | CLKS | | This field selects the number of clocks used for each conversion in Burst mode, and the number of bits of accuracy of the result in the LS bits of ADDR, between 11 clocks (10 bits) and 4 clocks (3 bits). | 000 |
| | | 0x0 | 11 clocks / 10 bits | |
| | | 0x1 | 10 clocks / 9 bits | |
| | | 0x2 | 9 clocks / 8 bits | |
| | | 0x3 | 8 clocks / 7 bits | |
| | | 0x4 | 7 clocks / 6 bits | |
| | | 0x5 | 6 clocks / 5 bits | |
| | | 0x6 | 5 clocks / 4 bits | |
| | | 0x7 | 4 clocks / 3 bits | |
| 23:20 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

**Table 258. A/D Control Register (AD0CR - address 0x4001 C000) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 26:24 | START | | When the BURST bit is 0, these bits control whether and when an A/D conversion is started: | 0 |
| | | 0x0 | No start (this value should be used when clearing PDN to 0). | |
| | | 0x1 | Start conversion now. | |
| | | 0x2 | Start conversion when the edge selected by bit 27 occurs on PIO0_2/SSEL/CT16B0_CAP0. | |
| | | 0x3 | Start conversion when the edge selected by bit 27 occurs on PIO1_5/DIR/CT32B0_CAP0. | |
| | | 0x4 | Start conversion when the edge selected by bit 27 occurs on CT32B0_MAT0[1]. | |
| | | 0x5 | Start conversion when the edge selected by bit 27 occurs on CT32B0_MAT1[1]. | |
| | | 0x6 | Start conversion when the edge selected by bit 27 occurs on CT16B0_MAT0[1]. | |
| | | 0x7 | Start conversion when the edge selected by bit 27 occurs on CT16B0_MAT1[1]. | |
| 27 | EDGE | | This bit is significant only when the START field contains 010-111. In these cases: | 0 |
| | | 0 | Start conversion on a rising edge on the selected CAP/MAT signal. | |
| | | 1 | Start conversion on a falling edge on the selected CAP/MAT signal. | |
| 31:28 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

[1] Note that this does not require that the timer match function appear on a device pin.

### 19.5.2 A/D Global Data Register (AD0GDR - 0x4001 C004)

The A/D Global Data Register contains the result of the most recent A/D conversion. This includes the data, DONE, and Overrun flags, and the number of the A/D channel to which the data relates.

**Table 259. A/D Global Data Register (AD0GDR - address 0x4001 C004) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 5:0 | - | Reserved. These bits always read as zeroes. | 0 |
| 15:6 | V_VREF | When DONE is 1, this field contains a binary fraction representing the voltage on the ADn pin selected by the SEL field, divided by the voltage on the $V_{DD}$ pin. Zero in the field indicates that the voltage on the ADn pin was less than, equal to, or close to that on $V_{SS}$, while 0x3FF indicates that the voltage on ADn was close to, equal to, or greater than that on $V_{REF}$. | X |
| 23:16 | - | Reserved. These bits always read as zeroes. | 0 |
| 26:24 | CHN | These bits contain the channel from which the result bits V_VREF were converted. | X |
| 29:27 | - | Reserved. These bits always read as zeroes. | 0 |
| 30 | OVERRUN | This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the V_VREF bits. | 0 |
| 31 | DONE | This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started. | 0 |

### 19.5.3 A/D Status Register (AD0STAT - 0x4001 C030)

The A/D Status register allows checking the status of all A/D channels simultaneously. The DONE and OVERRUN flags appearing in the ADDRn register for each A/D channel are mirrored in ADSTAT. The interrupt flag (the logical OR of all DONE flags) is also found in ADSTAT.

**Table 260. A/D Status Register (AD0STAT - address 0x4001 C030) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | DONE | These bits mirror the DONE status flags that appear in the result register for each A/D channel n. | 0 |
| 15:8 | OVERRUN | These bits mirror the OVERRRUN status flags that appear in the result register for each A/D channel n. Reading ADSTAT allows checking the status of all A/D channels simultaneously. | 0 |
| 16 | ADINT | This bit is the A/D interrupt flag. It is one when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register. | 0 |
| 31:17 | - | Reserved. Unused, always 0. | 0 |

### 19.5.4 A/D Interrupt Enable Register (AD0INTEN - 0x4001 C00C)

This register allows control over which A/D channels generate an interrupt when a conversion is complete. For example, it may be desirable to use some A/D channels to monitor sensors by continuously performing conversions on them. The most recent results are read by the application program whenever they are needed. In this case, an interrupt is not desirable at the end of each conversion for some A/D channels.

**Table 261. A/D Interrupt Enable Register (AD0INTEN - address 0x4001 C00C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | ADINTEN | These bits allow control over which A/D channels generate interrupts for conversion completion. When bit 0 is one, completion of a conversion on A/D channel 0 will generate an interrupt, when bit 1 is one, completion of a conversion on A/D channel 1 will generate an interrupt, etc. | 0x00 |
| 8 | ADGINTEN | When 1, enables the global DONE flag in ADDR to generate an interrupt. When 0, only the individual A/D channels enabled by ADINTEN 7:0 will generate interrupts.<br>**Remark:** This bit must be set to 0 in burst mode (BURST = 1 in the AD0CR register). | 1 |
| 31:9 | - | Reserved. Unused, always 0. | 0 |

### 19.5.5 A/D Data Registers (AD0DR0 to AD0DR7 - 0x4001 C010 to 0x4001 C02C)

The A/D Data Register hold the result when an A/D conversion is complete, and also include the flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

**Table 262. A/D Data Registers (AD0DR0 to AD0DR7 - addresses 0x4001 C010 to 0x4001 C02C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 5:0 | - | Reserved. | 0 |
| 15:6 | V_VREF | When DONE is 1, this field contains a binary fraction representing the voltage on the ADn pin, divided by the voltage on the $V_{REF}$ pin. Zero in the field indicates that the voltage on the ADn pin was less than, equal to, or close to that on $V_{REF}$, while 0x3FF indicates that the voltage on AD input was close to, equal to, or greater than that on $V_{REF}$. | NA |
| 29:16 | - | Reserved. | 0 |
| 30 | OVERRUN | This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the V_VREF bits.This bit is cleared by reading this register. | 0 |
| 31 | DONE | This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read. | 0 |

## 19.6 Operation

### 19.6.1 Hardware-triggered conversion

If the BURST bit in the ADCR0 is 0 and the START field contains 010-111, the A/D converter will start a conversion when a transition occurs on a selected pin or timer match signal.

### 19.6.2 Interrupts

An interrupt is requested to the interrupt controller when the ADINT bit in the ADSTAT register is 1. The ADINT bit is one when any of the DONE bits of A/D channels that are enabled for interrupts (via the ADINTEN register) are one. Software can use the Interrupt Enable bit in the interrupt controller that corresponds to the ADC to control whether this results in an interrupt. The result register for an A/D channel that is generating an interrupt must be read in order to clear the corresponding DONE flag.

### 19.6.3 Accuracy vs. digital receiver

While the A/D converter can be used to measure the voltage on any ADC input pin, regardless of the pin's setting in the IOCON block, selecting the ADC in the IOCON registers function improves the conversion accuracy by disabling the pin's digital receiver (see also Section 7.3.4).

## 20.1 How to read this chapter

See Table 263 for different flash configurations.

**Table 263. LPC111x/LPC11Cx flash configurations**

| Type number | Flash | ISP via UART | ISP via C_CAN |
|---|---|---|---|
| LPC1111 | 8 kB | yes | no |
| LPC1112 | 16 kB | yes | no |
| LPC1113 | 24 kB | yes | no |
| LPC1114 | 32 kB | yes | no |
| LPC11C12/C22 | 16 kB | yes | yes |
| LPC11C14/C24 | 32 kB | yes | yes |

**Remark:** In addition to the ISP and IAP commands, a register can be accessed in the flash controller block to configure flash memory access times, see Section 20.9.

## 20.2 Features

- In-System Programming: In-System programming (ISP) is programming or reprogramming the on-chip flash memory, using the bootloader software and UART serial port or the C_CAN interface. This can be done when the part resides in the end-user board.
- In-Application Programming: In-Application (IAP) programming is performing erase and write operation on the on-chip flash memory, as directed by the end-user application code.
- Flash access times can be configured through a register in the flash controller block.
- Erase time for one sector is 100 ms ± 5%. Programming time for one block of 256 bytes is 1 ms ± 5%.

## 20.3 General description

### 20.3.1 Bootloader

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

The bootloader code is executed every time the part is powered on or reset. The loader can execute the ISP command handler or the user application code. A LOW level after reset at the PIO0_1 pin is considered as an external hardware request to start the ISP command handler either via UART or C_CAN, if present.

If the C_CAN interface is present (LPC11Cx parts), the state of pin PIO0_3 at reset together with a LOW level on pin PIO0_1 determines whether UART ISP or C_CAN ISP routines are called:

- If PIO0_3 is LOW, the bootloader configures the C_CAN interface and calls the C_CAN ISP command handler.
- PIO0_3 is HIGH, the bootloader configures the UART serial port and calls the UART ISP command handler (this is the default).

**Remark:** On parts without C_CAN interface, the state of pin PIO0_3 does not matter.

Assuming that power supply pins are on their nominal levels when the rising edge on RESET pin is generated, it may take up to 3 ms before PIO0_1 is sampled and the decision whether to continue with user code or ISP handler is made. If PIO0_1 is sampled low and the watchdog overflow flag is set, the external hardware request to start the ISP command handler is ignored. If there is no request for the ISP command handler execution (PIO0_1 is sampled HIGH after reset), a search is made for a valid user program. If a valid user program is found then the execution control is transferred to it. If a valid user program is not found, the auto-baud routine is invoked.

**Remark:** The sampling of pin PIO0_1 can be disabled through programming flash location 0x0000 02FC (see Section 20.3.7.1).

### 20.3.2 Memory map after any reset

The boot block is 16 kB in size. The boot block is located in the memory region starting from the address 0x1FFF 0000. The bootloader is designed to run from this memory area, but both the ISP and IAP software use parts of the on-chip RAM. The RAM usage is described later in this chapter. The interrupt vectors residing in the boot block of the on-chip flash memory also become active after reset, i.e., the bottom 512 bytes of the boot block are also visible in the memory region starting from the address 0x0000 0000.

### 20.3.3 Criterion for Valid User Code

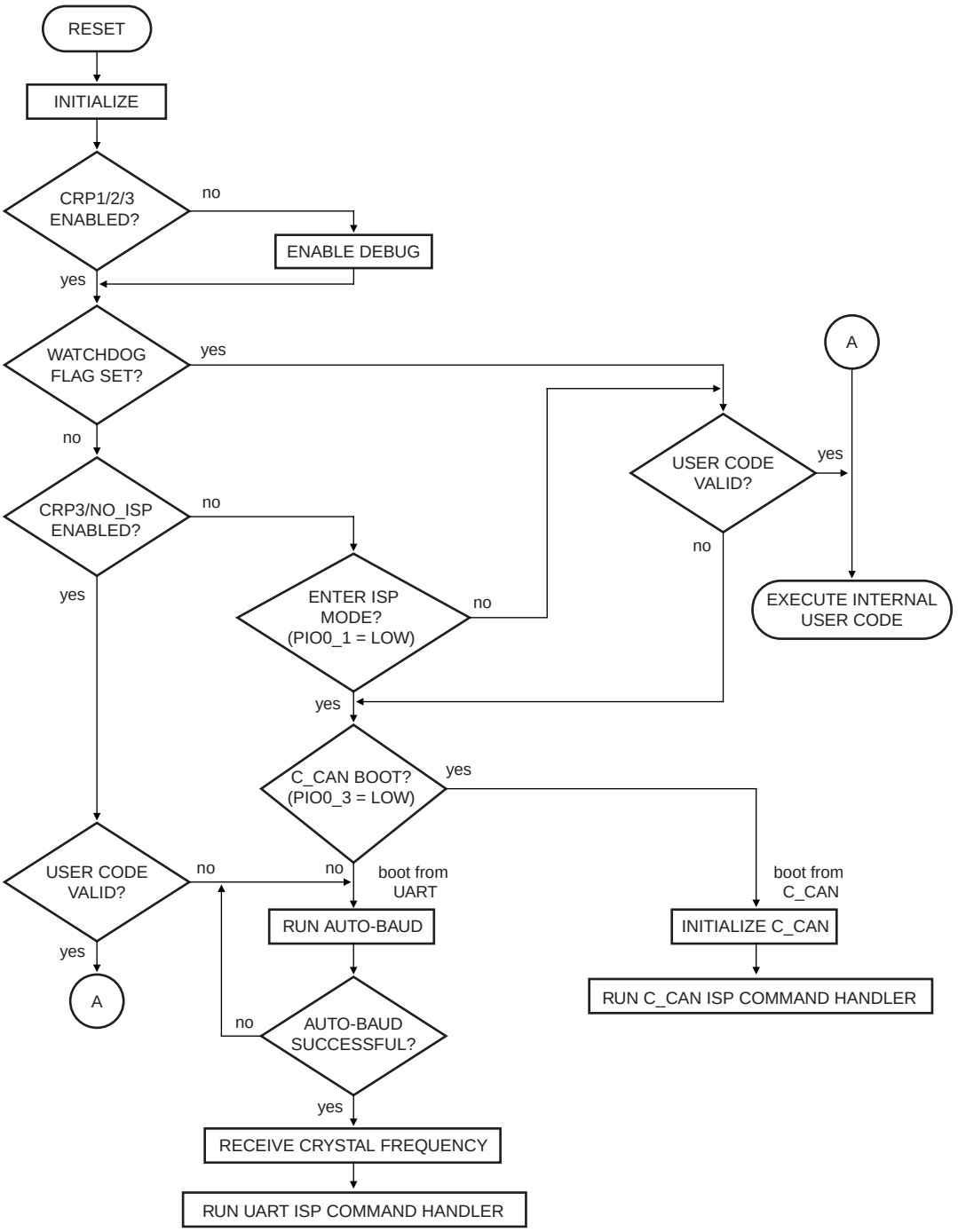
Criterion for valid user code: The reserved Cortex-M0 exception vector location 7 (offset 0x 0000 001C in the vector table) should contain the 2's complement of the check-sum of table entries 0 through 6. This causes the checksum of the first 8 table entries to be 0. The bootloader code checksums the first 8 locations in sector 0 of the flash. If the result is 0, then execution control is transferred to the user code.

If the signature is not valid, the auto-baud routine synchronizes with the host via serial port 0. The host should send a '?' (0x3F) as a synchronization character and wait for a response. The host side serial port settings should be 8 data bits, 1 stop bit and no parity. The auto-baud routine measures the bit time of the received synchronization character in terms of its own frequency and programs the baud rate generator of the serial port. It also sends an ASCII string ("Synchronized<CR><LF>") to the host. In response to this host should send the same string ("Synchronized<CR><LF>"). The auto-baud routine looks at the received characters to verify synchronization. If synchronization is verified then "OK<CR><LF>" string is sent to the host. The host should respond by sending the crystal frequency (in kHz) at which the part is running. For example, if the part is running at 10 MHz, the response from the host should be "10000<CR><LF>". "OK<CR><LF>" string is sent to the host after receiving the crystal frequency. If synchronization is not verified then

the auto-baud routine waits again for a synchronization character. For auto-baud to work correctly in case of user invoked ISP, the CCLK frequency should be greater than or equal to 10 MHz.

Once the crystal frequency is received the part is initialized and the ISP command handler is invoked. For safety reasons an "Unlock" command is required before executing the commands resulting in flash erase/write operations and the "Go" command. The rest of the commands can be executed without the unlock command. The Unlock command is required to be executed once per ISP session. The Unlock command is explained in Section 20.5 “UART ISP commands” on page 299.

### 20.3.4 Boot process flowchart




(1) For details on handling the crystal frequency, see Section 20.7.8 “Reinvoke ISP (IAP)” on page 316

**Fig 63. Boot process flowchart**

### 20.3.5 Sector numbers

Some IAP and ISP commands operate on sectors and specify sector numbers. The following table shows the correspondence between sector numbers and memory addresses for LPC111x/LPC11Cxx devices.

**Table 264. Flash sector configuration**

| Sector number | Sector size | Address range | LPC1111 8 kB flash | LPC1112/ LPC11C12/ LPC11C22 16 kB flash | LPC1113 24 kB flash | LPC1114/ LPC11C14/ LPC11C24 32 kB flash |
|---|---|---|---|---|---|---|
| 0 | 4 kB | 0x0000 0000 - 0x0000 0FFF | yes | yes | yes | yes |
| 1 | 4 kB | 0x0000 1000 - 0x0000 1FFF | yes | yes | yes | yes |
| 2 | 4 kB | 0x0000 2000 - 0x0000 2FFF | - | yes | yes | yes |
| 3 | 4 kB | 0x0000 3000 - 0x0000 3FFF | - | yes | yes | yes |
| 4 | 4 kB | 0x0000 4000 - 0x0000 4FFF | - | - | yes | yes |
| 5 | 4 kB | 0x0000 5000 - 0x0000 5FFF | - | - | yes | yes |
| 6 | 4 kB | 0x0000 6000 - 0x0000 6FFF | - | - | - | yes |
| 7 | 4 kB | 0x0000 7000 - 0x0000 7FFF | | - | - | yes |

### 20.3.6 Flash content protection mechanism

The LPC111x/LPC11C1x is equipped with the Error Correction Code (ECC) capable Flash memory. The purpose of an error correction module is twofold. Firstly, it decodes data words read from the memory into output data words. Secondly, it encodes data words to be written to the memory. The error correction capability consists of single bit error correction with Hamming code.

The operation of ECC is transparent to the running application. The ECC content itself is stored in a flash memory not accessible by user's code to either read from it or write into it on its own. A byte of ECC corresponds to every consecutive 128 bits of the user accessible Flash. Consequently, Flash bytes from 0x0000 0000 to 0x0000 000F are protected by the first ECC byte, Flash bytes from 0x0000 0010 to 0x0000 001F are protected by the second ECC byte, etc.

Whenever the CPU requests a read from user's Flash, both 128 bits of raw data containing the specified memory location and the matching ECC byte are evaluated. If the ECC mechanism detects a single error in the fetched data, a correction will be applied before data are provided to the CPU. When a write request into the user's Flash is made, write of user specified content is accompanied by a matching ECC value calculated and stored in the ECC memory.

When a sector of Flash memory is erased, the corresponding ECC bytes are also erased. Once an ECC byte is written, it can not be updated unless it is erased first. Therefore, for the implemented ECC mechanism to perform properly, data must be written into the flash memory in groups of 16 bytes (or multiples of 16), aligned as described above.

### 20.3.7 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 265. Code Read Protection options**

| Name | Pattern programmed in 0x0000 02FC | Description |
|---|---|---|
| NO_ISP | 0x4E69 7370 | Prevents sampling of pin PIO0_1 for entering ISP mode. PIO0_1 is available for other uses. |
| CRP1 | 0x12345678 | Access to chip via the SWD pins is disabled. This mode allows partial flash update using the following ISP commands and restrictions:<br>• Write to RAM command cannot access RAM below 0x1000 0300.<br>• Copy RAM to flash command can not write to Sector 0.<br>• Erase command can erase Sector 0 only when all sectors are selected for erase.<br>• Compare command is disabled.<br>• Read Memory command is disabled.<br>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:<br>• Read Memory<br>• Write to RAM<br>• Go<br>• Copy RAM to flash<br>• Compare<br>When CRP2 is enabled the ISP erase command only allows erasure of all user sectors. |
| CRP3 | 0x43218765 | Access to chip via the SWD pins is disabled. ISP entry by pulling PIO0_1 LOW is disabled if a valid user code is present in flash sector 0.<br>This mode effectively disables ISP override using PIO0_1 pin. It is up to the user's application to provide a flash update mechanism using IAP calls or call reinvoke ISP command to enable flash update via UART.<br>**Caution: If CRP3 is selected, no future factory testing can be performed on the device.** |

**Table 266. Code Read Protection hardware/software interaction**

| CRP option | User Code Valid | PIO0_1 pin at reset | SWD enabled | LPC111x/ LPC11Cxx enters ISP mode | partial flash update in ISP mode |
|---|---|---|---|---|---|
| None | No | x | Yes | Yes | Yes |
| None | Yes | High | Yes | No | NA |
| None | Yes | Low | Yes | Yes | Yes |
| CRP1 | Yes | High | No | No | NA |
| CRP1 | Yes | Low | No | Yes | Yes |
| CRP2 | Yes | High | No | No | NA |
| CRP2 | Yes | Low | No | Yes | No |
| CRP3 | Yes | x | No | No | NA |
| CRP1 | No | x | No | Yes | Yes |
| CRP2 | No | x | No | Yes | No |
| CRP3 | No | x | No | Yes | No |

**Table 267. ISP commands allowed for different CRP levels**

| ISP command | CRP1 | CRP2 | CRP3 (no entry in ISP mode allowed) |
|---|---|---|---|
| Unlock | yes | yes | n/a |
| Set Baud Rate | yes | yes | n/a |
| Echo | yes | yes | n/a |
| Write to RAM | yes; above 0x1000 0300 only | no | n/a |
| Read Memory | no | no | n/a |
| Prepare sector(s) for write operation | yes | yes | n/a |
| Copy RAM to flash | yes; not to sector 0 | no | n/a |
| Go | no | no | n/a |
| Erase sector(s) | yes; sector 0 can only be erased when all sectors are erased. | yes; all sectors only | n/a |
| Blank check sector(s) | no | no | n/a |
| Read Part ID | yes | yes | n/a |
| Read Boot code version | yes | yes | n/a |
| Compare | no | no | n/a |
| ReadUID | yes | yes | n/a |

In case a CRP mode is enabled and access to the chip is allowed via the ISP, an unsupported or restricted ISP command will be terminated with return code CODE_READ_PROTECTION_ENABLED.

#### 20.3.7.1 ISP entry protection

In addition to the three CRP modes, the user can prevent the sampling of pin PIO0_1 for entering ISP mode and thereby release pin PIO0_1 for other uses. This is called the NO_ISP mode. The NO_ISP mode can be entered by programming the pattern 0x4E69 7370 at location 0x0000 02FC.

## 20.4 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 UU-encoded format.

### 20.4.1 UART ISP command format

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

### 20.4.2 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).

### 20.4.3 UART ISP data format

The data stream is in UU-encoded format. The UU-encode algorithm converts 3 bytes of binary data in to 4 bytes of printable ASCII character set. It is more efficient than Hex format which converts 1 byte of binary data in to 2 bytes of ASCII hex. The sender should send the check-sum after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. The receiver should compare it with the check-sum of the received bytes. If the check-sum matches then the receiver should respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match the receiver should respond with "RESEND<CR><LF>". In response the sender should retransmit the bytes.

### 20.4.4 UART ISP flow control

A software XON/XOFF flow control scheme is used to prevent data loss due to buffer overrun. When the data arrives rapidly, the ASCII control character DC3 (stop) is sent to stop the flow of data. Data flow is resumed by sending the ASCII control character DC1 (start). The host should also support the same flow control scheme.

### 20.4.5 UART SP command abort

Commands can be aborted by sending the ASCII control character "ESC". This feature is not documented as a command under "ISP Commands" section. Once the escape code is received the ISP command handler waits for a new command.

### 20.4.6 Interrupts during UART ISP

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

### 20.4.7 Interrupts during IAP

The on-chip flash memory is not accessible during erase/write operations. When the user application code starts executing the interrupt vectors from the user flash area are active. The user should either disable interrupts, or ensure that user interrupt vectors are active in RAM and that the interrupt handlers reside in RAM, before making a flash erase/write IAP call. The IAP code does not use or disable interrupts.

### 20.4.8 RAM used by ISP command handler

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

### 20.4.9 RAM used by IAP command handler

Flash programming commands use the top 32 bytes of on-chip RAM. The maximum stack usage in the user allocated stack space is 128 bytes and it grows downwards.

## 20.5 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 268. UART ISP command summary**

| ISP Command | Usage | Described in |
|---|---|---|
| Unlock | U <Unlock Code> | Table 269 |
| Set Baud Rate | B <Baud Rate> <stop bit> | Table 270 |
| Echo | A <setting> | Table 271 |
| Write to RAM | W <start address> <number of bytes> | Table 272 |
| Read Memory | R <address> <number of bytes> | Table 273 |
| Prepare sector(s) for write operation | P <start sector number> <end sector number> | Table 274 |
| Copy RAM to flash | C <Flash address> <RAM address> <number of bytes> | Table 275 |
| Go | G <address> <Mode> | Table 276 |
| Erase sector(s) | E <start sector number> <end sector number> | Table 277 |
| Blank check sector(s) | I <start sector number> <end sector number> | Table 278 |
| Read Part ID | J | Table 279 |
| Read Boot code version | K | Table 281 |
| Compare | M <address1> <address2> <number of bytes> | Table 282 |
| ReadUID | N | Table 283 |

### 20.5.1 Unlock <Unlock code> (UART ISP)

**Table 269. UART ISP Unlock command**

| Command | U |
|---|---|
| Input | Unlock code: $23130_{10}$ |
| Return Code | CMD_SUCCESS \|<br>INVALID_CODE \|<br>PARAM_ERROR |
| Description | This command is used to unlock Flash Write, Erase, and Go commands. |
| Example | "U 23130<CR><LF>" unlocks the Flash Write/Erase & Go commands. |

### 20.5.2 Set Baud Rate <Baud Rate> <stop bit> (UART ISP)

**Table 270. UART ISP Set Baud Rate command**

| Command | B |
|---|---|
| Input | Baud Rate: 9600 \| 19200 \| 38400 \| 57600 \| 115200<br>Stop bit: 1 \| 2 |
| Return Code | CMD_SUCCESS \|<br>INVALID_BAUD_RATE \|<br>INVALID_STOP_BIT \|<br>PARAM_ERROR |
| Description | This command is used to change the baud rate. The new baud rate is effective after the command handler sends the CMD_SUCCESS return code. |
| Example | "B 57600 1<CR><LF>" sets the serial port to baud rate 57600 bps and 1 stop bit. |

### 20.5.3 Echo <setting> (UART ISP)

**Table 271. UART ISP Echo command**

| Command | A |
|---|---|
| Input | Setting: ON = 1 \| OFF = 0 |
| Return Code | CMD_SUCCESS \|<br>PARAM_ERROR |
| Description | The default setting for echo command is ON. When ON the ISP command handler sends the received serial data back to the host. |
| Example | "A 0<CR><LF>" turns echo off. |

### 20.5.4 Write to RAM <start address> <number of bytes> (UART ISP)

The host should send the data only after receiving the CMD_SUCCESS return code. The host should send the check-sum after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. When the data fits in less then 20 UU-encoded lines then the check-sum should be of the actual number of bytes sent. The ISP command handler compares it with the check-sum of the received bytes. If the check-sum matches, the ISP command handler responds with "OK<CR><LF>" to

continue further transmission. If the check-sum does not match, the ISP command handler responds with "RESEND<CR><LF>". In response the host should retransmit the bytes.

**Table 272. 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.<br>**Number of Bytes:** Number of bytes to be written. Count should be a multiple of 4 |
| Return Code | CMD_SUCCESS \|<br>ADDR_ERROR (Address not on word boundary) \|<br>ADDR_NOT_MAPPED \|<br>COUNT_ERROR (Byte count is not multiple of 4) \|<br>PARAM_ERROR \|<br>CODE_READ_PROTECTION_ENABLED |
| Description | This command is used to download data to RAM. Data should be in UU-encoded format. This command is blocked when code read protection is enabled. |
| Example | "W 268436224 4<CR><LF>" writes 4 bytes of data to address 0x1000 0300. |

### 20.5.5 Read Memory <address> <no. of bytes> (UART ISP)

The data stream is followed by the command success return code. The check-sum is sent after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters (bytes) i.e. it can hold 45 data bytes. When the data fits in less then 20 UU-encoded lines then the check-sum is of actual number of bytes sent. The host should compare it with the checksum of the received bytes. If the check-sum matches then the host should respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match then the host should respond with "RESEND<CR><LF>". In response the ISP command handler sends the data again.

**Table 273. 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.<br>**Number of Bytes:** Number of bytes to be read. Count should be a multiple of 4. |
| Return Code | CMD_SUCCESS followed by <actual data (UU-encoded)> \|<br>ADDR_ERROR (Address not on word boundary) \|<br>ADDR_NOT_MAPPED \|<br>COUNT_ERROR (Byte count is not a multiple of 4) \|<br>PARAM_ERROR \|<br>CODE_READ_PROTECTION_ENABLED |
| Description | This command is used to read data from RAM or flash memory. This command is blocked when code read protection is enabled. |
| Example | "R 268435456 4<CR><LF>" reads 4 bytes of data from address 0x1000 0000. |

### 20.5.6 Prepare sector(s) for write operation <start sector number> <end sector number> (UART ISP)

This command makes flash write/erase operation a two step process.

**Table 274. UART ISP Prepare sector(s) for write operation command**

| Command | P |
|---|---|
| Input | **Start Sector Number**<br>**End Sector Number:** Should be greater than or equal to start sector number. |
| Return Code | CMD_SUCCESS \|<br>BUSY \|<br>INVALID_SECTOR \|<br>PARAM_ERROR |
| Description | This command must be executed before executing "Copy RAM to flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot block can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers. |
| Example | "P 0 0<CR><LF>" prepares the flash sector 0. |

### 20.5.7 Copy RAM to flash <Flash address> <RAM address> <no of bytes> (UART ISP)

When writing to the flash, the following limitations apply:

1. The smallest amount of data that can be written to flash by the copy RAM to flash command is 256 byte (equal to one page).
2. One page consists of 16 flash words (lines), and the smallest amount that can be modified per flash write is one flash word (one line). This limitation follows from the application of ECC to the flash write operation, see Section 20.3.6.
3. To avoid write disturbance (a mechanism intrinsic to flash memories), an erase should be performed after following 16 consecutive writes inside the same page. Note that the erase operation then erases the entire sector.

   **Remark:** Once a page has been written to 16 times, it is still possible to write to other pages within the same sector without performing a sector erase (assuming that those pages have been erased previously).

**Table 275. 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.<br>**RAM Address(SRC):** Source RAM address from where data bytes are to be read.<br>**Number of Bytes:** Number of bytes to be written. Should be 256 \| 512 \| 1024 \| 4096. |
| Return Code | CMD_SUCCESS \|<br>SRC_ADDR_ERROR (Address not on word boundary) \|<br>DST_ADDR_ERROR (Address not on correct boundary) \|<br>SRC_ADDR_NOT_MAPPED \|<br>DST_ADDR_NOT_MAPPED \|<br>COUNT_ERROR (Byte count is not 256 \| 512 \| 1024 \| 4096) \|<br>SECTOR_NOT_PREPARED_FOR WRITE_OPERATION \|<br>BUSY \|<br>CMD_LOCKED \|<br>PARAM_ERROR \|<br>CODE_READ_PROTECTION_ENABLED |
| Description | This command is used to program the flash memory. The "Prepare Sector(s) for Write Operation" command should precede this command. The affected sectors are automatically protected again once the copy command is successfully executed. The boot block cannot be written by this command. This command is blocked when code read protection is enabled. |
| Example | "C 0 268467504 512<CR><LF>" copies 512 bytes from the RAM address 0x1000 0800 to the flash address 0. |

### 20.5.8 Go <address> <mode> (UART ISP)

**Table 276. 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.<br>**Mode:** T (Execute program in Thumb Mode) \| A (Execute program in ARM mode). |
| Return Code | CMD_SUCCESS \|<br>ADDR_ERROR \|<br>ADDR_NOT_MAPPED \|<br>CMD_LOCKED \|<br>PARAM_ERROR \|<br>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. |

### 20.5.9 Erase sector(s) <start sector number> <end sector number> (UART ISP)

**Table 277. UART ISP Erase sector command**

| Command | E |
|---|---|
| Input | **Start Sector Number**<br>**End Sector Number:** Should be greater than or equal to start sector number. |
| Return Code | CMD_SUCCESS \|<br>BUSY \|<br>INVALID_SECTOR \|<br>SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION \|<br>CMD_LOCKED \|<br>PARAM_ERROR \|<br>CODE_READ_PROTECTION_ENABLED |
| Description | This command is used to erase one or more sector(s) of on-chip flash memory. The boot block can not be erased using this command. This command only allows erasure of all user sectors when the code read protection is enabled. |
| Example | "E 2 3<CR><LF>" erases the flash sectors 2 and 3. |

### 20.5.10 Blank check sector(s) <sector number> <end sector number> (UART ISP)

**Table 278. UART ISP Blank check sector command**

| Command | I |
|---|---|
| Input | **Start Sector Number:**<br>**End Sector Number:** Should be greater than or equal to start sector number. |
| Return Code | CMD_SUCCESS \|<br>SECTOR_NOT_BLANK (followed by <Offset of the first non blank word location> <Contents of non blank word location>) \|<br>INVALID_SECTOR \|<br>PARAM_ERROR |
| Description | This command is used to blank check one or more sectors of on-chip flash memory.<br>**Blank check on sector 0 always fails as first 64 bytes are re-mapped to flash boot block.** |
| Example | "I 2 3<CR><LF>" blank checks the flash sectors 2 and 3. |

### 20.5.11 Read Part Identification number (UART ISP)

**Table 279. UART ISP Read Part Identification command**

| Command | J |
|---|---|
| Input | None. |
| Return Code | CMD_SUCCESS followed by part identification number in ASCII (see Table 280). |
| Description | This command is used to read the part identification number. |

**Table 280. LPC111x and LPC11Cxx part identification numbers**

| Device | Hex coding |
|---|---|
| **LPC111x** | |
| LPC1111FHN33/101 | 0x041E 502B |
| LPC1111FHN33/102 | 0x2516 D02B |
| LPC1111FHN33/201 | 0x0416 502B |
| LPC1111FHN33/202 | 0x2516 902B |
| LPC1112FHN33/101 | 0x042D 502B |
| LPC1112FHN33/102 | 0x2524 D02B |
| LPC1112FHN33/201 | 0x0425 502B |
| LPC1112FHN33/202 | 0x2524 902B |
| LPC1113FHN33/201 | 0x0434 502B |
| LPC1113FHN33/202 | 0x2532 902B |
| LPC1113FHN33/301 | 0x0434 102B |
| LPC1113FHN33/302 | 0x2532 102B |
| LPC1113FBD48/301 | 0x0434 102B |
| LPC1113FBD48/302 | 0x2532 102B |
| LPC1114FHN33/201 | 0x0444 502B |
| LPC1114FHN33/202 | 0x2540 902B |
| LPC1114FHN33/301 | 0x0444 102B |
| LPC1114FHN33/302 | 0x2540 102B |
| LPC1114FBD48/301 | 0x0444 102B |
| LPC1114FBD48/302 | 0x2540 102B |
| LPC1114FA44/301 | 0x0444 102B |
| LPC1114FA44/302 | 0x2540 102B |
| **LPC11Cxx** | |
| LPC11C12FBD48/301 | 0x1421 102B |
| LPC11C14FBD48/301 | 0x1440 102B |
| LPC11C22FBD48/301 | 0x1431 102B |
| LPC11C24FBD48/301 | 0x1430 102B |

## 20.5.12 Read Boot code version number (UART ISP)

**Table 281. 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. |

### 20.5.13 Compare <address1> <address2> <no of bytes> (UART ISP)

**Table 282. 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.<br>**Address2 (SRC):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary.<br>**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)<br>COMPARE_ERROR \| (Followed by the offset of first mismatch)<br>COUNT_ERROR (Byte count is not a multiple of 4) \|<br>ADDR_ERROR \|<br>ADDR_NOT_MAPPED \|<br>PARAM_ERROR |
| Description | This command is used to compare the memory contents at two locations.<br>**Compare result may not be correct when source or destination address contains any of the first 512 bytes starting from address zero. First 512 bytes are re-mapped to boot ROM** |
| Example | "M 8192 268468224 4<CR><LF>" compares 4 bytes from the RAM address 0x1000 8000 to the 4 bytes from the flash address 0x2000. |

### 20.5.14 ReadUID (UART ISP)

**Table 283. UART ISP ReadUID command**

| Command | N |
|---|---|
| Input | None |
| Return Code | CMD_SUCCESS followed by four 32-bit words of E-sort test information in ASCII format. The word sent at the lowest address is sent first. |
| Description | This command is used to read the unique ID. |

### 20.5.15 UART ISP Return Codes

**Table 284. UART ISP Return Codes Summary**

| Return Code | Mnemonic | Description |
|---|---|---|
| 0 | CMD_SUCCESS | Command is executed successfully. Sent by ISP handler only when command given by the host has been completely and successfully executed. |
| 1 | INVALID_COMMAND | Invalid command. |
| 2 | SRC_ADDR_ERROR | Source address is not on word boundary. |
| 3 | DST_ADDR_ERROR | Destination address is not on a correct boundary. |
| 4 | SRC_ADDR_NOT_MAPPED | Source address is not mapped in the memory map. Count value is taken in to consideration where applicable. |
| 5 | DST_ADDR_NOT_MAPPED | Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable. |

**Table 284. UART ISP Return Codes Summary**

| Return Code | Mnemonic | Description |
|---|---|---|
| 6 | COUNT_ERROR | Byte count is not multiple of 4 or is not a permitted value. |
| 7 | INVALID_SECTOR | Sector number is invalid or end sector number is greater than start sector number. |
| 8 | SECTOR_NOT_BLANK | Sector is not blank. |
| 9 | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | Command to prepare sector for write operation was not executed. |
| 10 | COMPARE_ERROR | Source and destination data not equal. |
| 11 | BUSY | Flash programming hardware interface is busy. |
| 12 | PARAM_ERROR | Insufficient number of parameters or invalid parameter. |
| 13 | ADDR_ERROR | Address is not on word boundary. |
| 14 | ADDR_NOT_MAPPED | Address is not mapped in the memory map. Count value is taken in to consideration where applicable. |
| 15 | CMD_LOCKED | Command is locked. |
| 16 | INVALID_CODE | Unlock code is invalid. |
| 17 | INVALID_BAUD_RATE | Invalid baud rate setting. |
| 18 | INVALID_STOP_BIT | Invalid stop bit setting. |
| 19 | CODE_READ_PROTECTION_ENABLED | Code read protection enabled. |

## 20.6 C_CAN communication protocol

**Remark:** The C_CAN interface is available on LPC11Cxx parts only.

The C_CAN bootloader is activated by the ROM reset handler automatically if PIO0_3 is LOW on reset and the ISP entry enabled (PIO0_1 LOW). The C_CAN bootloader 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 bootloader 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 285.

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 285. C_CAN ISP and UART ISP command summary**

| ISP Command | C_CAN usage | UART usage |
|---|---|---|
| Unlock | Section 20.6.3 | Table 269 |
| Set Baud Rate | n/a | Table 270 |
| Echo | n/a | Table 271 |
| Write to RAM | Section 20.6.4 | Table 272 |
| Read Memory | Section 20.6.5 | Table 273 |
| Prepare sector(s) for write operation | Section 20.6.6 | Table 274 |
| Copy RAM to flash | Section 20.6.7 | Table 275 |
| Go | Section 20.6.8 | Table 276 |
| Erase sector(s) | Section 20.6.9 | Table 277 |
| Blank check sector(s) | Section 20.6.10 | Table 278 |
| Read Part ID | Section 20.6.11 | Table 279 |
| Read Boot code version | Section 20.6.12 | Table 281 |
| ReadUID | Section 20.6.13 | Table 283 |
| Compare | Section 20.6.14 | Table 282 |

### 20.6.1 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*.

### 20.6.2 C_CAN ISP object directory

**Table 286. 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 286. 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 |

### 20.6.3 Unlock (C_CAN ISP)

Write <Unlock Code> to [0x5000, 0]. Writing an invalid unlock code will return a dedicated abort code.

### 20.6.4 Write to RAM (C_CAN ISP)

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].

### 20.6.5 Read memory (C_CAN ISP)

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].

### 20.6.6 Prepare sectors for write operation (C_CAN ISP)

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.

### 20.6.7 Copy RAM to flash (C_CAN ISP)

Write the parameters into entry [0x5050, 1 to 3]. The write of the number of bytes into [0x5050,3] starts the programming.

See Section 20.5.4 for limitations on the write-to-flash process.

### 20.6.8 Go (C_CAN ISP)

Write the start address into [0x5070, 0]. Then trigger the “start application” command by writing the value 0x1 to [0x1F51, 1].

### 20.6.9 Erase sectors (C_CAN ISP)

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.

### 20.6.10 Blank check sectors (C_CAN ISP)

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.

### 20.6.11 Read PartID (C_CAN ISP)

Read [0x1018, 2]. See Table 280.

### 20.6.12 Read boot code version (C_CAN ISP)

Read [0x1018, 3]

### 20.6.13 Read serial number (C_CAN ISP)

Read [0x5100, 1 to 4]

### 20.6.14 Compare (C_CAN ISP)

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.

### 20.6.15 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 287 shoes the list of abort codes.

In addition, the regular CANopen SDO abort codes for invalid access to OD entries are also supported.

**Table 287. 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 |

### 20.6.16 Differences to fully-compliant CANopen

While the bootloader 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.

## 20.7 IAP commands

For in application programming the IAP routine should be called with a word pointer in register r0 pointing to memory (RAM) containing command code and parameters. Result of the IAP command is returned in the result table pointed to by register r1. The user can reuse the command table for result by passing the same pointer in registers r0 and r1. The parameter table should be 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 64. The number of parameters and results vary according to the IAP command. The maximum number of parameters is 5, passed to the "Copy RAM to FLASH" command. The maximum number of results is 4, returned by the "ReadUID" command. The command handler sends the status code INVALID_COMMAND when an undefined command is received. The IAP routine resides at 0x1FFF 1FF0 location and it is thumb code.

The IAP function could be called in the following way using C.

Define the IAP location entry point. Since the 0th bit of the IAP location is set there will be a change to Thumb instruction set when the program counter branches to this address.

```
#define IAP_LOCATION 0x1fff1ff1
```

Define data structure or pointers to pass IAP command table and result table to the IAP function:

```
unsigned long command[5];
unsigned long result[4];
```

or

```
unsigned long * command;
unsigned long * result;
command=(unsigned long *) 0x……
result= (unsigned long *) 0x……
```

Define pointer to function type, which takes two parameters and returns void. Note the IAP returns the result with the base address of the table residing in R1.

```
typedef void (*IAP)(unsigned int [],unsigned int[]);
IAP iap_entry;
```

Setting function pointer:

```
iap_entry=(IAP) IAP_LOCATION;
```

Whenever you wish to call IAP you could use the following statement.

```
iap_entry (command, result);
```

As per the ARM specification (The ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05) up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively. Additional parameters are passed on the stack. Up to 4 parameters can be returned in the r0, r1, r2 and r3 registers respectively. Additional parameters are returned indirectly via memory. Some of the IAP calls require more than 4 parameters. If the ARM

suggested scheme is used for the parameter passing/returning then it might create problems due to difference in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such risk.

The flash memory is not accessible during a write or erase operation. IAP commands, which results in a flash write/erase operation, use 32 bytes of space in the top portion of the on-chip RAM for execution. The user program should not be use this space if IAP flash programming is permitted in the application.

**Table 288. IAP Command Summary**

| IAP Command | Command Code | Described in |
|---|---|---|
| Prepare sector(s) for write operation | $50_{10}$ | Table 289 |
| Copy RAM to flash | $51_{10}$ | Table 290 |
| Erase sector(s) | $52_{10}$ | Table 291 |
| Blank check sector(s) | $53_{10}$ | Table 292 |
| Read Part ID | $54_{10}$ | Table 293 |
| Read Boot code version | $55_{10}$ | Table 294 |
| Compare | $56_{10}$ | Table 295 |
| Reinvoke ISP | $57_{10}$ | Table 296 |
| Read UID | $58_{10}$ | Table 297 |

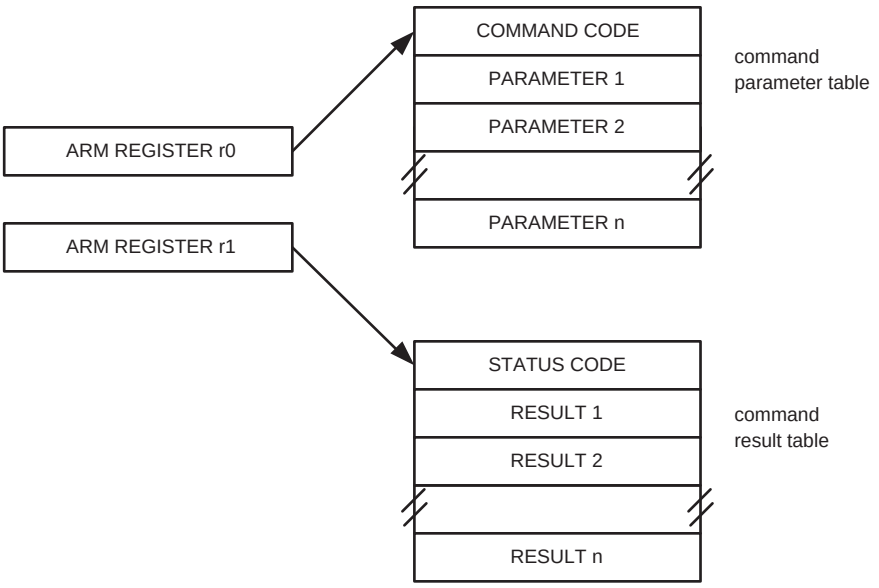



**Fig 64. IAP parameter passing**

## 20.7.1 Prepare sector(s) for write operation (IAP)

This command makes flash write/erase operation a two step process.

**Table 289. IAP Prepare sector(s) for write operation command**

| Command | Prepare sector(s) for write operation |
|---|---|
| Input | **Command code: 5010**<br>**Param0:** Start Sector Number<br>**Param1:** End Sector Number (should be greater than or equal to start sector number). |
| Return Code | CMD_SUCCESS \|<br>BUSY \|<br>INVALID_SECTOR |
| Result | None |
| Description | This command must be executed before executing "Copy RAM to flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot sector can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers. |

### 20.7.2 Copy RAM to flash (IAP)

See Section 20.5.4 for limitations on the write-to-flash process.

**Table 290. IAP Copy RAM to flash command**

| Command | Copy RAM to flash |
|---|---|
| Input | **Command code: 5110**<br>**Param0(DST):** Destination flash address where data bytes are to be written. This address should be a 256 byte boundary.<br>**Param1(SRC):** Source RAM address from which data bytes are to be read. This address should be a word boundary.<br>**Param2:** Number of bytes to be written. Should be 256 \| 512 \| 1024 \| 4096.<br>**Param3:** System Clock Frequency (CCLK) in kHz. |
| Return Code | CMD_SUCCESS \|<br>SRC_ADDR_ERROR (Address not a word boundary) \|<br>DST_ADDR_ERROR (Address not on correct boundary) \|<br>SRC_ADDR_NOT_MAPPED \|<br>DST_ADDR_NOT_MAPPED \|<br>COUNT_ERROR (Byte count is not 256 \| 512 \| 1024 \| 4096) \|<br>SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION \|<br>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. |

### 20.7.3 Erase Sector(s) (IAP)

**Table 291. IAP Erase Sector(s) command**

| Command | Erase Sector(s) |
|---|---|
| Input | **Command code: 5210**<br>**Param0:** Start Sector Number<br>**Param1:** End Sector Number (should be greater than or equal to start sector number).<br>**Param2:** System Clock Frequency (CCLK) in kHz. |
| Return Code | CMD_SUCCESS \|<br>BUSY \|<br>SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION \|<br>INVALID_SECTOR |
| Result | None |
| Description | This command is used to erase a sector or multiple sectors of on-chip flash memory. The boot sector can not be erased by this command. To erase a single sector use the same "Start" and "End" sector numbers. |

### 20.7.4 Blank check sector(s) (IAP)

**Table 292. IAP Blank check sector(s) command**

| Command | Blank check sector(s) |
|---|---|
| Input | **Command code: 5310**<br>**Param0:** Start Sector Number<br>**Param1:** End Sector Number (should be greater than or equal to start sector number). |
| Return Code | CMD_SUCCESS \|<br>BUSY \|<br>SECTOR_NOT_BLANK \|<br>INVALID_SECTOR |
| Result | **Result0:** Offset of the first non blank word location if the Status Code is SECTOR_NOT_BLANK.<br>**Result1:** Contents of non blank word location. |
| Description | This command is used to blank check a sector or multiple sectors of on-chip flash memory. To blank check a single sector use the same "Start" and "End" sector numbers. |

### 20.7.5 Read Part Identification number (IAP)

**Table 293. IAP Read Part Identification command**

| Command | Read part identification number |
|---|---|
| Input | **Command code: 5410**<br>**Parameters:** None |
| Return Code | CMD_SUCCESS \| |
| Result | **Result0:** Part Identification Number. |
| Description | This command is used to read the part identification number. |

### 20.7.6 Read Boot code version number (IAP)

**Table 294. IAP Read Boot Code version number command**

| Command | Read boot code version number |
|---|---|
| Input | **Command code: 5510**<br>**Parameters:** None |
| Return Code | CMD_SUCCESS \| |
| Result | **Result0:** 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)> |
| Description | This command is used to read the boot code version number. |

### 20.7.7 Compare <address1> <address2> <no of bytes> (IAP)

**Table 295. IAP Compare command**

| Command | Compare |
|---|---|
| Input | **Command code: 5610**<br>**Param0(DST):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary.<br>**Param1(SRC):** Starting flash or RAM address of data bytes to be compared. This address should be a word boundary.<br>**Param2:** Number of bytes to be compared; should be a multiple of 4. |
| Return Code | CMD_SUCCESS \|<br>COMPARE_ERROR \|<br>COUNT_ERROR (Byte count is not a multiple of 4) \|<br>ADDR_ERROR \|<br>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.<br>**The result may not be correct when the source or destination includes any of the first 512 bytes starting from address zero. The first 512 bytes can be re-mapped to RAM.** |

### 20.7.8 Reinvoke ISP (IAP)

**Table 296. IAP Reinvoke ISP**

| Command | Compare |
|---|---|
| Input | **Command code: 5710** |
| Return Code | None |
| Result | **None.** |
| Description | This command is used to invoke the bootloader in ISP mode. It maps boot vectors, sets PCLK = CCLK, configures UART pins RXD and TXD, resets counter/timer CT32B1 and resets the U0FDR (see Table 134). This command may be used when a valid user program is present in the internal flash memory and the PIO0_1 pin is not accessible to force the ISP mode. |

### 20.7.9 ReadUID (IAP)

**Table 297. IAP ReadUID command**

| Command | Compare |
|---|---|
| Input | **Command code: 5810** |
| Return Code | CMD_SUCCESS |
| Result | **Result0:** The first 32-bit word (at the lowest address).<br>**Result1:** The second 32-bit word.<br>**Result2:** The third 32-bit word.<br>**Result3:** The fourth 32-bit word. |
| Description | This command is used to read the unique ID. |

### 20.7.10 IAP Status Codes

**Table 298. 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. |

## 20.8 Debug notes

### 20.8.1 Comparing flash images

Depending on the debugger used and the IDE debug settings, the memory that is visible when the debugger connects might be the boot ROM, the internal SRAM, or the flash. To help determine which memory is present in the current debug environment, check the value contained at flash address 0x0000 0004. This address contains the entry point to the code in the ARM Cortex-M0 vector table, which is the bottom of the boot ROM, the internal SRAM, or the flash memory respectively.

**Table 299. Memory mapping in debug mode**

| Memory mapping mode | Memory start address visible at 0x0000 0004 |
|---|---|
| Bootloader mode | 0x1FFF 0000 |
| User flash mode | 0x0000 0000 |
| User SRAM mode | 0x1000 0000 |

### 20.8.2 Serial Wire Debug (SWD) flash programming interface

Debug tools can write parts of the flash image to RAM and then execute the IAP call "Copy RAM to flash" repeatedly with proper offset.

## 20.9 Flash memory access

Depending on the system clock frequency, access to the flash memory can be configured with various access times by writing to the FLASHCFG register at address 0x4003 C010.

**Remark:** Improper setting of this register may result in incorrect operation of the LPC111x/LPC11Cxx flash memory.

**Table 300. Flash configuration register (FLASHCFG, address 0x4003 C010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | FLASHTIM | | Flash memory access time. FLASHTIM +1 is equal to the number of system clocks used for flash access. | 10 |
| | | 00 | 1 system clock flash access time (for system clock frequencies of up to 20 MHz). | |
| | | 01 | 2 system clocks flash access time (for system clock frequencies of up to 40 MHz). | |
| | | 10 | 3 system clocks flash access time (for system clock frequencies of up to 50 MHz). | |
| | | 11 | Reserved. | |
| 31:2 | - | - | Reserved. **User software must not change the value of these bits. Bits 31:2 must be written back exactly as read**. | - |

## 20.10 Flash signature generation

The flash module contains a built-in signature generator. This generator can produce a 128-bit signature from a range of flash memory. A typical usage is to verify the flashed contents against a calculated signature (e.g. during programming).

The address range for generating a signature must be aligned on flash-word boundaries, i.e. 128-bit boundaries. Once started, signature generation completes independently. While signature generation is in progress, the flash memory cannot be accessed for other purposes, and an attempted read will cause a wait state to be asserted until signature generation is complete. Code outside of the flash (e.g. internal RAM) can be executed during signature generation. This can include interrupt services, if the interrupt vector table is re-mapped to memory other than the flash memory. The code that initiates signature generation should also be placed outside of the flash memory.

### 20.10.1 Register description for signature generation

**Table 301. Register overview: FMC (base address 0x4003 C000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| FMSSTART | R/W | 0x020 | Signature start address register | 0 | Table 302 |
| FMSSTOP | R/W | 0x024 | Signature stop-address register | 0 | Table 303 |
| FMSW0 | R | 0x02C | Word 0 [31:0] | - | Table 304 |
| FMSW1 | R | 0x030 | Word 1 [63:32] | - | Table 305 |
| FMSW2 | R | 0x034 | Word 2 [95:64] | - | Table 306 |
| FMSW3 | R | 0x038 | Word 3 [127:96] | - | Table 307 |
| FMSTAT | R | 0xFE0 | Signature generation status register | 0 | Section 20.10.1.3 |
| FMSTATCLR | W | 0xFE8 | Signature generation status clear register | - | Section 20.10.1.4 |

#### 20.10.1.1 Signature generation address and control registers

These registers control automatic signature generation. A signature can be generated for any part of the flash memory contents. The address range to be used for generation is defined by writing the start address to the signature start address register (FMSSTART) and the stop address to the signature stop address register (FMSSTOP. The start and stop addresses must be aligned to 128-bit boundaries and can be derived by dividing the byte address by 16.

Signature generation is started by setting the SIG_START bit in the FMSSTOP register. Setting the SIG_START bit is typically combined with the signature stop address in a single write.

Table 302 and Table 303 show the bit assignments in the FMSSTART and FMSSTOP registers respectively.

**Table 302. Flash Module Signature Start register (FMSSTART - 0x4003 C020) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:17 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 16:0 | START | Signature generation start address (corresponds to AHB byte address bits[20:4]). | 0 |

**Table 303. Flash Module Signature Stop register (FMSSTOP - 0x4003 C024) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:18 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 17 | SIG_START | | Start control bit for signature generation. | 0 |
| | | 0 | Signature generation is stopped | |
| | | 1 | Initiate signature generation | |
| 16:0 | STOP | | BIST stop address divided by 16 (corresponds to AHB byte address [20:4]). | 0 |

### 20.10.1.2 Signature generation result registers

The signature generation result registers return the flash signature produced by the embedded signature generator. The 128-bit signature is reflected by the four registers FMSW0, FMSW1, FMSW2 and FMSW3.

The generated flash signature can be used to verify the flash memory contents. The generated signature can be compared with an expected signature and thus makes saves time and code space. The method for generating the signature is described in Section 20.10.2.

Table 307 show bit assignment of the FMSW0 and FMSW1, FMSW2, FMSW3 registers respectively.

**Table 304. FMSW0 register bit description (FMSW0, address: 0x4003 C02C)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | SW0[31:0] | Word 0 of 128-bit signature (bits 31 to 0). | - |

**Table 305. FMSW1 register bit description (FMSW1, address: 0x4003 C030)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | SW1[63:32] | Word 1 of 128-bit signature (bits 63 to 32). | - |

**Table 306. FMSW2 register bit description (FMSW2, address: 0x4003 C034)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | SW2[95:64] | Word 2 of 128-bit signature (bits 95 to 64). | - |

**Table 307. FMSW3 register bit description (FMSW3, address: 0x4003 40C8)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | SW3[127:96] | Word 3 of 128-bit signature (bits 127 to 96). | - |

#### 20.10.1.3 Flash Module Status register

The read-only FMSTAT register provides a means of determining when signature generation has completed. Completion of signature generation can be checked by polling the SIG_DONE bit in FMSTAT. SIG_DONE should be cleared via the FMSTATCLR register before starting a signature generation operation, otherwise the status might indicate completion of a previous operation.

**Table 308. Flash module Status register (FMSTAT - 0x4003 CFE0) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | SIG_DONE | When 1, a previously started signature generation has completed. See FMSTATCLR register description for clearing this flag. | 0 |
| 1:0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

#### 20.10.1.4 Flash Module Status Clear register

The FMSTATCLR register is used to clear the signature generation completion flag.

**Table 309. Flash Module Status Clear register (FMSTATCLR - 0x0x4003 CFE8) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | SIG_DONE_CLR | Writing a 1 to this bits clears the signature generation completion flag (SIG_DONE) in the FMSTAT register. | 0 |
| 1:0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 20.10.2 Algorithm and procedure for signature generation

#### Signature generation

A signature can be generated for any part of the flash contents. The address range to be used for signature generation is defined by writing the start address to the FMSSTART register, and the stop address to the FMSSTOP register.

The signature generation is started by writing a ‘1’ to FMSSTOP.MISR_START. Starting the signature generation is typically combined with defining the stop address, which is done in another field FMSSTOP.FMSSTOP of the same register.

The time that the signature generation takes is proportional to the address range for which the signature is generated. Reading of the flash memory for signature generation uses a self-timed read mechanism and does not depend on any configurable timing settings for the flash. A safe estimation for the duration of the signature generation is:

Duration = int( (60 / tcy) + 3 ) x (FMSSTOP - FMSSTART + 1)

When signature generation is triggered via software, the duration is in AHB clock cycles, and tcy is the time in ns for one AHB clock. The SIG_DONE bit in FMSTAT can be polled by software to determine when signature generation is complete.

After signature generation, a 128-bit signature can be read from the FMSW0 to FMSW3 registers. The 128-bit signature reflects the corrected data read from the flash. The 128-bit signature reflects flash parity bits and check bit values.

### Content verification

The signature as it is read from the FMSW0 to FMSW3 registers must be equal to the reference signature. The algorithms to derive the reference signature is given in Figure 65.

```
int128 signature = 0
int128 nextSignature
FOR address = flashpage 0 TO address = flashpage max
{
        FOR i = 0 TO 126 {
                nextSignature[i] = flashword[i] XOR signature[i+1] }
        nextSignature[127] = flashword[127] XOR signature[0] XOR signature[2]
                XOR signature[27] XOR signature[29]
        signature = nextSignature
}
return signature
```

**Fig 65. Algorithm for generating a 128-bit signature**

## 21.1 How to read this chapter

The debug functionality is identical for all LPC111x and LPC11Cxx parts.

## 21.2 Features

- Supports ARM Serial Wire Debug mode.
- Direct debug access to all memories, registers, and peripherals.
- No target resources are required for the debugging session.
- Four breakpoints. Four instruction breakpoints that can also be used to remap instruction addresses for code patches. Two data comparators that can be used to remap addresses for patches to literal values.
- Two data watchpoints that can also be used as triggers.

## 21.3 Introduction

Debug functions are integrated into the ARM Cortex-M0. Serial wire debug functions are supported. The ARM Cortex-M0 is configured to support up to four breakpoints and two watchpoints.

## 21.4 Description

Debugging with the LPC111x/LPC11Cxx uses the Serial Wire Debug mode.

## 21.5 Pin description

The tables below indicate the various pin functions related to debug. Some of these functions share pins with other functions which therefore may not be used at the same time.

**Table 310. Serial Wire Debug pin description**

| Pin Name | Type | Description |
|---|---|---|
| SWCLK | Input | **Serial Wire Clock.** This pin is the clock for debug logic when in the Serial Wire Debug mode (SWCLK). This pin is pulled up internally. |
| SWDIO | Input / Output | **Serial wire debug data input/output.** The SWDIO pin is used by an external debug tool to communicate with and control the LPC111x/LPC11Cxx. This pin is pulled up internally. |

## 21.6 Debug notes

### 21.6.1 Debug limitations

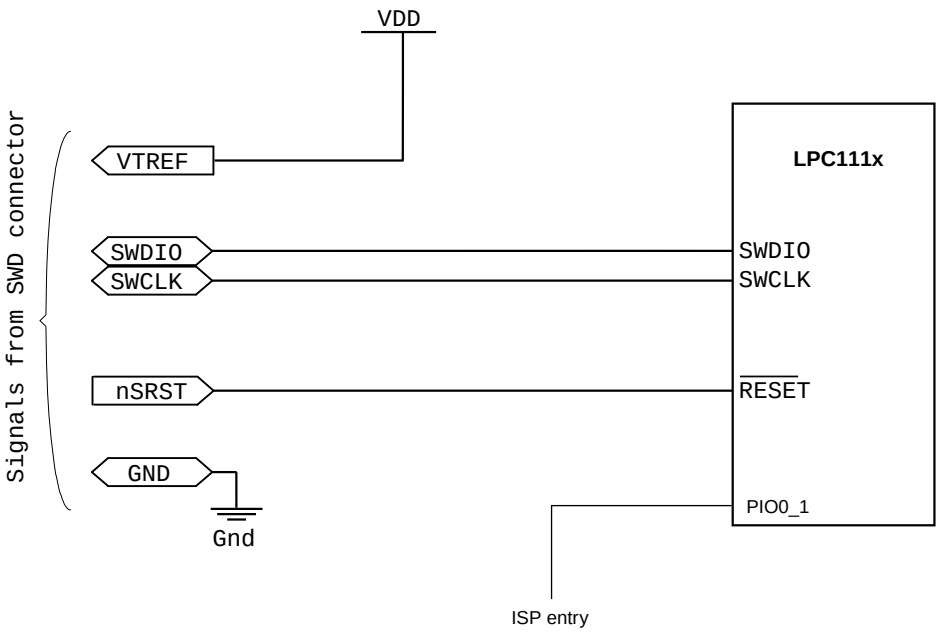
**Important:** The user should be aware of certain limitations during debugging. The most important is that, due to limitations of the ARM Cortex-M0 integration, the LPC111x/LPC11Cxx cannot wake up in the usual manner from Deep-sleep mode. It is recommended not to use this mode during debug.

Another issue is that debug mode changes the way in which reduced power modes work internal to the ARM Cortex-M0 CPU, and this ripples through the entire system. These differences mean that power measurements should not be made while debugging, the results will be higher than during normal operation in an application.

During a debugging session, the System Tick Timer is automatically stopped whenever the CPU is stopped. Other peripherals are not affected.

### 21.6.2 Debug connections

For debugging purposes, it is useful to provide access to the ISP entry pin PIO0_1. This pin can be used to recover the part from configurations which would disable the SWD port such as improper PLL configuration, reconfiguration of SWD pins as ADC inputs, entry into Deep power-down mode out of reset, etc. This pin can be used for other functions such as GPIO, but it should not be held low on power-up or reset.




The VTREF pin on the SWD connector enables the debug connector to match the target voltage.

**Fig 66. Connecting the SWD pins to a standard SWD connector**

## 22.1 Introduction

The following material is using the ARM *Cortex-M0 User Guide*. Minor changes have been made regarding the specific implementation of the Cortex-M0 for the LPC111x/LPC11Cxx.

The ARM Cortex-M0 documentation is also available in Ref. 1 and Ref. 2.

## 22.2 About the Cortex-M0 processor and core peripherals

The Cortex-M0 processor is an entry-level 32-bit ARM Cortex processor designed for a broad range of embedded applications. It offers significant benefits to developers, including:

- a simple architecture that is easy to learn and program
- ultra-low power, energy efficient operation
- excellent code density
- deterministic, high-performance interrupt handling
- upward compatibility with Cortex-M processor family.

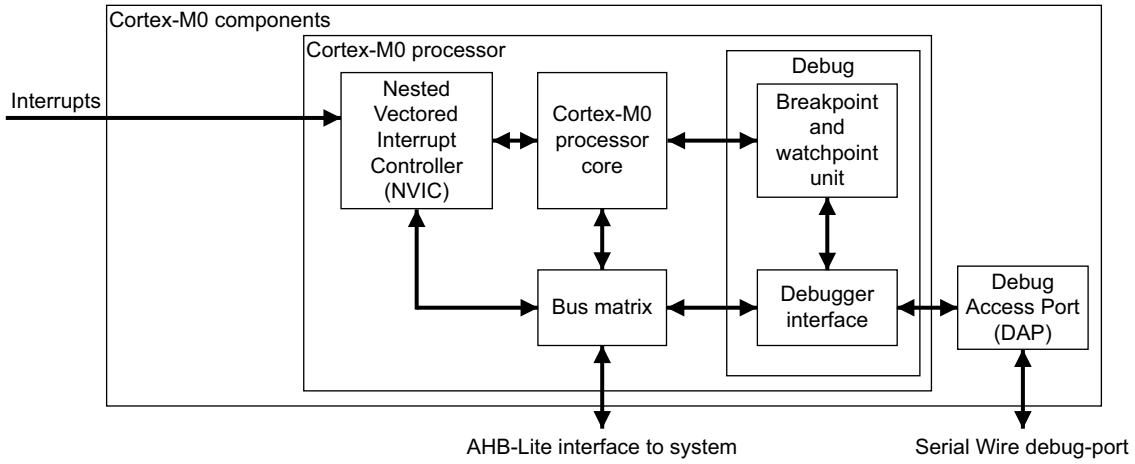



**Fig 67. Cortex-M0 implementation**

The Cortex-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single-cycle multiplier.

The Cortex-M0 processor implements the ARMv6-M architecture, which is based on the 16-bit Thumb instruction set and includes Thumb-2 technology. This provides the exceptional performance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

The Cortex-M0 processor closely integrates a configurable **Nested Vectored Interrupt Controlle**r (NVIC), to deliver industry-leading interrupt performance. The NVIC:

- includes a **non-maskable interrupt** (NMI). The NMI is not implemented on the LPC111x/LPC11Cxx.
- provides zero jitter interrupt option
- provides four interrupt priority levels.

The tight integration of the processor core and NVIC provides fast execution of **interrupt service routines** (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to abandon and restart load-multiple and store-multiple operations. Interrupt handlers do not require any assembler wrapper code, removing any code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a Deep-sleep function that enables the entire device to be rapidly powered down.

### 22.2.1 System-level interface

The Cortex-M0 processor provides a single system-level interface using AMBA technology to provide high speed, low latency memory accesses.

### 22.2.2 Integrated configurable debug

The Cortex-M0 processor implements a complete hardware debug solution, with extensive hardware breakpoint and watchpoint options. This provides high system visibility of the processor, memory and peripherals through a 2-pin **Serial Wire Debug** (SWD) port that is ideal for microcontrollers and other small package devices.

### 22.2.3 Cortex-M0 processor features summary

- high code density with 32-bit performance
- tools and binary upwards compatible with Cortex-M processor family
- integrated ultra low-power sleep modes
- efficient code execution permits slower processor clock or increases sleep mode time
- single-cycle 32-bit hardware multiplier
- zero jitter interrupt handling
- extensive debug capabilities.

### 22.2.4 Cortex-M0 core peripherals

These are:

**NVIC —** The NVIC is an embedded interrupt controller that supports low latency interrupt processing.

**System Control Block —** The **System Control Block** (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

**System timer —** The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

## 22.3 Processor

### 22.3.1 Programmers model

This section describes the Cortex-M0 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and stacks.

#### 22.3.1.1 Processor modes

The processor **modes** are:

**Thread mode —** Used to execute application software. The processor enters Thread mode when it comes out of reset.

**Handler mode —** Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

#### 22.3.1.2 Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the main stack and the process stack, with independent copies of the stack pointer, see Section 22.3.1.3.2.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see Section 22–22.3.1.3.7. In Handler mode, the processor always uses the main stack. The options for processor operations are:

**Table 311. Summary of processor mode and stack use options**

| Processor mode | Used to execute | Stack used |
|---|---|---|
| Thread | Applications | Main stack or process stack<br>See Section 22–22.3.1.3.7 |
| Handler | Exception handlers | Main stack |

#### 22.3.1.3 Core registers

The processor core registers are:

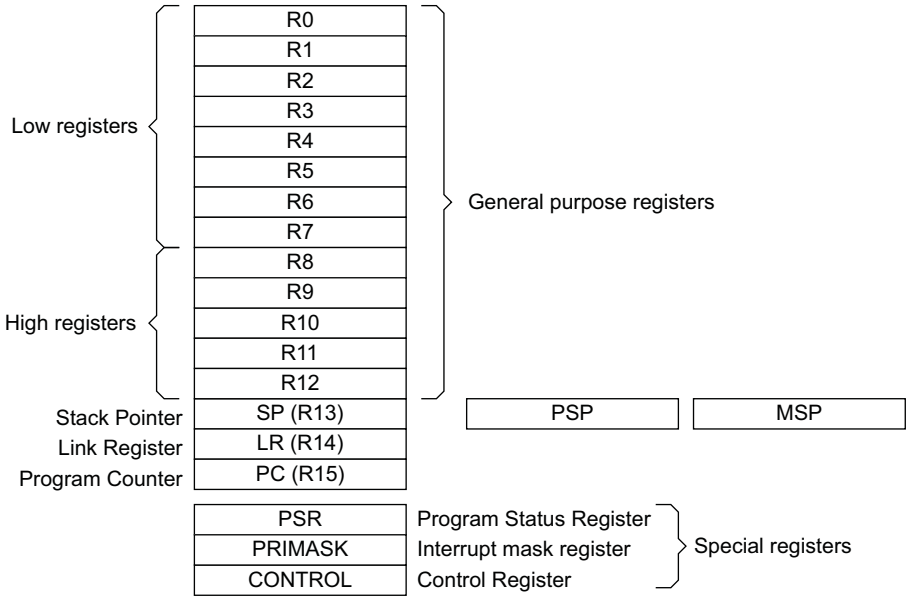



**Fig 68. Processor core register set**

**Table 312. Core register set summary**

| Name | Type[1] | Reset value | Description |
|---|---|---|---|
| R0-R12 | RW | Unknown | Section 22–22.3.1.3.1 |
| MSP | RW | See description | Section 22–22.3.1.3.2 |
| PSP | RW | Unknown | Section 22–22.3.1.3.2 |
| LR | RW | Unknown | Section 22–22.3.1.3.3 |
| PC | RW | See description | Section 22–22.3.1.3.4 |
| PSR | RW | Unknown[2] | Table 22–313 |
| APSR | RW | Unknown | Table 22–314 |
| IPSR | RO | 0x00000000 | Table 315 |
| EPSR | RO | Unknown[2] | Table 22–316 |
| PRIMASK | RW | 0x00000000 | Table 22–317 |
| CONTROL | RW | 0x00000000 | Table 22–318 |

[1] Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

[2] Bit[24] is the T-bit and is loaded from bit[0] of the reset vector.

### 22.3.1.3.1 General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

### 22.3.1.3.2 Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = **Main Stack Pointer** (MSP). This is the reset value.
- 1 = **Process Stack Pointer** (PSP).

On reset, the processor loads the MSP with the value from address 0x00000000.

#### 22.3.1.3.3 Link Register

The **Link Register** (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the LR value is Unknown.

#### 22.3.1.3.4 Program Counter

The **Program Counter** (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

#### 22.3.1.3.5 Program Status Register

The **Program Status Register** (PSR) combines:

- **Application Program Status Register** (APSR)
- **Interrupt Program Status Register** (IPSR)
- **Execution Program Status Register** (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The PSR bit assignments are:

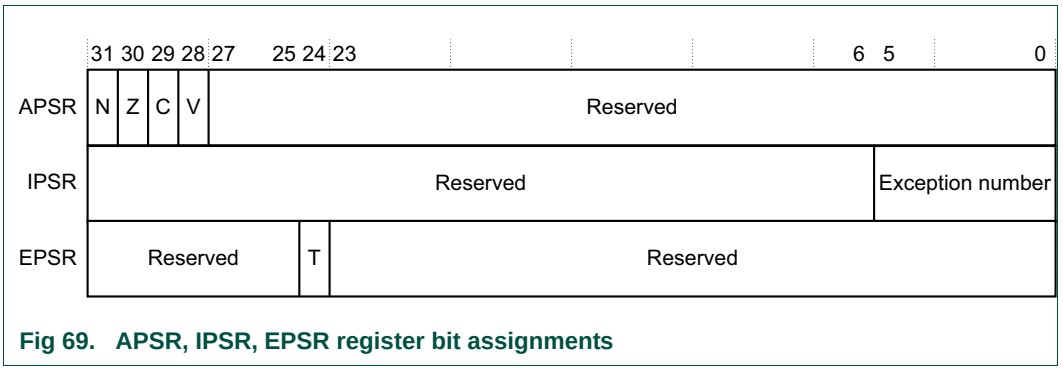



**Fig 69. APSR, IPSR, EPSR register bit assignments**

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR using APSR with the MSR instruction.

The PSR combinations and attributes are:

**Table 313. PSR register combinations**

| Register | Type | Combination |
|---|---|---|
| PSR | RW[1][2] | APSR, EPSR, and IPSR |
| IEPSR | RO | EPSR and IPSR |
| IAPSR | RW[1] | APSR and IPSR |
| EAPSR | RW[2] | APSR and EPSR |

[1] The processor ignores writes to the IPSR bits.

[2] Reads of the EPSR bits return zero, and the processor ignores writes to the these bits

See the instruction descriptions Section 22–22.4.7.6 and Section 22–22.4.7.7 for more information about how to access the program status registers.

**Application Program Status Register:** The APSR contains the current state of the condition flags, from previous instruction executions. See the register summary in Table 22–312 for its attributes. The bit assignments are:

**Table 314. APSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | N | Negative flag |
| [30] | Z | Zero flag |
| [29] | C | Carry or borrow flag |
| [28] | V | Overflow flag |
| [27:0] | - | Reserved |

See Section 22.4.4.1.4 for more information about the APSR negative, zero, carry or borrow, and overflow flags.

**Interrupt Program Status Register:** The IPSR contains the exception number of the current **Interrupt Service Routine** (ISR). See the register summary in Table 22–312 for its attributes. The bit assignments are:

**Table 315. IPSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:6] | - | Reserved |
| [5:0] | Exception number | This is the number of the current exception:<br>0 = Thread mode<br>1 = Reserved<br>2 = NMI<br>3 = HardFault<br>4-10 = Reserved<br>11 = SVCall<br>12, 13 = Reserved<br>14 = PendSV<br>15 = SysTick<br>16 = IRQ0<br>.<br>.<br>.<br>47 = IRQ31<br>48-63 = Reserved.<br>see Section 22–22.3.3.2 for more information. |

**Execution Program Status Register:** The EPSR contains the Thumb state bit.

See the register summary in Table 22–312 for the EPSR attributes. The bit assignments are:

**Table 316. EPSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:25] | - | Reserved |
| [24] | T | Thumb state bit |
| [23:0] | - | Reserved |

Attempts by application software to read the EPSR directly using the `MRS` instruction always return zero. Attempts to write the EPSR using the `MSR` instruction are ignored. Fault handlers can examine the EPSR value in the stacked PSR to determine the cause of the fault. See Section 22–22.3.3.6. The following can clear the T bit to 0:

- instructions `BLX`, `BX` and `POP{PC}`
- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry.

Attempting to execute instructions when the T bit is 0 results in a HardFault or lockup. See Section 22–22.3.4.1 for more information.

**Interruptible-restartable instructions:** The interruptible-restartable instructions are `LDM` and `STM`. When an interrupt occurs during the execution of one of these instructions, the processor abandons execution of the instruction.

After servicing the interrupt, the processor restarts execution of the instruction from the beginning.

#### 22.3.1.3.6 Exception mask register

The exception mask register disables the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks or code sequences requiring atomicity.

To disable or re-enable exceptions, use the `MSR` and `MRS` instructions, or the `CPS` instruction, to change the value of PRIMASK. See Section 22–22.4.7.6, Section 22–22.4.7.7, and Section 22–22.4.7.2 for more information.

**Priority Mask Register:** The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in Table 22–312 for its attributes. The bit assignments are:

**Table 317. PRIMASK register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:1] | - | Reserved |
| [0] | PRIMASK | 0 = no effect<br>1 = prevents the activation of all exceptions with configurable priority. |

#### 22.3.1.3.7 CONTROL register

The CONTROL register controls the stack used when the processor is in Thread mode. See the register summary in Table 22–312 for its attributes. The bit assignments are:

**Table 318. CONTROL register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:2] | - | Reserved |
| [1] | Active stack pointer | Defines the current stack:<br>0 = MSP is the current stack pointer<br>1 = PSP is the current stack pointer.<br>In Handler mode this bit reads as zero and ignores writes. |
| [0] | - | Reserved. |

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms update the CONTROL register.

In an OS environment, it is recommended that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, use the `MSR` instruction to set the Active stack pointer bit to 1, see Section 22–22.4.7.6.

**Remark:** When changing the stack pointer, software must use an `ISB` instruction immediately after the `MSR` instruction. This ensures that instructions after the ISB execute using the new stack pointer. See Section 22–22.4.7.5.

### 22.3.1.4 Exceptions and interrupts

The Cortex-M0 processor supports interrupts and system exceptions. The processor and the **Nested Vectored Interrupt Controller** (NVIC) prioritize and handle all exceptions. An interrupt or exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See Section 22–22.3.3.6.1 and Section 22–22.3.3.6.2 for more information.

The NVIC registers control interrupt handling. See Section 22–22.5.2 for more information.

### 22.3.1.5 Data types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- manages all data memory accesses as little-endian. Instruction memory and **Private Peripheral Bus** (PPB) accesses are always little-endian. See Section 22–22.3.2.1 for more information.

### 22.3.1.6 The Cortex Microcontroller Software Interface Standard

ARM provides the **Cortex Microcontroller Software Interface Standard** (CMSIS) for programming Cortex-M0 microcontrollers. The CMSIS is an integrated part of the device driver library.

For a Cortex-M0 microcontroller system, CMSIS defines:

- a common way to:
  - access peripheral registers
  - define exception vectors
- the names of:
  - the registers of the core peripherals
  - the core exception vectors
- a device-independent interface for RTOS kernels.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M0 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

The CMSIS simplifies software development by enabling the reuse of template code, and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

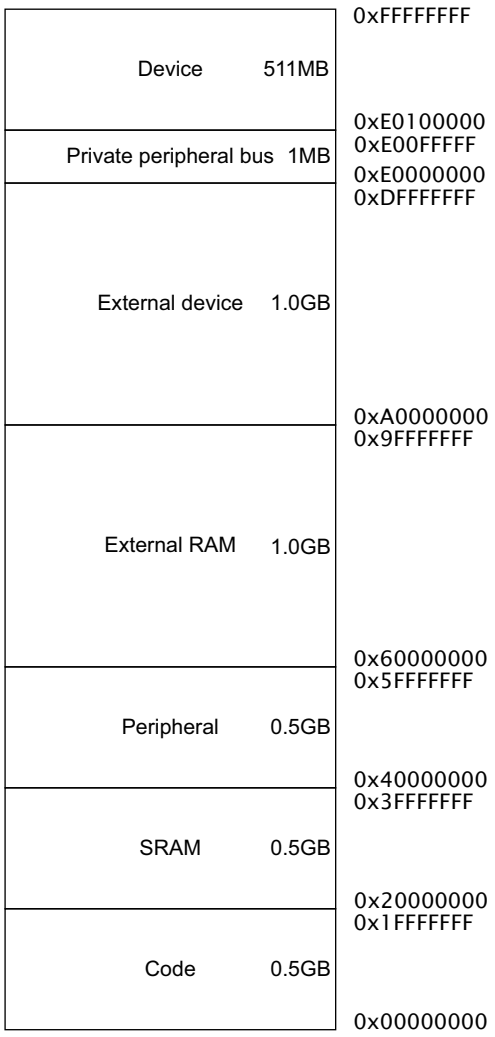
**Remark:** This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- Section 22.3.5.3 “Power management programming hints”
- Section 22.4.2 “Intrinsic functions”
- Section 22.5.2.1 “Accessing the Cortex-M0 NVIC registers using CMSIS”
- Section 22.5.2.8.1 “NVIC programming hints”.

## 22.3.2 Memory model

This section describes the processor memory map and the behavior of memory accesses. The processor has a fixed memory map that provides up to 4GB of addressable memory. The memory map is:




See Figure 2 for the LPC111x/LPC11Cxx specific implementation of the memory map. SRAM and code locations are different on the LPC111x/LPC11Cxx.

**Fig 70. Generic ARM Cortex-M0 memory map**

The processor reserves regions of the **Private peripheral bus** (PPB) address range for core peripheral registers, see Section 22–22.2.

#### 22.3.2.1 Memory regions, types and attributes

The memory map is split into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

**Normal —** The processor can re-order transactions for efficiency, or perform speculative reads.

**Device —** The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.

**Strongly-ordered —** The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include.

**Execute Never (XN) —** Means the processor prevents instruction accesses. A HardFault exception is generated on executing an instruction fetched from an XN region of memory.

### 22.3.2.2 Memory system ordering of memory accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing any re-ordering does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions, see Section 22–22.3.2.4.

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses caused by two instructions is:

| A1 \ A2 | Normal access | Device access | | Strongly-ordered access |
|---|---|---|---|---|
| | | Non-shareable | Shareable | |
| Normal access | - | - | - | - |
| Device access, non-shareable | - | < | - | < |
| Device access, shareable | - | - | < | < |
| Strongly-ordered access | - | < | < | < |

**Fig 71. Memory ordering restrictions**

Where:

**- —** Means that the memory system does not guarantee the ordering of the accesses.

**< —** Means that accesses are observed in program order, that is, A1 is always observed before A2.

### 22.3.2.3 Behavior of memory accesses

The behavior of accesses to each region in the memory map is:

**Table 319. Memory access behavior**

| Address range | Memory region | Memory type[1] | XN[1] | Description |
|---|---|---|---|---|
| 0x00000000-0x1FFFFFFF | Code | Normal | - | Executable region for program code. You can also put data here. |
| 0x20000000-0x3FFFFFFF | SRAM | Normal | - | Executable region for data. You can also put code here. |
| 0x40000000-0x5FFFFFFF | Peripheral | Device | XN | External device memory. |
| 0x60000000-0x9FFFFFFF | External RAM | Normal | - | Executable region for data. |
| 0xA0000000-0xDFFFFFFF | External device | Device | XN | External device memory. |
| 0xE0000000-0xE00FFFFF | Private Peripheral Bus | Strongly-ordered | XN | This region includes the NVIC, System timer, and System Control Block. Only word accesses can be used in this region. |
| 0xE0100000-0xFFFFFFFF | Device | Device | XN | Vendor specific. |

[1] See Section 22–22.3.2.1 for more information.

The Code, SRAM, and external RAM regions can hold programs.

### 22.3.2.4 Software ordering of memory accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence
- memory or devices in the memory map might have different wait states
- some memory accesses are buffered or speculative.

Section 22–22.3.2.2 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

**DMB — The Data Memory Barrier** (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See Section 22–22.4.7.3.

**DSB — The Data Synchronization Barrier** (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See Section 22–22.4.7.4.

**ISB — The Instruction Synchronization Barrier** (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See Section 22–22.4.7.5.

The following are examples of using memory barrier instructions:

**Vector table —** If the program changes an entry in the vector table, and then enables the corresponding exception, use a `DMB` instruction between the operations. This ensures that if the exception is taken immediately after being enabled the processor uses the new exception vector.

**Self-modifying code —** If a program contains self-modifying code, use an `ISB` instruction immediately after the code modification in the program. This ensures subsequent instruction execution uses the updated program.

**Memory map switching —** If the system contains a memory map switching mechanism, use a `DSB` instruction after switching the memory map. This ensures subsequent instruction execution uses the updated memory map.

Memory accesses to Strongly-ordered memory, such as the System Control Block, do not require the use of `DMB` instructions.

The processor preserves transaction order relative to all other transactions.

#### 22.3.2.5 Memory endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. Section 22–22.3.2.5.1 describes how words of data are stored in memory.

##### 22.3.2.5.1 Little-endian format

In little-endian format, the processor stores the **least significant byte** (lsbyte) of a word at the lowest-numbered byte, and the **most significant byte** (msbyte) at the highest-numbered byte. For example:

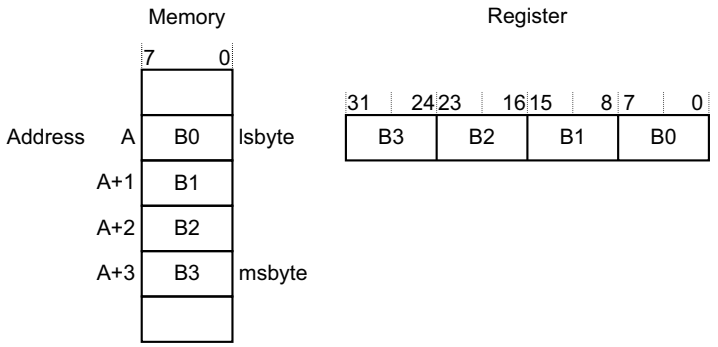



**Fig 72. Little-endian format**

### 22.3.3 Exception model

This section describes the exception model.

#### 22.3.3.1 Exception states

Each exception is in one of the following states:

**Inactive —** The exception is not active and not pending.

**Pending —** The exception is waiting to be serviced by the processor.

An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.

**Active —** An exception that is being serviced by the processor but has not completed.

An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.

**Active and pending —** The exception is being serviced by the processor and there is a pending exception from the same source.

### 22.3.3.2 Exception types

The exception types are:

**Remark:** The NMI is not implemented on the LPC111x/LPC11Cxx.

**Reset —** Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts in Thread mode.

**NMI —** A **NonMaskable Interrupt** (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of −2. NMIs cannot be:

- masked or prevented from activation by any other exception
- preempted by any exception other than Reset.

**HardFault —** A HardFault is an exception that occurs because of an error during normal or exception processing. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

**SVCall —** A **supervisor call** (SVC) is an exception that is triggered by the `SVC` instruction. In an OS environment, applications can use `SVC` instructions to access OS kernel functions and device drivers.

**PendSV —** PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

**SysTick —** A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

**Interrupt (IRQ) —** An interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

**Table 320. Properties of different exception types**

| Exception number[1] | IRQ number[1] | Exception type | Priority | Vector address[2] |
|---|---|---|---|---|
| 1 | - | Reset | -3, the highest | `0x00000004` |
| 2 | -14 | NMI | -2 | `0x00000008` |
| 3 | -13 | HardFault | -1 | `0x0000000C` |
| 4-10 | - | Reserved | - | - |
| 11 | -5 | SVCall | Configurable[3] | `0x0000002C` |

**Table 320. Properties of different exception types**

| Exception number[1] | IRQ number[1] | Exception type | Priority | Vector address[2] |
|---|---|---|---|---|
| 12-13 | - | Reserved | - | - |
| 14 | -2 | PendSV | Configurable[3] | |
| 15 | -1 | SysTick | Configurable[3] | |
| 16 and above | 0 and above | Interrupt (IRQ) | Configurable[3] | and above[4] |

[1] To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see Table 22–315.

[2] See Section 22.3.3.4 for more information.

[3] See Section 22–22.5.2.6.

[4] Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute additional instructions between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 22–320 shows as having configurable priority, see Section 22–22.5.2.3.

For more information about HardFaults, see Section 22–22.3.4.

### 22.3.3.3 Exception handlers

The processor handles exceptions using:

**Interrupt Service Routines (ISRs) —** Interrupts IRQ0 to IRQ31 are the exceptions handled by ISRs.

**Fault handler —** HardFault is the only exception handled by the fault handler.

**System handlers —** NMI, PendSV, SVCall SysTick, and HardFault are all system exceptions handled by system handlers.

### 22.3.3.4 Vector table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. Figure 22–73 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is written in Thumb code.

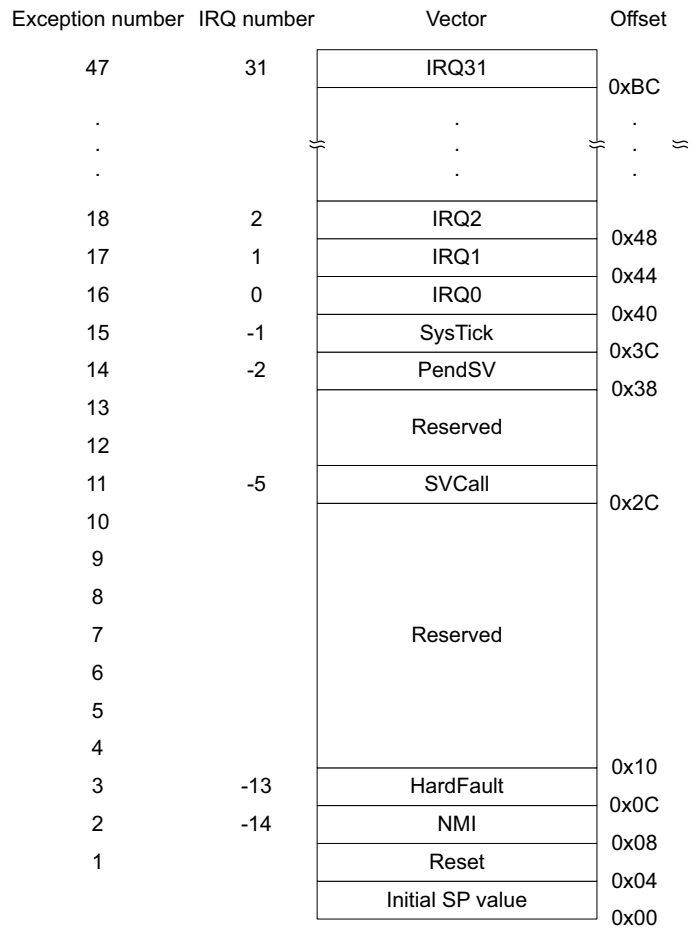



**Fig 73. Vector table**

The vector table is fixed at address .

### 22.3.3.5 Exception priorities

As Table 22–320 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- Section 22–22.5.3.7
- Section 22–22.5.2.6.

**Remark:** Configurable priority values are in the range 0-3. The Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

Assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

#### 22.3.3.6 Exception entry and return

Descriptions of exception handling use the following terms:

**Preemption —** When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled.

When one exception preempts another, the exceptions are called nested exceptions. See Section 22–22.3.3.6.1 for more information.

**Return —** This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Section 22–22.3.3.6.2 for more information.

**Tail-chaining —** This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.

**Late-arriving —** This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved would be the same for both exceptions. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

##### 22.3.3.6.1 Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the exception being handled.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has greater priority than any limit set by the mask register, see Section 22–22.3.1.3.6. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as **stacking** and the structure of eight data words is referred as a **stack frame**. The stack frame contains the following information:

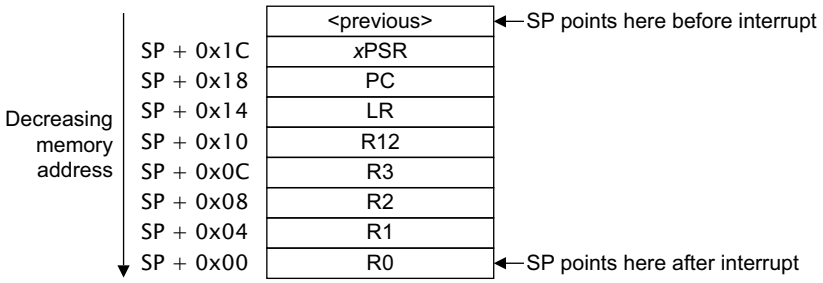



**Fig 74. Exception entry stack contents**

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The stack frame is aligned to a double-word address.

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

The processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

#### 22.3.3.6.2 Exception return

Exception return occurs when the processor is in Handler mode and execution of one of the following instructions attempts to set the PC to an EXC_RETURN value:

- a instruction that loads the PC
- a instruction using any register.

The processor saves an EXC_RETURN value to the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. Bits[31:4] of an EXC_RETURN value are `0xFFFFFFF`. When the processor loads a value matching this pattern to the PC it detects that the operation is a

not a normal branch operation and, instead, that the exception is complete. Therefore, it starts the exception return sequence. Bits[3:0] of the EXC_RETURN value indicate the required return stack and processor mode, as Table 22–321 shows.

**Table 321. Exception return behavior**

| EXC_RETURN | Description |
|---|---|
| 0xFFFFFFF1 | Return to Handler mode.<br>Exception return gets state from the main stack.<br>Execution uses MSP after return. |
| 0xFFFFFFF9 | Return to Thread mode.<br>Exception return gets state from MSP.<br>Execution uses MSP after return. |
| 0xFFFFFFFD | Return to Thread mode.<br>Exception return gets state from PSP.<br>Execution uses PSP after return. |
| All other values | Reserved. |

### 22.3.4 Fault handling

Faults are a subset of exceptions, see Section 22–22.3.3. All faults result in the HardFault exception being taken or cause lockup if they occur in the NMI or HardFault handler. The faults are:

- execution of an SVC instruction at a priority equal or higher than SVCall
- execution of a BKPT instruction without a debugger attached
- a system-generated bus error on a load or store
- execution of an instruction from an XN memory address
- execution of an instruction from a location for which the system generates a bus fault
- a system-generated bus error on a vector fetch
- execution of an Undefined instruction
- execution of an instruction when not in Thumb-State as a result of the T-bit being previously cleared to 0
- an attempted load or store to an unaligned address.

**Remark:** Only Reset and NMI can preempt the fixed priority HardFault handler. A HardFault can preempt any exception other than Reset, NMI, or another hard fault.

#### 22.3.4.1 Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers, or if the system generates a bus error when unstacking the PSR on an exception return using the MSP. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until one of the following occurs:

- it is reset
- a debugger halts it
- an NMI occurs and the current lockup is in the HardFault handler.

**Remark:** If lockup state occurs in the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

### 22.3.5 Power management

The Cortex-M0 processor sleep modes reduce power consumption:

- a sleep mode, that stops the processor clock
- a Deep-sleep mode.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see Section 22–22.5.3.5.

This section describes the mechanisms for entering sleep mode and the conditions for waking up from sleep mode.

#### 22.3.5.1 Entering sleep mode

This section describes the mechanisms software can use to put the processor into sleep mode.

The system can generate spurious wake-up events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back in to sleep mode.

##### 22.3.5.1.1 Wait for interrupt

The Wait For Interrupt instruction, `WFI`, causes immediate entry to sleep mode. When the processor executes a `WFI` instruction it stops executing instructions and enters sleep mode. See Section 22–22.4.7.12 for more information.

##### 22.3.5.1.2 Wait for event

**Remark:** The WFE instruction is not implemented on the LPC111x/LPC11Cxx.

The Wait For Event instruction, `WFE`, causes entry to sleep mode conditional on the value of a one-bit event register. When the processor executes a `WFE` instruction, it checks the value of the event register:

**0 —** The processor stops executing instructions and enters sleep mode

**1 —** The processor sets the register to zero and continues executing instructions without entering sleep mode.

See Section 22–22.4.7.11 for more information.

If the event register is 1, this indicates that the processor must not enter sleep mode on execution of a `WFE` instruction. Typically, this is because of the assertion of an external event, or because another processor in the system has executed a `SEV` instruction, see Section 22–22.4.7.9. Software cannot access this register directly.

#### 22.3.5.1.3 Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of an exception handler and returns to Thread mode it immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an interrupt occurs.

### 22.3.5.2 Wake-up from sleep mode

The conditions for the processor to wake-up depend on the mechanism that caused it to enter sleep mode.

#### 22.3.5.2.1 Wake-up from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK, see Section 22–22.3.1.3.6.

#### 22.3.5.2.2 Wake-up from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry
- in a multiprocessor system, another processor in the system executes a `SEV` instruction.

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see Section 22–22.5.3.5.

### 22.3.5.3 Power management programming hints

ISO/IEC C cannot directly generate the `WFI`, `WFE`, and `SEV` instructions. The CMSIS provides the following intrinsic functions for these instructions:

```
void __WFE(void) // Wait for Event

void __WFI(void) // Wait for Interrupt

void __SEV(void) // Send Event
```

## 22.4 Instruction set

### 22.4.1 Instruction set summary

The processor implements a version of the Thumb instruction set. Table 322 lists the supported instructions.

**Remark:** In Table 322

- angle brackets, <>, enclose alternative forms of the operand
- braces, {}, enclose optional operands and mnemonic parts
- the Operands column is not exhaustive.

For more information on the instructions and operands, see the instruction descriptions.

**Table 322. Cortex-M0 instructions**

| Mnemonic | Operands | Brief description | Flags | Reference |
|---|---|---|---|---|
| ADCS | *{Rd,} Rn, Rm* | Add with Carry | N,Z,C,V | Section 22–22.4.5.1 |
| ADD{S} | *{Rd,} Rn, <Rm\|#imm>* | Add | N,Z,C,V | Section 22–22.4.5.1 |
| ADR | *Rd, label* | PC-relative Address to Register | - | Section 22–22.4.4.1 |
| ANDS | *{Rd,} Rn, Rm* | Bitwise AND | N,Z | Section 22–22.4.5.1 |
| ASRS | *{Rd,} Rm, <Rs\|#imm>* | Arithmetic Shift Right | N,Z,C | Section 22–22.4.5.3 |
| B{cc} | *label* | Branch {conditionally} | - | Section 22–22.4.6.1 |
| BICS | *{Rd,} Rn, Rm* | Bit Clear | N,Z | Section 22–22.4.5.2 |
| BKPT | *#imm* | Breakpoint | - | Section 22–22.4.7.1 |
| BL | *label* | Branch with Link | - | Section 22–22.4.6.1 |
| BLX | *Rm* | Branch indirect with Link | - | Section 22–22.4.6.1 |
| BX | *Rm* | Branch indirect | - | Section 22–22.4.6.1 |
| CMN | *Rn, Rm* | Compare Negative | N,Z,C,V | Section 22–22.4.5.4 |
| CMP | *Rn, <Rm\|#imm>* | Compare | N,Z,C,V | Section 22–22.4.5.4 |
| CPSID | *i* | Change Processor State, Disable Interrupts | - | Section 22–22.4.7.2 |
| CPSIE | *i* | Change Processor State, Enable Interrupts | - | Section 22–22.4.7.2 |
| DMB | - | Data Memory Barrier | - | Section 22–22.4.7.3 |
| DSB | - | Data Synchronization Barrier | - | Section 22–22.4.7.4 |
| EORS | *{Rd,} Rn, Rm* | Exclusive OR | N,Z | Section 22–22.4.5.2 |
| ISB | - | Instruction Synchronization Barrier | - | Section 22–22.4.7.5 |
| LDM | *Rn{!}, reglist* | Load Multiple registers, increment after | - | Section 22–22.4.4.5 |
| LDR | *Rt, label* | Load Register from PC-relative address | - | Section 22–22.4.4 |
| LDR | *Rt, [Rn, <Rm\|#imm>]* | Load Register with word | - | Section 22–22.4.4 |
| LDRB | *Rt, [Rn, <Rm\|#imm>]* | Load Register with byte | - | Section 22–22.4.4 |
| LDRH | *Rt, [Rn, <Rm\|#imm>]* | Load Register with halfword | - | Section 22–22.4.4 |
| LDRSB | *Rt, [Rn, <Rm\|#imm>]* | Load Register with signed byte | - | Section 22–22.4.4 |
| LDRSH | *Rt, [Rn, <Rm\|#imm>]* | Load Register with signed halfword | - | Section 22–22.4.4 |
| LSLS | *{Rd,} Rn, <Rs\|#imm>* | Logical Shift Left | N,Z,C | Section 22–22.4.5.3 |
| U | *{Rd,} Rn, <Rs\|#imm>* | Logical Shift Right | N,Z,C | Section 22–22.4.5.3 |
| MOV{S} | *Rd, Rm* | Move | N,Z | Section 22–22.4.5.5 |
| MRS | *Rd, spec_reg* | Move to general register from special register | - | Section 22–22.4.7.6 |
| MSR | *spec_reg, Rm* | Move to special register from general register | N,Z,C,V | Section 22–22.4.7.7 |
| MULS | *Rd, Rn, Rm* | Multiply, 32-bit result | N,Z | Section 22–22.4.5.6 |
| MVNS | *Rd, Rm* | Bitwise NOT | N,Z | Section 22–22.4.5.5 |

**Table 322. Cortex-M0 instructions**

| Mnemonic | Operands | Brief description | Flags | Reference |
|---|---|---|---|---|
| NOP | - | No Operation | - | Section 22–22.4.7.8 |
| ORRS | *{Rd,} Rn, Rm* | Logical OR | N,Z | Section 22–22.4.5.2 |
| POP | *reglist* | Pop registers from stack | - | Section 22–22.4.4.6 |
| PUSH | *reglist* | Push registers onto stack | - | Section 22–22.4.4.6 |
| REV | *Rd, Rm* | Byte-Reverse word | - | Section 22–22.4.5.7 |
| REV16 | *Rd, Rm* | Byte-Reverse packed halfwords | - | Section 22–22.4.5.7 |
| REVSH | *Rd, Rm* | Byte-Reverse signed halfword | - | Section 22–22.4.5.7 |
| RORS | *{Rd,} Rn, Rs* | Rotate Right | N,Z,C | Section 22–22.4.5.3 |
| RSBS | *{Rd,} Rn, #0* | Reverse Subtract | N,Z,C,V | Section 22–22.4.5.1 |
| SBCS | *{Rd,} Rn, Rm* | Subtract with Carry | N,Z,C,V | Section 22–22.4.5.1 |
| SEV | - | Send Event | - | Section 22–22.4.7.9 |
| STM | *Rn!, reglist* | Store Multiple registers, increment after | - | Section 22–22.4.4.5 |
| STR | *Rt, [Rn, <Rm\|#imm>]* | Store Register as word | - | Section 22–22.4.4 |
| STRB | *Rt, [Rn, <Rm\|#imm>]* | Store Register as byte | - | Section 22–22.4.4 |
| STRH | *Rt, [Rn, <Rm\|#imm>]* | Store Register as halfword | - | Section 22–22.4.4 |
| SUB{S} | *{Rd,} Rn, <Rm\|#imm>* | Subtract | N,Z,C,V | Section 22–22.4.5.1 |
| SVC | *#imm* | Supervisor Call | - | Section 22–22.4.7.10 |
| SXTB | *Rd, Rm* | Sign extend byte | - | Section 22–22.4.5.8 |
| SXTH | *Rd, Rm* | Sign extend halfword | - | Section 22–22.4.5.8 |
| TST | *Rn, Rm* | Logical AND based test | N,Z | Section 22–22.4.5.9 |
| UXTB | *Rd, Rm* | Zero extend a byte | - | Section 22–22.4.5.8 |
| UXTH | *Rd, Rm* | Zero extend a halfword | - | Section 22–22.4.5.8 |
| WFE | - | Wait For Event | - | Section 22–22.4.7.11 |
| WFI | - | Wait For Interrupt | - | Section 22–22.4.7.12 |

## 22.4.2 Intrinsic functions

ISO/IEC C code cannot directly access some Cortex-M0 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access the relevant instruction.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

**Table 323. CMSIS intrinsic functions to generate some Cortex-M0 instructions**

| Instruction | CMSIS intrinsic function |
|---|---|
| CPSIE i | void __enable_irq(void) |
| CPSID i | void __disable_irq(void) |
| ISB | void __ISB(void) |
| DSB | void __DSB(void) |
| DMB | void __DMB(void) |
| NOP | void __NOP(void) |

**Table 323. CMSIS intrinsic functions to generate some Cortex-M0 instructions**

| Instruction | CMSIS intrinsic function |
|---|---|
| REV | uint32_t __REV(uint32_t int value) |
| REV16 | uint32_t __REV16(uint32_t int value) |
| REVSH | uint32_t __REVSH(uint32_t int value) |
| SEV | void __SEV(void) |
| WFE | void __WFE(void) |
| WFI | void __WFI(void) |

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

**Table 324. insic functions to access the special registers**

| Special register | Access | CMSIS function |
|---|---|---|
| PRIMASK | Read | uint32_t __get_PRIMASK (void) |
| | Write | void __set_PRIMASK (uint32_t value) |
| CONTROL | Read | uint32_t __get_CONTROL (void) |
| | Write | void __set_CONTROL (uint32_t value) |
| MSP | Read | uint32_t __get_MSP (void) |
| | Write | void __set_MSP (uint32_t TopOfMainStack) |
| PSP | Read | uint32_t __get_PSP (void) |
| | Write | void __set_PSP (uint32_t TopOfProcStack) |

### 22.4.3 About the instruction descriptions

The following sections give more information about using the instructions:

- Section 22.4.3.1 “Operands”
- Section 22.4.3.2 “Restrictions when using PC or SP”
- Section 22.4.3.3 “Shift Operations”
- Section 22.4.3.4 “Address alignment”
- Section 22.4.3.5 “PC-relative expressions”
- Section 22.4.3.6 “Conditional execution”.

#### 22.4.3.1 Operands

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the other operands.

#### 22.4.3.2 Restrictions when using PC or SP

Many instructions are unable to use, or have restrictions on whether you can use, the **Program Counter** (PC) or **Stack Pointer** (SP) for the operands or destination register. See instruction descriptions for more information.

**Remark:** When you update the PC with a BX, BLX, or POP instruction, bit[0] of any address must be 1 for correct execution. This is because this bit indicates the destination instruction set, and the Cortex-M0 processor only supports Thumb instructions. When a BL or BLX instruction writes the value of bit[0] into the LR it is automatically assigned the value 1.

### 22.4.3.3 Shift Operations

Register shift operations move the bits in a register left or right by a specified number of bits, the **shift length**. Register shift can be performed directly by the instructions ASR, LSR, LSL, and ROR and the result is written to a destination register.The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions, *Rm* is the register containing the value to be shifted, and *n* is the shift length.

#### 22.4.3.3.1 ASR

Arithmetic shift right by *n* bits moves the left-hand 32 -*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32 -*n* bits of the result, and it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 22–75.

You can use the ASR operation to divide the signed value in the register *Rm* by $2^n$, with the result being rounded towards negative-infinity.

When the instruction is ASRS the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

**Remark:**

- If *n* is 32 or more, then all the bits in the result are set to the value of bit[31] of *Rm*.
- If *n* is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of *Rm*.


**Fig 75. ASR #3**

#### 22.4.3.3.2 LSR

Logical shift right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32 -*n* bits of the result, and it sets the left-hand *n* bits of the result to 0. See Figure 76.

You can use the LSR operation to divide the value in the register *Rm* by $2^n$, if the value is regarded as an unsigned integer.

When the instruction is LSRS, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

**Remark:**

- If *n* is 32 or more, then all the bits in the result are cleared to 0.
- If *n* is 33 or more and the carry flag is updated, it is updated to 0.


**Fig 76. LSR #3**

#### 22.4.3.3.3 LSL

Logical shift left by *n* bits moves the right-hand 32-*n* bits of the register *Rm*, to the left by *n* places, into the left-hand 32-*n* bits of the result, and it sets the right-hand *n* bits of the result to 0. See Figure 77.

You can use the LSL operation to multiply the value in the register *Rm* by $2^n$, if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is LSLS the carry flag is updated to the last bit shifted out, bit[32-*n*], of the register *Rm*. These instructions do not affect the carry flag when used with LSL #0.

**Remark:**

- If *n* is 32 or more, then all the bits in the result are cleared to 0.
- If *n* is 33 or more and the carry flag is updated, it is updated to 0.


**Fig 77. LSL #3**

#### 22.4.3.3.4 ROR

Rotate right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result, and it moves the right-hand *n* bits of the register into the left-hand *n* bits of the result. See Figure 22–78.

When the instruction is RORS the carry flag is updated to the last bit rotation, bit[*n*-1], of the register *Rm*.

**Remark:**

- If *n* is 32, then the value of the result is same as the value in *Rm*, and if the carry flag is updated, it is updated to bit[31] of *Rm*.
- ROR

  with shift length, *n*, greater than 32 is the same as

  ROR

  with shift length *n*-32.

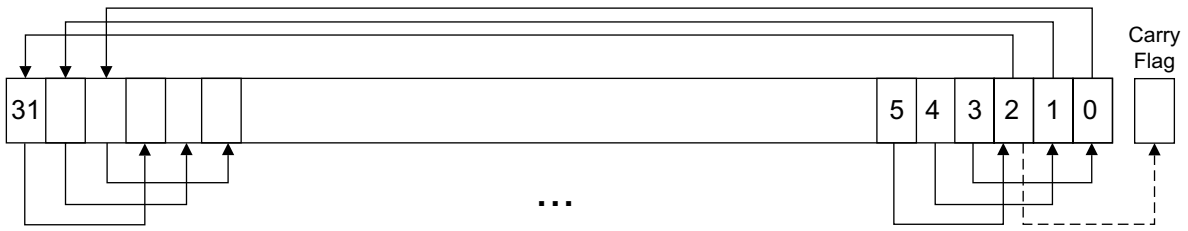



**Fig 78. ROR #3**

### 22.4.3.4 Address alignment

An aligned access is an operation where a word-aligned address is used for a word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

There is no support for unaligned accesses on the Cortex-M0 processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

### 22.4.3.5 PC-relative expressions

A PC-relative expression or **label** is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset. The assembler calculates the required offset from the label and the address of the current instruction. If the offset is too big, the assembler produces an error.

**Remark:**

- For most instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form [PC, #*imm*].

#### 22.4.3.6 Conditional execution

Most data processing instructions update the condition flags in the **Application Program Status Register** (APSR) according to the result of the operation, see Section . Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

You can execute a conditional branch instruction, based on the condition flags set in another instruction, either:

- immediately after the instruction that updated the flags
- after any number of intervening instructions that have not updated the flags.

On the Cortex-M0 processor, conditional execution is available by using conditional branches.

This section describes:

- Section 22.4.3.6.1 “The condition flags”
- Section 22.4.3.6.2 “Condition code suffixes”.

##### 22.4.3.6.1 The condition flags

The APSR contains the following condition flags:

**N —** Set to 1 when the result of the operation was negative, cleared to 0 otherwise.

**Z —** Set to 1 when the result of the operation was zero, cleared to 0 otherwise.

**C —** Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.

**V —** Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR see Section 22–22.3.1.3.5.

A carry occurs:

- if the result of an addition is greater than or equal to $2^{32}$
- if the result of a subtraction is positive or zero
- as the result of a shift or rotate instruction.

Overflow occurs when the sign of the result, in bit[31], does not match the sign of the result had the operation been performed at infinite precision, for example:

- if adding two negative values results in a positive value
- if adding two positive values results in a negative value
- if subtracting a positive value from a negative value generates a positive value
- if subtracting a negative value from a positive value generates a negative value.

The Compare operations are identical to subtracting, for CMP, or adding, for CMN, except that the result is discarded. See the instruction descriptions for more information.

##### 22.4.3.6.2 Condition code suffixes

Conditional branch is shown in syntax descriptions as B{*cond*}. A branch instruction with a condition code is only taken if the condition code flags in the APSR meet the specified condition, otherwise the branch instruction is ignored.  shows the condition codes to use.

Table 325 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

**Table 325. Condition code suffixes**

| Suffix | Flags | Meaning |
|---|---|---|
| EQ | Z = 1 | Equal, last flag setting result was zero |
| NE | Z = 0 | Not equal, last flag setting result was non-zero |
| CS or HS | C = 1 | Higher or same, unsigned |
| CC or LO | C = 0 | Lower, unsigned |
| MI | N = 1 | Negative |
| PL | N = 0 | Positive or zero |
| VS | V = 1 | Overflow |
| VC | V = 0 | No overflow |
| HI | C = 1 and Z = 0 | Higher, unsigned |
| LS | C = 0 or Z = 1 | Lower or same, unsigned |
| GE | N = V | Greater than or equal, signed |
| LT | N != V | Less than, signed |
| GT | Z = 0 and N = V | Greater than, signed |
| LE | Z = 1 and N != V | Less than or equal, signed |
| AL | Can have any value | Always. This is the default when no suffix is specified. |

### 22.4.4 Memory access instructions

Table 326 shows the memory access instructions:

**Table 326. Access instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| LDR{type} | Load Register using register offset | Section 22–22.4.4.3 |
| LDR | Load Register from PC-relative address | Section 22–22.4.4.4 |
| POP | Pop registers from stack | Section 22–22.4.4.6 |
| PUSH | Push registers onto stack | Section 22–22.4.4.6 |
| STM | Store Multiple registers | Section 22–22.4.4.5 |
| STR{type} | Store Register using immediate offset | Section 22–22.4.4.2 |
| STR{type} | Store Register using register offset | Section 22–22.4.4.3 |

#### 22.4.4.1 ADR

Generates a PC-relative address.

##### 22.4.4.1.1 Syntax

ADR *Rd*, *label*

where:

*Rd* is the destination register.

*label* is a PC-relative expression. See Section 22–22.4.3.5.

#### 22.4.4.1.2 Operation

ADR generates an address by adding an immediate value to the PC, and writes the result to the destination register.

ADR facilitates the generation of position-independent code, because the address is PC-relative.

If you use ADR to generate a target address for a BX or BLX instruction, you must ensure that bit[0] of the address you generate is set to 1 for correct execution.

#### 22.4.4.1.3 Restrictions

In this instruction *Rd* must specify R0-R7. The data-value addressed must be word aligned and within 1020 bytes of the current PC.

#### 22.4.4.1.4 Condition flags

This instruction does not change the flags.

#### 22.4.4.1.5 Examples

```
ADR    R1, TextMessage   ; Write address value of a location labelled as
                               ; TextMessage to R1

ADR    R3, [PC,#996]     ; Set R3 to value of PC + 996.
```

### 22.4.4.2 LDR and STR, immediate offset

Load and Store with immediate offset.

#### 22.4.4.2.1 Syntax

LDR *Rt*, [<*Rn* | SP> {, #*imm*}]

LDR<B|H> *Rt*, [*Rn* {, #*imm*}]

STR *Rt*, [<*Rn* | SP>, {,#*imm*}]

STR<B|H> *Rt*, [*Rn* {,#*imm*}]

where:

*Rt is the register to load or store.*

*Rn is the register on which the memory address is based.*

*imm* is an offset from *Rn*. If *imm* is omitted, it is assumed to be zero.

#### 22.4.4.2.2 Operation

LDR, LDRB and LDRH instructions load the register specified by *Rt* with either a word, byte or halfword data value from memory. Sizes less than word are zero extended to 32-bits before being written to the register specified by *Rt*.

STR, STRB and STRH instructions store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* in to memory. The memory address to load from or store to is the sum of the value in the register specified by either *Rn* or SP and the immediate value *imm*.

#### 22.4.4.2.3 Restrictions

In these instructions:

- *Rt* and *Rn* must only specify R0-R7.
- *imm* must be between:
  - 0 and 1020 and an integer multiple of four for LDR and STR using SP as the base register
  - 0 and 124 and an integer multiple of four for LDR and STR using R0-R7 as the base register
  - 0 and 62 and an integer multiple of two for LDRH and STRH
  - 0 and 31 for LDRB and STRB.
- The computed address must be divisible by the number of bytes in the transaction, see Section 22–22.4.3.4.

#### 22.4.4.2.4 Condition flags

These instructions do not change the flags.

#### 22.4.4.2.5 Examples

```
LDR     R4, [R7                ; Loads R4 from the address in R7.
STR     R2, [R0,#const-struc]  ; const-struc is an expression evaluating
                               ; to a constant in the range 0-1020.
```

### 22.4.4.3 LDR and STR, register offset

Load and Store with register offset.

#### 22.4.4.3.1 Syntax

LDR *Rt*, [*Rn*, *Rm*]

LDR<B|H> *Rt*, [*Rn*, *Rm*]

LDR<SB|SH> *Rt*, [*Rn*, *Rm*]

STR *Rt*, [*Rn*, *Rm*]

STR<B|H> *Rt*, [*Rn*, *Rm*]

where:

*Rt* is the register to load or store.

*Rn* is the register on which the memory address is based.

*Rm* is a register containing a value to be used as the offset.

#### 22.4.4.3.2 Operation

LDR, LDRB, U, LDRSB and LDRSH load the register specified by *Rt* with either a word, zero extended byte, zero extended halfword, sign extended byte or sign extended halfword value from memory.

STR, STRB and STRH store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* into memory.

The memory address to load from or store to is the sum of the values in the registers specified by *Rn* and *Rm*.

#### 22.4.4.3.3 Restrictions

In these instructions:

- *Rt*, *Rn*, and *Rm* must only specify R0-R7.
- the computed memory address must be divisible by the number of bytes in the load or store, see Section 22–22.4.3.4.

#### 22.4.4.3.4 Condition flags

These instructions do not change the flags.

#### 22.4.4.3.5 Examples

```
STR    R0, [R5, R1]         ; Store value of R0 into an address equal to
                                 ; sum of R5 and R1

LDRSH  R1, [R2, R3]         ; Load a halfword from the memory address
                                 ; specified by (R2 + R3), sign extend to 32-bits

                            ; and write to R1.
```

### 22.4.4.4 LDR, PC-relative

Load register (literal) from memory.

#### 22.4.4.4.1 Syntax

LDR *Rt*, *label*

where:

*Rt* is the register to load.

*label* is a PC-relative expression. See Section 22–22.4.3.5.

#### 22.4.4.4.2 Operation

Loads the register specified by *Rt* from the word in memory specified by *label*.

#### 22.4.4.4.3 Restrictions

In these instructions, *label* must be within 1020 bytes of the current PC and word aligned.

#### 22.4.4.4.4 Condition flags

These instructions do not change the flags.

#### 22.4.4.4.5 Examples

```
LDR     R0, LookUpTable   ; Load R0 with a word of data from an address
                                ; labelled as LookUpTable.

LDR     R3, [PC, #100]    ; Load R3 with memory word at (PC + 100).
```

### 22.4.4.5 LDM and STM

Load and Store Multiple registers.

#### 22.4.4.5.1 Syntax

LDM *Rn*{!}, reglist

STM *Rn*!, reglist

where:

*Rn* is the register on which the memory addresses are based.

! writeback suffix.

*reglist* is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range, see Section 22–22.4.4.5.5.

LDMIA and LDMFD are synonyms for LDM. LDMIA refers to the base register being Incremented After each access. LDMFD refers to its use for popping data from Full Descending stacks.

STMIA and STMEA are synonyms for STM. STMIA refers to the base register being Incremented After each access. STMEA refers to its use for pushing data onto Empty Ascending stacks.

#### 22.4.4.5.2 Operation

LDM instructions load the registers in *reglist* with word values from memory addresses based on *Rn*.

STM instructions store the word values in the registers in *reglist* to memory addresses based on *Rn*.

The memory addresses used for the accesses are at 4-byte intervals ranging from the value in the register specified by *Rn* to the value in the register specified by *Rn* + 4 * (*n*-1), where *n* is the number of registers in *reglist*. The accesses happens in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address. If the writeback suffix is specified, the value in the register specified by *Rn* + 4 **n* is written back to the register specified by *Rn*.

#### 22.4.4.5.3 Restrictions

In these instructions:

- *reglist* and *Rn* are limited to R0-R7.
- the writeback suffix must always be used unless the instruction is an LDM where reglist also contains *Rn*, in which case the writeback suffix must not be used.

- the value in the register specified by *Rn* must be word aligned. See Section 22–22.4.3.4 for more information.
- for STM, if *Rn* appears in *reglist*, then it must be the first register in the list.

#### 22.4.4.5.4 Condition flags

These instructions do not change the flags.

#### 22.4.4.5.5 Examples

```
LDM     R0,{R0,R3,R4}      ; LDMIA is a synonym for LDM
     STMIA   R1!,{R2-R4,R6}
```

#### 22.4.4.5.6 Incorrect examples

```
STM     R5!,{R4,R5,R6} ; Value stored for R5 is unpredictable
     LDM     R2,{}          ; There must be at least one register in the list
```

### 22.4.4.6 PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

#### 22.4.4.6.1 Syntax

PUSH *reglist*

POP *reglist*

where:

*reglist* is a non-empty list of registers, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

#### 22.4.4.6.2 Operation

PUSH stores registers on the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

POP loads registers from the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

PUSH uses the value in the SP register minus four as the highest memory address,

POP uses the value in the SP register as the lowest memory address, implementing a full-descending stack. On completion,

PUSH updates the SP register to point to the location of the lowest store value,

POP updates the SP register to point to the location above the highest location loaded.

If a POP instruction includes PC in its *reglist*, a branch to this location is performed when the POP instruction has completed. Bit[0] of the value read for the PC is used to update the APSR T-bit. This bit must be 1 to ensure correct operation.

#### 22.4.4.6.3 Restrictions

In these instructions:

- *reglist* must use only R0-R7.

- The exception is LR for a PUSH and PC for a POP.

#### 22.4.4.6.4 Condition flags

These instructions do not change the flags.

#### 22.4.4.6.5 Examples

```
PUSH    {R0,R4-R7}      ; Push R0,R4,R5,R6,R7 onto the stack
     PUSH    {R2,LR}         ; Push R2 and the link-register onto the stack
     POP     {R0,R6,PC}      ; Pop r0,r6 and PC from the stack, then branch to
                             ; the new PC.
```

### 22.4.5 General data processing instructions

Table 327 shows the data processing instructions:

**Table 327. Data processing instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| ADCS | Add with Carry | Section 22–22.4.5.1 |
| ADD{S} | Add | Section 22–22.4.5.1 |
| ANDS | Logical AND | Section 22–22.4.5.2 |
| ASRS | Arithmetic Shift Right | Section 22–22.4.5.3 |
| BICS | Bit Clear | Section 22–22.4.5.2 |
| CMN | Compare Negative | Section 22–22.4.5.4 |
| CMP | Compare | Section 22–22.4.5.4 |
| EORS | Exclusive OR | Section 22–22.4.5.2 |
| LSLS | Logical Shift Left | Section 22–22.4.5.3 |
| LSRS | Logical Shift Right | Section 22–22.4.5.3 |
| MOV{S} | Move | Section 22–22.4.5.5 |
| MULS | Multiply | Section 22–22.4.5.6 |
| MVNS | Move NOT | Section 22–22.4.5.5 |
| ORRS | Logical OR | Section 22–22.4.5.2 |
| REV | Reverse byte order in a word | Section 22–22.4.5.7 |
| REV16 | Reverse byte order in each halfword | Section 22–22.4.5.7 |
| REVSH | Reverse byte order in bottom halfword and sign extend | Section 22–22.4.5.7 |
| RORS | Rotate Right | Section 22–22.4.5.3 |
| RSBS | Reverse Subtract | Section 22–22.4.5.1 |
| SBCS | Subtract with Carry | Section 22–22.4.5.1 |
| SUBS | Subtract | Section 22–22.4.5.1 |
| SXTB | Sign extend a byte | Section 22–22.4.5.8 |
| SXTH | Sign extend a halfword | Section 22–22.4.5.8 |
| UXTB | Zero extend a byte | Section 22–22.4.5.8 |
| UXTH | Zero extend a halfword | Section 22–22.4.5.8 |
| TST | Test | Section 22–22.4.5.9 |

#### 22.4.5.1 ADC, ADD, RSB, SBC, and SUB

Add with carry, Add, Reverse Subtract, Subtract with carry, and Subtract.

##### 22.4.5.1.1 Syntax

ADCS {*Rd*,} *Rn*, *Rm*

ADD{S} {*Rd*,} *Rn*, <*Rm*|#*imm*>

RSBS {*Rd*,} *Rn*, *Rm*, *#0*

SBCS {*Rd*,} *Rn*, *Rm*

SUB{S} {*Rd*,} *Rn*,

<*Rm*|#*imm*>

Where:

S causes an ADD or SUB instruction to update flags

*Rd* specifies the result register

*Rn* specifies the first source register

*Rm* specifies the second source register

*imm* specifies a constant immediate value.

When the optional *Rd* register specifier is omitted, it is assumed to take the same value as *Rn*, for example ADDS R1,R2 is identical to ADDS R1,R1,R2.

##### 22.4.5.1.2 Operation

The ADCS instruction adds the value in *Rn* to the value in *Rm*, adding a further one if the carry flag is set, places the result in the register specified by *Rd* and updates the N, Z, C, and V flags.

The ADD instruction adds the value in *Rn* to the value in *Rm* or an immediate value specified by *imm* and places the result in the register specified by *Rd*.

The ADDS instruction performs the same operation as ADD and also updates the N, Z, C and V flags.

The RSBS instruction subtracts the value in *Rn* from zero, producing the arithmetic negative of the value, and places the result in the register specified by Rd and updates the N, Z, C and V flags.

The SBCS instruction subtracts the value of *Rm* from the value in *Rn*, deducts a further one if the carry flag is set. It places the result in the register specified by Rd and updates the N, Z, C and V flags.

The SUB instruction subtracts the value in *Rm* or the immediate specified by *imm*. It places the result in the register specified by *Rd*.

The SUBS instruction performs the same operation as SUB and also updates the N, Z, C and V flags.

Use ADC and SBC to synthesize multiword arithmetic, see Section 22.4.5.1.4.

See also Section 22–22.4.4.1.

#### 22.4.5.1.3 Restrictions

Table 328 lists the legal combinations of register specifiers and immediate values that can be used with each instruction.

**Table 328. ADC, ADD, RSB, SBC and SUB operand restrictions**

| Instruction | Rd | Rn | Rm | imm | Restrictions |
|---|---|---|---|---|---|
| ADCS | R0-R7 | R0-R7 | R0-R7 | - | *Rd* and *Rn* must specify the same register. |
| ADD | R0-R15 | R0-R15 | R0-PC | - | *Rd* and *Rn* must specify the same register.<br>*Rn* and *Rm* must not both specify PC. |
| | R0-R7 | SP or PC | - | 0-1020 | Immediate value must be an integer multiple of four. |
| | SP | SP | - | 0-508 | Immediate value must be an integer multiple of four. |
| ADDS | R0-R7 | R0-R7 | - | 0-7 | - |
| | R0-R7 | R0-R7 | - | 0-255 | *Rd* and *Rn* must specify the same register. |
| | R0-R7 | R0-R7 | R0-R7 | - | - |
| RSBS | R0-R7 | R0-R7 | - | - | - |
| SBCS | R0-R7 | R0-R7 | R0-R7 | - | *Rd* and *Rn* must specify the same register. |
| SUB | SP | SP | - | 0-508 | Immediate value must be an integer multiple of four. |
| SUBS | R0-R7 | R0-R7 | - | 0-7 | - |
| | R0-R7 | R0-R7 | - | 0-255 | Rd and *Rn* must specify the same register. |
| | R0-R7 | R0-R7 | R0-R7 | - | - |

#### 22.4.5.1.4 Examples

The following shows two instructions that add a 64-bit integer contained in R0 and R1 to another 64-bit integer contained in R2 and R3, and place the result in R0 and R1.

**64-bit addition:**

```
ADDS    R0, R0, R2    ; add the least significant words
ADCS    R1, R1, R3    ; add the most significant words with carry
```

Multiword values do not have to use consecutive registers. The following shows instructions that subtract a 96-bit integer contained in R1, R2, and R3 from another contained in R4, R5, and R6. The example stores the result in R4, R5, and R6.

**96-bit subtraction:**

```
SUBS    R4, R4, R1    ; subtract the least significant words
SBCS    R5, R5, R2    ; subtract the middle words with carry
SBCS    R6, R6, R3    ; subtract the most significant words with carry
```

The following shows the RSBS instruction used to perform a 1's complement of a single register.

**Arithmetic negation:**

```
RSBS    R7, R7, #0    ; subtract R7 from zero
```

### 22.4.5.2 AND, ORR, EOR, and BIC

Logical AND, OR, Exclusive OR, and Bit Clear.

#### 22.4.5.2.1 Syntax

ANDS {*Rd,*} *Rn*, *Rm*

ORRS {*Rd,*} *Rn*, *Rm*

EORS {*Rd,*} *Rn*, *Rm*

BICS {*Rd,*} *Rn*, *Rm*

where:

*Rd* is the destination register.

*Rn* is the register holding the first operand and is the same as the destination register.

*Rm* second register.

#### 22.4.5.2.2 Operation

The AND, EOR, and ORR instructions perform bitwise AND, exclusive OR, and inclusive OR operations on the values in *Rn* and *Rm*.

The BIC instruction performs an AND operation on the bits in *Rn* with the logical negation of the corresponding bits in the value of *Rm*.

The condition code flags are updated on the result of the operation, see Section 22.4.3.6.1.

#### 22.4.5.2.3 Restrictions

In these instructions, *Rd*, *Rn*, and *Rm* must only specify R0-R7.

#### 22.4.5.2.4 Condition flags

These instructions:

- update the N and Z flags according to the result
- do not affect the C or V flag.

#### 22.4.5.2.5 Examples

```
ANDS    R2, R2, R1
     ORRS    R2, R2, R5
     ANDS    R5, R5, R8
     EORS    R7, R7, R6
     BICS    R0, R0, R1
```

### 22.4.5.3 ASR, LSL, LSR, and ROR

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, and Rotate Right.

#### 22.4.5.3.1 Syntax

ASRS {Rd,} *Rm*, *Rs*

ASRS {Rd,} *Rm*, *#imm*

LSLS {Rd,} *Rm*, *Rs*

LSLS {Rd,} *Rm*, *#imm*

LSRS {Rd,} *Rm*, *Rs*

LSRS {Rd,} *Rm*, *#imm*

RORS {Rd,} *Rm*, *Rs*

where:

*Rd* is the destination register. If *Rd* is omitted, it is assumed to take the same value as *Rm*.

*Rm* is the register holding the value to be shifted.

*Rs* is the register holding the shift length to apply to the value in *Rm*.

*imm* is the shift length.

The range of shift length depends on the instruction:

**ASR —** shift length from 1 to 32

**LSL —** shift length from 0 to 31

**LSR —** shift length from 1 to 32.

**Remark:** MOVS Rd, *Rm* is a pseudonym for LSLS Rd, *Rm*, #0.

#### 22.4.5.3.2 Operation

ASR, LSL, LSR, and ROR perform an arithmetic-shift-left, logical-shift-left, logical-shift-right or a right-rotation of the bits in the register *Rm* by the number of places specified by the immediate *imm* or the value in the least-significant byte of the register specified by *Rs*.

For details on what result is generated by the different instructions, see Section 22–22.4.3.3.

#### 22.4.5.3.3 Restrictions

In these instructions, *Rd*, *Rm*, and *Rs* must only specify R0-R7. For non-immediate instructions, *Rd* and *Rm* must specify the same register.

#### 22.4.5.3.4 Condition flags

These instructions update the N and Z flags according to the result.

The C flag is updated to the last bit shifted out, except when the shift length is 0, see Section 22–22.4.3.3. The V flag is left unmodified.

#### 22.4.5.3.5 Examples

```
ASRS    R7, R5, #9  ; Arithmetic shift right by 9 bits
     LSLS    R1, R2, #3  ; Logical shift left by 3 bits with flag update
     LSRS    R4, R5, #6  ; Logical shift right by 6 bits
     RORS    R4, R4, R6  ; Rotate right by the value in the bottom byte of R6.
```

### 22.4.5.4 CMP and CMN

Compare and Compare Negative.

#### 22.4.5.4.1 Syntax

CMN *Rn*, *Rm*

CMP *Rn*, *#imm*

CMP *Rn*, *Rm*

where:

*Rn* is the register holding the first operand.

*Rm* is the register to compare with.

*imm* is the immediate value to compare with.

#### 22.4.5.4.2 Operation

These instructions compare the value in a register with either the value in another register or an immediate value. They update the condition flags on the result, but do not write the result to a register.

The CMP instruction subtracts either the value in the register specified by *Rm*, or the immediate *imm* from the value in *Rn* and updates the flags. This is the same as a SUBS instruction, except that the result is discarded.

The CMN instruction adds the value of *Rm* to the value in *Rn* and updates the flags. This is the same as an ADDS instruction, except that the result is discarded.

#### 22.4.5.4.3 Restrictions

For the:

- CMN

  instruction *Rn*, and *Rm* must only specify R0-R7.
- CMP instruction:
  - *Rn* and *Rm* can specify R0-R14
  - immediate must be in the range 0-255.

#### 22.4.5.4.4 Condition flags

These instructions update the N, Z, C and V flags according to the result.

#### 22.4.5.4.5 Examples

```
CMP     R2, R9
     CMN     R0, R2
```

### 22.4.5.5 MOV and MVN

Move and Move NOT.

#### 22.4.5.5.1 Syntax

MOV{S} *Rd*, *Rm*

MOVS *Rd*, *#imm*

MVNS *Rd*, *Rm*

where:

*S* is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see Section 22–22.4.3.6.

*Rd* is the destination register.

*Rm* is a register.

*imm* is any value in the range 0-255.

#### 22.4.5.5.2 Operation

The MOV instruction copies the value of *Rm* into *Rd*.

The MOVS instruction performs the same operation as the MOV instruction, but also updates the N and Z flags.

The MVNS instruction takes the value of *Rm*, performs a bitwise logical negate operation on the value, and places the result into *Rd*.

#### 22.4.5.5.3 Restrictions

In these instructions, *Rd*, and *Rm* must only specify R0-R7.

When *Rd* is the PC in a MOV instruction:

- Bit[0] of the result is discarded.
- A branch occurs to the address created by forcing bit[0] of the result to 0. The T-bit remains unmodified.

**Remark:** Though it is possible to use MOV as a branch instruction, ARM strongly recommends the use of a BX or BLX instruction to branch for software portability.

#### 22.4.5.5.4 Condition flags

If *S* is specified, these instructions:

- update the N and Z flags according to the result
- do not affect the C or V flags.

#### 22.4.5.5.5 Example

```
MOVS  R0, #0x000B     ; Write value of 0x000B to R0, flags get updated
      MOVS  R1, #0x0        ; Write value of zero to R1, flags are updated
      MOV   R10, R12        ; Write value in R12 to R10, flags are not updated
      MOVS  R3, #23         ; Write value of 23 to R3
      MOV   R8, SP          ; Write value of stack pointer to R8
      MVNS  R2, R0          ; Write inverse of R0 to the R2 and update flags
```

### 22.4.5.6 MULS

Multiply using 32-bit operands, and producing a 32-bit result.

#### 22.4.5.6.1 Syntax

MULS Rd, *Rn*, *Rm*

where:

*Rd* is the destination register.

*Rn, Rm* are registers holding the values to be multiplied.

#### 22.4.5.6.2 Operation

The MUL instruction multiplies the values in the registers specified by *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*. The condition code flags are updated on the result of the operation, see Section 22–22.4.3.6.

The results of this instruction does not depend on whether the operands are signed or unsigned.

#### 22.4.5.6.3 Restrictions

In this instruction:

- *Rd*, *Rn*, and *Rm* must only specify R0-R7
- *Rd* must be the same as *Rm*.

#### 22.4.5.6.4 Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

#### 22.4.5.6.5 Examples

```
MULS    R0, R2, R0      ; Multiply with flag update, R0 = R0 x R2
```

### 22.4.5.7 REV, REV16, and REVSH

Reverse bytes.

#### 22.4.5.7.1 Syntax

REV Rd, *Rn*

REV16 Rd, *Rn*

REVSH Rd, *Rn*

where:

*Rd* is the destination register.

*Rn* is the source register.

#### 22.4.5.7.2 Operation

Use these instructions to change endianness of data:

**REV —** converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

**REV16 —** converts two packed 16-bit big-endian data into little-endian data or two packed 16-bit little-endian data into big-endian data.

**REVSH —** converts 16-bit signed big-endian data into 32-bit signed little-endian data or 16-bit signed little-endian data into 32-bit signed big-endian data.

#### 22.4.5.7.3 Restrictions

In these instructions, *Rd*, and *Rn* must only specify R0-R7.

#### 22.4.5.7.4 Condition flags

These instructions do not change the flags.

#### 22.4.5.7.5 Examples

```
REV    R3, R7  ; Reverse byte order of value in R7 and write it to R3
     REV16  R0, R0  ; Reverse byte order of each 16-bit halfword in R0
     REVSH  R0, R5  ; Reverse signed halfword
```

### 22.4.5.8 SXT and UXT

Sign extend and Zero extend.

#### 22.4.5.8.1 Syntax

SXTB Rd, *Rm*

SXTH Rd, *Rm*

UXTB Rd, *Rm*

UXTH Rd, *Rm*

where:

*Rd* is the destination register.

*Rm* is the register holding the value to be extended.

#### 22.4.5.8.2 Operation

These instructions extract bits from the resulting value:

- SXTB extracts bits[7:0] and sign extends to 32 bits
- UXTB extracts bits[7:0] and zero extends to 32 bits
- SXTH extracts bits[15:0] and sign extends to 32 bits
- UXTH extracts bits[15:0] and zero extends to 32 bits.

#### 22.4.5.8.3 Restrictions

In these instructions, *Rd* and *Rm* must only specify R0-R7.

#### 22.4.5.8.4 Condition flags

These instructions do not affect the flags.

#### 22.4.5.8.5 Examples

```
    SXTH  R4, R6          ; Obtain the lower halfword of the
                                ; value in R6 and then sign extend to
                                ; 32 bits and write the result to R4.
         UXTB  R3, R1           ; Extract lowest byte of the value in R10 and zero
                                ; extend it, and write the result to R3
```

### 22.4.5.9 TST

Test bits.

#### 22.4.5.9.1 Syntax

TST *Rn*, *Rm*

where:

*Rn* is the register holding the first operand.

*Rm* the register to test against.

#### 22.4.5.9.2 Operation

This instruction tests the value in a register against another register. It updates the condition flags based on the result, but does not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value in *Rm*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with a register that has that bit set to 1 and all other bits cleared to 0.

#### 22.4.5.9.3 Restrictions

In these instructions, *Rn* and *Rm* must only specify R0-R7.

#### 22.4.5.9.4 Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

#### 22.4.5.9.5 Examples

```
    TST     R0, R1  ; Perform bitwise AND of R0 value and R1 value,
                          ; condition code flags are updated but result is discarded
```

## 22.4.6 Branch and control instructions

Table 329 shows the branch and control instructions:

**Table 329. Branch and control instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| B{cc} | Branch {conditionally} | Section 22–22.4.6.1 |

**Table 329. Branch and control instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| BL | Branch with Link | Section 22–22.4.6.1 |
| BLX | Branch indirect with Link | Section 22–22.4.6.1 |
| BX | Branch indirect | Section 22–22.4.6.1 |

### 22.4.6.1 B, BL, BX, and BLX

Branch instructions.

#### 22.4.6.1.1 Syntax

B{*cond*} *label*

BL *label*

BX *Rm*

BLX *Rm*

where:

*cond* is an optional condition code, see Section 22–22.4.3.6.

*label* is a PC-relative expression. See Section 22–22.4.3.5.

*Rm* is a register providing the address to branch to.

#### 22.4.6.1.2 Operation

All these instructions cause a branch to the address indicated by *label* or contained in the register specified by *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR, the link register R14.
- The BX and BLX instructions result in a HardFault exception if bit[0] of *Rm* is 0.

BL and BLX instructions also set bit[0] of the LR to 1. This ensures that the value is suitable for use by a subsequent POP {PC} or BX instruction to perform a successful return branch.

Table 330 shows the ranges for the various branch instructions.

**Table 330. Branch ranges**

| Instruction | Branch range |
|---|---|
| B *label* | –2 KB to +2 KB |
| B*cond label* | –256 bytes to +254 bytes |
| BL *label* | –16 MB to +16 MB |
| BX *Rm* | Any value in register |
| BLX *Rm* | Any value in register |

#### 22.4.6.1.3 Restrictions

In these instructions:

- Do not use SP or PC in the BX or BLX instruction.

- For BX and BLX, bit[0] of *Rm* must be 1 for correct execution. Bit[0] is used to update the EPSR T-bit and is discarded from the target address.

**Remark:** B*cond* is the only conditional instruction on the Cortex-M0 processor.

#### 22.4.6.1.4 Condition flags

These instructions do not change the flags.

#### 22.4.6.1.5 Examples

```
B      loopA  ; Branch to loopA
      BL     funC   ; Branch with link (Call) to function funC, return address
                    ; stored in LR
      BX     LR     ; Return from function call
      BLX    R0     ; Branch with link and exchange (Call) to a address stored
                    ; in R0

BEQ    labelD ; Conditionally branch to labelD if last flag setting

               ; instruction set the Z flag, else do not branch.
```

### 22.4.7 Miscellaneous instructions

Table 331 shows the remaining Cortex-M0 instructions:

**Table 331. Miscellaneous instructions**

| Mnemonic | Brief description | See |
|---|---|---|
| BKPT | Breakpoint | Section 22–22.4.7.1 |
| CPSID | Change Processor State, Disable Interrupts | Section 22–22.4.7.2 |
| CPSIE | Change Processor State, Enable Interrupts | Section 22–22.4.7.2 |
| DMB | Data Memory Barrier | Section 22–22.4.7.3 |
| DSB | Data Synchronization Barrier | Section 22–22.4.7.4 |
| ISB | Instruction Synchronization Barrier | Section 22–22.4.7.5 |
| MRS | Move from special register to register | Section 22–22.4.7.6 |
| MSR | Move from register to special register | Section 22–22.4.7.7 |
| NOP | No Operation | Section 22–22.4.7.8 |
| SEV | Send Event | Section 22–22.4.7.9 |

Table 331. Miscellaneous instructions

| Mnemonic | Brief description | See |
|---|---|---|
| SVC | Supervisor Call | Section 22–22.4.7.10 |
| WFE | Wait For Event | Section 22–22.4.7.11 |
| WFI | Wait For Interrupt | Section 22–22.4.7.12 |

### 22.4.7.1 BKPT

Breakpoint.

#### 22.4.7.1.1 Syntax

BKPT #*imm*

where:

*imm* is an integer in the range 0-255.

#### 22.4.7.1.2 Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached. *imm* is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The processor might also produce a HardFault or go in to lockup if a debugger is not attached when a BKPT instruction is executed. See Section 22–22.3.4.1 for more information.

#### 22.4.7.1.3 Restrictions

There are no restrictions.

#### 22.4.7.1.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.1.5 Examples

```
BKPT #0     ; Breakpoint with immediate value set to 0x0.
```

### 22.4.7.2 CPS

Change Processor State.

#### 22.4.7.2.1 Syntax

CPSID i

CPSIE i

#### 22.4.7.2.2 Operation

CPS changes the PRIMASK special register values. CPSID causes interrupts to be disabled by setting PRIMASK. CPSIE cause interrupts to be enabled by clearing PRIMASK.See Section 22–22.3.1.3.6 for more information about these registers.

#### 22.4.7.2.3 Restrictions

There are no restrictions.

#### 22.4.7.2.4 Condition flags

This instruction does not change the condition flags.

#### 22.4.7.2.5 Examples

```
CPSID i ; Disable all interrupts except NMI (set PRIMASK)

CPSIE i ; Enable interrupts (clear PRIMASK)
```

### 22.4.7.3 DMB

Data Memory Barrier.

#### 22.4.7.3.1 Syntax

DMB

#### 22.4.7.3.2 Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear in program order before the DMB instruction are observed before any explicit memory accesses that appear in program order after the DMB instruction. DMB does not affect the ordering of instructions that do not access memory.

#### 22.4.7.3.3 Restrictions

There are no restrictions.

#### 22.4.7.3.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.3.5 Examples

```
DMB  ; Data Memory Barrier
```

### 22.4.7.4 DSB

Data Synchronization Barrier.

#### 22.4.7.4.1 Syntax

DSB

#### 22.4.7.4.2 Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

#### 22.4.7.4.3 Restrictions

There are no restrictions.

#### 22.4.7.4.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.4.5 Examples

```
DSB ; Data Synchronisation Barrier
```

### 22.4.7.5 ISB

Instruction Synchronization Barrier.

#### 22.4.7.5.1 Syntax

ISB

#### 22.4.7.5.2 Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

#### 22.4.7.5.3 Restrictions

There are no restrictions.

#### 22.4.7.5.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.5.5 Examples

```
ISB  ; Instruction Synchronisation Barrier
```

### 22.4.7.6 MRS

Move the contents of a special register to a general-purpose register.

#### 22.4.7.6.1 Syntax

MRS *Rd*, *spec_reg*

where:

*Rd* is the general-purpose destination register.

*spec_reg* is one of the special-purpose registers: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

#### 22.4.7.6.2 Operation

MRS stores the contents of a special-purpose register to a general-purpose register. The MRS instruction can be combined with the MR instruction to produce read-modify-write sequences, which are suitable for modifying a specific flag in the PSR.

See Section 22–22.4.7.7.

#### 22.4.7.6.3 Restrictions

In this instruction, *Rd* must not be SP or PC.

#### 22.4.7.6.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.6.5 Examples

```
MRS  R0, PRIMASK ; Read PRIMASK value and write it to R0
```

### 22.4.7.7 MSR

Move the contents of a general-purpose register into the specified special register.

#### 22.4.7.7.1 Syntax

MSR *spec_reg*, *Rn*

where:

*Rn* is the general-purpose source register.

*spec_reg* is the special-purpose destination register: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

#### 22.4.7.7.2 Operation

MSR updates one of the special registers with the value from the register specified by *Rn*.

See Section 22–22.4.7.6.

#### 22.4.7.7.3 Restrictions

In this instruction, *Rn* must not be SP and must not be PC.

#### 22.4.7.7.4 Condition flags

This instruction updates the flags explicitly based on the value in *Rn*.

#### 22.4.7.7.5 Examples

```
MSR  CONTROL, R1 ; Read R1 value and write it to the CONTROL register
```

### 22.4.7.8 NOP

No Operation.

#### 22.4.7.8.1 Syntax

NOP

#### 22.4.7.8.2 Operation

NOP performs no operation and is not guaranteed to be time consuming. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to place the subsequent instructions on a 64-bit boundary.

#### 22.4.7.8.3 Restrictions

There are no restrictions.

#### 22.4.7.8.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.8.5 Examples

```
NOP  ; No operation
```

### 22.4.7.9 SEV

Send Event.

#### 22.4.7.9.1 Syntax

SEV

#### 22.4.7.9.2 Operation

SEV causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register, see Section 22–22.3.5.

See also Section 22–22.4.7.11.

#### 22.4.7.9.3 Restrictions

There are no restrictions.

#### 22.4.7.9.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.9.5 Examples

```
SEV ; Send Event
```

### 22.4.7.10 SVC

Supervisor Call.

#### 22.4.7.10.1 Syntax

SVC #*imm*

where:

*imm* is an integer in the range 0-255.

#### 22.4.7.10.2 Operation

The SVC instruction causes the SVC exception.

*imm* is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

#### 22.4.7.10.3 Restrictions

There are no restrictions.

#### 22.4.7.10.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.10.5 Examples

```
SVC  #0x32 ; Supervisor Call (SVC handler can extract the immediate value
                ; by locating it via the stacked PC)
```

### 22.4.7.11 WFE

Wait For Event.

**Remark:** The WFE instruction is not implemented on the LPC111x/LPC11Cxx

#### 22.4.7.11.1 Syntax

WFE

#### 22.4.7.11.2 Operation

If the event register is 0, WFE suspends execution until one of the following events occurs:

- an exception, unless masked by the exception mask registers or the current priority level
- an exception enters the Pending state, if SEVONPEND in the System Control Register is set
- a Debug Entry request, if debug is enabled
- an event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and completes immediately.

For more information see Section 22–22.3.5.

**Remark:** WFE is intended for power saving only. When writing software assume that WFE might behave as NOP.

#### 22.4.7.11.3 Restrictions

There are no restrictions.

#### 22.4.7.11.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.11.5 Examples

```
WFE  ; Wait for event
```

### 22.4.7.12 WFI

Wait for Interrupt.

#### 22.4.7.12.1 Syntax

WFI

#### 22.4.7.12.2 Operation

WFI

suspends execution until one of the following events occurs:

- an exception
- an interrupt becomes pending which would preempt if PRIMASK was clear
- a Debug Entry request, regardless of whether debug is enabled.

**Remark:** WFI is intended for power saving only. When writing software assume that WFI might behave as a NOP operation.

#### 22.4.7.12.3 Restrictions

There are no restrictions.

#### 22.4.7.12.4 Condition flags

This instruction does not change the flags.

#### 22.4.7.12.5 Examples

```
WFI ; Wait for interrupt
```

## 22.5 Peripherals

### 22.5.1 About the ARM Cortex-M0

The address map of the **Private peripheral bus** (PPB) is:

**Table 332. Core peripheral register regions**

| Address | Core peripheral | Description |
|---|---|---|
| 0xE000E008-0xE000E00F | System Control Block | Table 22–341 |
| 0xE000E010-0xE000E01F | System timer | Table 22–350 |
| 0xE000E100-0xE000E4EF | Nested Vectored Interrupt Controller | Table 22–333 |
| 0xE000ED00-0xE000ED3F | System Control Block | Table 22–341 |
| 0xE000EF00-0xE000EF03 | Nested Vectored Interrupt Controller | Table 22–333 |

In register descriptions, the register **type** is described as follows:

**RW —** Read and write.

**RO —** Read-only.

**WO —** Write-only.

### 22.5.2 Nested Vectored Interrupt Controller

This section describes the **Nested Vectored Interrupt Controller** (NVIC) and the registers it uses. The NVIC supports:

- 32 interrupts.

- A programmable priority level of 0-3 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Interrupt tail-chaining.
- An external **Non-maskable interrupt** (NMI). The NMI is not implemented on the LPC111x/LPC11Cxx.

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

**Table 333. NVIC register summary**

| Address | Name | Type | Reset value | Description |
|---|---|---|---|---|
| 0xE000E100 | ISER | RW | 0x00000000 | Section 22–22.5.2.2 |
| 0xE000E180 | ICER | RW | 0x00000000 | Section 22–22.5.2.3 |
| 0xE000E200 | ISPR | RW | 0x00000000 | Section 22–22.5.2.4 |
| 0xE000E280 | ICPR | RW | 0x00000000 | Section 22–22.5.2.5 |
| 0xE000E400-0xE000E41C | IPR0-7 | RW | 0x00000000 | Section 22–22.5.2.6 |

### 22.5.2.1 Accessing the Cortex-M0 NVIC registers using CMSIS

CMSIS functions enable software portability between different Cortex-M profile processors.

To access the NVIC registers when using CMSIS, use the following functions:

**Table 334. CMISIS access NVIC functions**

| CMSIS function | Description |
|---|---|
| void NVIC_EnableIRQ(IRQn_Type IRQn)[1] | Enables an interrupt or exception. |
| void NVIC_DisableIRQ(IRQn_Type IRQn)[1] | Disables an interrupt or exception. |
| void NVIC_SetPendingIRQ(IRQn_Type IRQn)[1] | Sets the pending status of interrupt or exception to 1. |
| void NVIC_ClearPendingIRQ(IRQn_Type IRQn)[1] | Clears the pending status of interrupt or exception to 0. |
| uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)[1] | Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1. |
| void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)[1] | Sets the priority of an interrupt or exception with configurable priority level to 1. |
| uint32_t NVIC_GetPriority(IRQn_Type IRQn)[1] | Reads the priority of an interrupt or exception with configurable priority level. This function returns the current priority level. |

[1] The input parameter IRQn is the IRQ number, see Table 320 for more information.

### 22.5.2.2 Interrupt Set-enable Register

The ISER enables interrupts, and shows which interrupts are enabled. See the register summary in Table 333 for the register attributes.

The bit assignments are:

**Table 335. ISER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | SETENA | Interrupt set-enable bits.<br>Write:<br>0 = no effect<br>1 = enable interrupt.<br>Read:<br>0 = interrupt disabled<br>1 = interrupt enabled. |

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

### 22.5.2.3 Interrupt Clear-enable Register

The ICER disables interrupts, and show which interrupts are enabled. See the register summary in Table 22–333 for the register attributes.

The bit assignments are:

**Table 336. ICER bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | CLRENA | Interrupt clear-enable bits.<br>Write:<br>0 = no effect<br>1 = disable interrupt.<br>Read:<br>0 = interrupt disabled<br>1 = interrupt enabled. |

### 22.5.2.4 Interrupt Set-pending Register

The ISPR forces interrupts into the pending state, and shows which interrupts are pending. See the register summary in Table 22–333 for the register attributes.

The bit assignments are:

**Table 337. ISPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | SETPEND | Interrupt set-pending bits.<br>Write:<br>0 = no effect<br>1 = changes interrupt state to pending.<br>Read:<br>0 = interrupt is not pending<br>1 = interrupt is pending. |

**Remark:** Writing 1 to the ISPR bit corresponding to:

- an interrupt that is pending has no effect

- a disabled interrupt sets the state of that interrupt to pending.

### 22.5.2.5 Interrupt Clear-pending Register

The ICPR removes the pending state from interrupts, and shows which interrupts are pending. See the register summary in Table 22–333 for the register attributes.

The bit assignments are:

**Table 338. ICPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:0] | CLRPEND | Interrupt clear-pending bits.<br>Write:<br>0 = no effect<br>1 = removes pending state an interrupt.<br>Read:<br>0 = interrupt is not pending<br>1 = interrupt is pending. |

**Remark:** Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

### 22.5.2.6 Interrupt Priority Registers

The IPR0-IPR7 registers provide an 2-bit priority field for each interrupt. These registers are only word-accessible. See the register summary in Table 22–333 for their attributes. Each register holds four priority fields as shown:

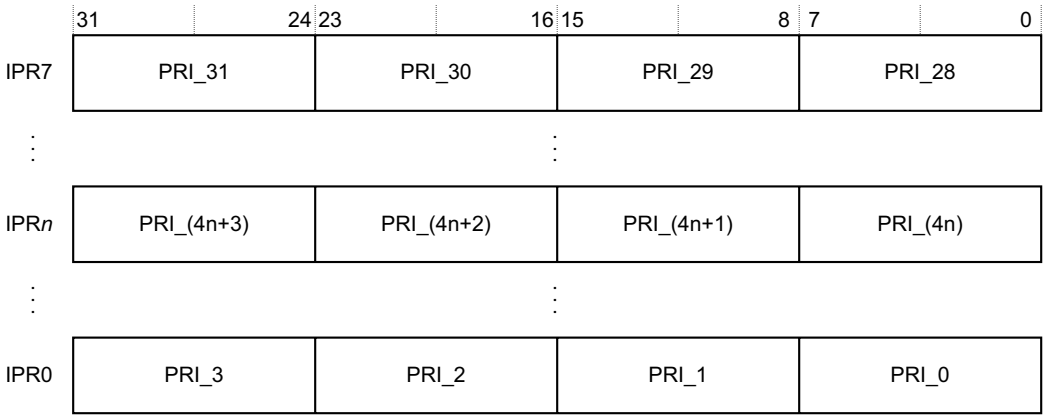



**Fig 79. IPR register**

**Table 339. IPR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | Priority, byte offset 3 | Each priority field holds a priority value, 0-3. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:6] of each field, bits [5:0] read as zero and ignore writes. |
| [23:16] | Priority, byte offset 2 | |
| [15:8] | Priority, byte offset 1 | |
| [7:0] | Priority, byte offset 0 | |

See Section 22–22.5.2.1 for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt **M** as follows:

- the corresponding IPR number, **N**, is given by **N** = **N** DIV 4
- the byte offset of the required Priority field in this register is **M** MOD 4, where:
  - byte offset 0 refers to register bits[7:0]
  - byte offset 1 refers to register bits[15:8]
  - byte offset 2 refers to register bits[23:16]
  - byte offset 3 refers to register bits[31:24].

### 22.5.2.7 Level-sensitive and pulse interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see Section 22.5.2.7.1. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

#### 22.5.2.7.1 Hardware and software control of interrupts

The Cortex-M0 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is active and the corresponding interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see Section 22–22.5.2.4.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.

- For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.

  If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.

- Software writes to the corresponding interrupt clear-pending register bit.

  For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

  For a pulse interrupt, state of the interrupt changes to:

  - inactive, if the state was pending
  - active, if the state was active and pending.

#### 22.5.2.8 NVIC usage hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

##### 22.5.2.8.1 NVIC programming hints

Software uses the `CPSIE i` and  instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts

void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

**Table 340. CMSIS functions for NVIC control**

| CMSIS interrupt control function | Description |
|---|---|
| `void NVIC_EnableIRQ(IRQn_t IRQn)` | Enable IRQn |
| `void NVIC_DisableIRQ(IRQn_t IRQn)` | Disable IRQn |
| `uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)` | Return true (1) if IRQn is pending |
| `void NVIC_SetPendingIRQ (IRQn_t IRQn)` | Set IRQn pending |
| `void NVIC_ClearPendingIRQ (IRQn_t IRQn)` | Clear IRQn pending status |
| `void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)` | Set priority for IRQn |
| `uint32_t NVIC_GetPriority (IRQn_t IRQn)` | Read priority of IRQn |
| `void NVIC_SystemReset (void)` | Reset the system |

The input parameter IRQn is the IRQ number, see Table 22–320 for more information. For more information about these functions, see the CMSIS documentation.

### 22.5.3 System Control Block

The **System Control Block** (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The SCB registers are:

**Table 341. Summary of the SCB registers**

| Address | Name | Type | Reset value | Description |
|---|---|---|---|---|
| 0xE000ED00 | CPUID | RO | 0x410CC200 | Section 22.5.3.2 |
| 0xE000ED04 | ICSR | RW[1] | 0x00000000 | Section 22–22.5.3.3 |
| 0xE000ED0C | AIRCR | RW[1] | 0xFA050000 | Section 22–22.5.3.4 |
| 0xE000ED10 | SCR | RW | 0x00000000 | Section 22–22.5.3.5 |
| 0xE000ED14 | CCR | RO | 0x00000204 | Section 22–22.5.3.6 |
| 0xE000ED1C | SHPR2 | RW | 0x00000000 | Section 22–22.5.3.7.1 |
| 0xE000ED20 | SHPR3 | RW | 0x00000000 | Section 22–22.5.3.7.2 |

[1] See the register description for more information.

### 22.5.3.1 The CMSIS mapping of the Cortex-M0 SCB registers

To improve software efficiency, the CMSIS simplifies the SCB register presentation. In the CMSIS, the array `SHP[1]` corresponds to the registers SHPR2-SHPR3.

### 22.5.3.2 CPUID Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in  for its attributes. The bit assignments are:

**Table 342. CPUID register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | Implementer | Implementer code:<br>0x41 = ARM |
| [23:20] | Variant | Variant number, the r value in the r**n**p**n** product revision identifier:<br>0x0 = Revision 0 |
| [19:16] | Constant | Constant that defines the architecture of the processor:, reads as<br>0xC = ARMv6-M architecture |
| [15:4] | Partno | Part number of the processor:<br>0xC20 = Cortex-M0 |
| [3:0] | Revision | Revision number, the p value in the r**n**p**n** product revision identifier:<br>0x0 = Patch 0 |

### 22.5.3.3 Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the **Non-Maskable Interrupt** (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception

– whether any interrupts are pending.

See the register summary in Table 22–341 for the ICSR attributes. The bit assignments are:

**Table 343. ICSR bit assignments**

| Bits | Name | Type | Function |
|---|---|---|---|
| [31] | NMIPENDSET[2] | RW | NMI set-pending bit.<br>Write:<br>0 = no effect<br>1 = changes NMI exception state to pending.<br>Read:<br>0 = NMI exception is not pending<br>1 = NMI exception is pending.<br>Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it detects a write of 1 to this bit. Entering the handler then clears this bit to 0. This means a read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler. |
| [30:29] | - | - | Reserved. |
| [28] | PENDSVSET | RW | PendSV set-pending bit.<br>Write:<br>0 = no effect<br>1 = changes PendSV exception state to pending.<br>Read:<br>0 = PendSV exception is not pending<br>1 = PendSV exception is pending.<br>Writing 1 to this bit is the only way to set the PendSV exception state to pending. |
| [27] | PENDSVCLR | WO | PendSV clear-pending bit.<br>Write:<br>0 = no effect<br>1 = removes the pending state from the PendSV exception. |
| [26] | PENDSTSET | RW | SysTick exception set-pending bit.<br>Write:<br>0 = no effect<br>1 = changes SysTick exception state to pending.<br>Read:<br>0 = SysTick exception is not pending<br>1 = SysTick exception is pending. |

**Table 343. ICSR bit assignments**

| Bits | Name | Type | Function |
|---|---|---|---|
| [25] | PENDSTCLR | WO | SysTick exception clear-pending bit.<br>Write:<br>0 = no effect<br>1 = removes the pending state from the SysTick exception.<br>This bit is WO. On a register read its value is Unknown. |
| [24:23] | - | - | Reserved. |
| [22] | ISRPENDING | RO | Interrupt pending flag, excluding NMI and Faults:<br>0 = interrupt not pending<br>1 = interrupt pending. |
| [21:18] | - | - | Reserved. |
| [17:12] | VECTPENDING | RO | Indicates the exception number of the highest priority pending enabled exception:<br>0 = no pending exceptions<br>Nonzero = the exception number of the highest priority pending enabled exception. |
| [11:6] | - | - | Reserved. |
| [5:0] | VECTACTIVE[1] | RO | Contains the active exception number:<br>0 = Thread mode<br>Nonzero = The exception number[1] of the currently active exception.<br>**Remark:** Subtract 16 from this value to obtain the CMSIS IRQ number that identifies the corresponding bit in the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-pending, and Priority Register, see Table 22–315. |

[1] This is the same value as IPSR bits[5:0], see Table 22–315.

[2] The NMI is not implemented on the LPC111x/LPC11Cxx.

When you write to the ICSR, the effect is Unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

### 22.5.3.4 Application Interrupt and Reset Control Register

The AIRCR provides endian status for data accesses and reset control of the system. See the register summary in Table 22–341 and Table 22–344 for its attributes.

To write to this register, you must write `0x05FA` to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

**Table 344. AIRCR bit assignments**

| Bits | Name | Type | Function |
|---|---|---|---|
| [31:16] | Read: Reserved<br>Write: VECTKEY | RW | Register key:<br>Reads as Unknown<br>On writes, write 0x05FA to VECTKEY, otherwise the write is ignored. |
| [15] | ENDIANESS | RO | Data endianness implemented:<br>0 = Little-endian<br>1 = Big-endian. |
| [14:3] | - | - | Reserved |
| [2] | SYSRESETREQ | WO | System reset request:<br>0 = no effect<br>1 = requests a system level reset.<br>This bit reads as 0. |
| [1] | VECTCLRACTIVE | WO | Reserved for debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable. |
| [0] | - | - | Reserved. |

### 22.5.3.5 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in Table 22–341 for its attributes. The bit assignments are:

**Table 345. SCR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:5] | - | Reserved. |
| [4] | SEVONPEND | Send Event on Pending bit:<br>0 = only enabled interrupts or events can wake-up the processor, disabled interrupts are excluded<br>1 = enabled events and all interrupts, including disabled interrupts, can wake-up the processor.<br>When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.<br>The processor also wakes up on execution of an SEV instruction. |
| [3] | - | Reserved. |
| [2] | SLEEPDEEP | Controls whether the processor uses sleep or deep sleep as its low power mode:<br>0 = sleep<br>1 = deep sleep. |
| [1] | SLEEPONEXIT | Indicates sleep-on-exit when returning from Handler mode to Thread mode:<br>0 = do not sleep when returning to Thread mode.<br>1 = enter sleep, or deep sleep, on return from an ISR to Thread mode.<br>Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application. |
| [0] | - | Reserved. |

#### 22.5.3.6 Configuration and Control Register

The CCR is a read-only register and indicates some aspects of the behavior of the Cortex-M0 processor. See the register summary in Table 22–341 for the CCR attributes.

The bit assignments are:

**Table 346. CCR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:10] | - | Reserved. |
| [9] | STKALIGN | Always reads as one, indicates 8-byte stack alignment on exception entry.<br>On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment. |
| [8:4] | - | Reserved. |
| [3] | UNALIGN_TRP | Always reads as one, indicates that all unaligned accesses generate a HardFault. |
| [2:0] | - | Reserved. |

#### 22.5.3.7 System Handler Priority Registers

The SHPR2-SHPR3 registers set the priority level, 0 to 3, of the exception handlers that have configurable priority.

SHPR2-SHPR3 are word accessible. See the register summary in Table 22–341 for their attributes.

To access to the system exception priority level using CMSIS, use the following CMSIS functions:

- `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
- `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`

The input parameter `IRQn` is the IRQ number, see Table 22–320 for more information.

The system fault handlers, and the priority field and register for each handler are:

**Table 347. System fault handler priority fields**

| Handler | Field | Register description |
|---|---|---|
| SVCall | PRI_11 | Section 22–22.5.3.7.1 |
| PendSV | PRI_14 | Section 22–22.5.3.7.2 |
| SysTick | PRI_15 | |

Each PRI_N field is 8 bits wide, but the processor implements only bits[7:6] of each field, and bits[5:0] read as zero and ignore writes.

##### 22.5.3.7.1 System Handler Priority Register 2

The bit assignments are:

**Table 348. SHPR2 register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | PRI_11 | Priority of system handler 11, SVCall |
| [23:0] | - | Reserved |

#### 22.5.3.7.2 System Handler Priority Register 3

The bit assignments are:

**Table 349. SHPR3 register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | PRI_15 | Priority of system handler 15, SysTick exception |
| [23:16] | PRI_14 | Priority of system handler 14, PendSV |
| [15:0] | - | Reserved |

### 22.5.3.8 SCB usage hints and tips

Ensure software uses aligned 32-bit word size transactions to access all the SCB registers.

## 22.5.4 System timer, SysTick

When enabled, the timer counts down from the current value (SYST_CVR) to zero, reloads (wraps) to the value in the SysTick Reload Value Register (SYST_RVR) on the next clock edge, then decrements on subsequent clocks. When the counter transitions to zero, the COUNTFLAG status bit is set to 1. The COUNTFLAG bit clears on reads.

**Remark:** The SYST_CVR value is UNKNOWN on reset. Software should write to the register to clear it to zero before enabling the feature. This ensures the timer will count from the SYST_RVR value rather than an arbitrary value when it is enabled.

**Remark:** If the SYST_RVR is zero, the timer will be maintained with a current value of zero after it is reloaded with this value. This mechanism can be used to disable the feature independently from the timer enable bit.

A write to the SYST_CVR will clear the register and the COUNTFLAG status bit. The write causes the SYST_CVR to reload from the SYST_RVR on the next timer clock, however, it does not trigger the SysTick exception logic. On a read, the current value is the value of the register at the time the register is accessed.

**Remark:** When the processor is halted for debugging the counter does not decrement.

The system timer registers are:

**Table 350. System timer registers summary**

| Address | Name | Type | Reset value | Description |
|---|---|---|---|---|
| 0xE000E010 | SYST_CSR | RW | 0x00000000 | Section 22.5.4.1 |
| 0xE000E014 | SYST_RVR | RW | Unknown | Section 22–22.5.4.2 |
| 0xE000E018 | SYST_CVR | RW | Unknown | Section 22–22.5.4.3 |
| 0xE000E01C | SYST_CALIB | RO | 0x00000004 [1] | Section 22–22.5.4.4 |

[1] SysTick calibration value.

#### 22.5.4.1 SysTick Control and Status Register

The SYST_CSR enables the SysTick features. See the register summary in  for its attributes. The bit assignments are:

**Table 351. SYST_CSR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:17] | - | Reserved. |
| [16] | COUNTFLAG | Returns 1 if timer counted to 0 since the last read of this register. |
| [15:3] | - | Reserved. |
| [2] | CLKSOURCE | Selects the SysTick timer clock source:<br>0 = external reference clock.<br>1 = processor clock. |
| [1] | TICKINT | Enables SysTick exception request:<br>0 = counting down to zero does not assert the SysTick exception request.<br>1 = counting down to zero asserts the SysTick exception request. |
| [0] | ENABLE | Enables the counter:<br>0 = counter disabled.<br>1 = counter enabled. |

#### 22.5.4.2 SysTick Reload Value Register

The SYST_RVR specifies the start value to load into the SYST_CVR. See the register summary in Table 22–350 for its attributes. The bit assignments are:

**Table 352. SYST_RVR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | - | Reserved. |
| [23:0] | RELOAD | Value to load into the SYST_CVR when the counter is enabled and when it reaches 0, see Section 22.5.4.2.1. |

##### 22.5.4.2.1 Calculating the RELOAD value

The RELOAD value can be any value in the range `0x00000001-0x00FFFFFF`. You can program a value of 0, but this has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

#### 22.5.4.3 SysTick Current Value Register

The SYST_CVR contains the current value of the SysTick counter. See the register summary in Table 22–350 for its attributes. The bit assignments are:

**Table 353. SYST_CVR bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31:24] | - | Reserved. |
| [23:0] | CURRENT | Reads return the current value of the SysTick counter.<br>A write of any value clears the field to 0, and also clears the SYST_CSR.COUNTFLAG bit to 0. |

#### 22.5.4.4 SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties. See the register summary in Table 22–350 for its attributes. The bit assignments are:

**Table 354. SYST_CALIB register bit assignments**

| Bits | Name | Function |
|---|---|---|
| [31] | NOREF | Reads as one. Indicates that no separate reference clock is provided. |
| [30] | SKEW | Reads as one. Calibration value for the 10ms inexact timing is not known because TENMS is not known. This can affect the suitability of SysTick as a software real time clock. |
| [29:24] | - | Reserved. |
| [23:0] | TENMS | Reads as zero. Indicates calibration value is not known. |

If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

#### 22.5.4.5 SysTick usage hints and tips

The interrupt controller clock updates the SysTick counter. If this clock signal is stopped for low power mode, the SysTick counter stops.

Ensure software uses word accesses to access the SysTick registers.

If the SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

## 22.6 Cortex-M0 instruction summary

**Table 355. Cortex M0- instruction summary**

| Operation | Description | Assembler | Cycles |
|---|---|---|---|
| Move | 8-bit immediate | MOVS Rd, #<imm> | 1 |
| | Lo to Lo | MOVS Rd, Rm | 1 |
| | Any to Any | MOV Rd, Rm | 1 |
| | Any to PC | MOV PC, Rm | 3 |
| Add | 3-bit immediate | ADDS Rd, Rn, #<imm> | 1 |
| | All registers Lo | ADDS Rd, Rn, Rm | 1 |
| | Any to Any | ADD Rd, Rd, Rm | 1 |
| | Any to PC | ADD PC, PC, Rm | 3 |

**Table 355. Cortex M0- instruction summary**

| Operation | Description | Assembler | Cycles |
|---|---|---|---|
| Add | 8-bit immediate | ADDS Rd, Rd, #<imm> | 1 |
| | With carry | ADCS Rd, Rd, Rm | 1 |
| | Immediate to SP | ADD SP, SP, #<imm> | 1 |
| | Form address from SP | ADD Rd, SP, #<imm> | 1 |
| | Form address from PC | ADR Rd, <label> | 1 |
| Subtract | Lo and Lo | SUBS Rd, Rn, Rm | 1 |
| | 3-bit immediate | SUBS Rd, Rn, #<imm> | 1 |
| | 8-bit immediate | SUBS Rd, Rd, #<imm> | 1 |
| | With carry | SBCS Rd, Rd, Rm | 1 |
| | Immediate from SP | SUB SP, SP, #<imm> | 1 |
| | Negate | RSBS Rd, Rn, #0 | 1 |
| Multiply | Multiply | MULS Rd, Rm, Rd | 1 |
| Compare | Compare | CMP Rn, Rm | 1 |
| | Negative | CMN Rn, Rm | 1 |
| | Immediate | CMP Rn, #<imm> | 1 |
| Logical | AND | ANDS Rd, Rd, Rm | 1 |
| | Exclusive OR | EORS Rd, Rd, Rm | 1 |
| | OR | ORRS Rd, Rd, Rm | 1 |
| | Bit clear | BICS Rd, Rd, Rm | 1 |
| | Move NOT | MVNS Rd, Rm | 1 |
| | AND test | TST Rn, Rm | 1 |
| Shift | Logical shift left by immediate | LSLS Rd, Rm, #<shift> | 1 |
| | Logical shift left by register | LSLS Rd, Rd, Rs | 1 |
| | Logical shift right by immediate | LSRS Rd, Rm, #<shift> | 1 |
| | Logical shift right by register | LSRS Rd, Rd, Rs | 1 |
| | Arithmetic shift right | ASRS Rd, Rm, #<shift> | 1 |
| | Arithmetic shift right by regist | ASRS Rd, Rd, Rs | 1 |
| Rotate | Rotate right by register | RORS Rd, Rd, Rs | 1 |
| Load | Word, immediate offset | LDR Rd, [Rn, #<imm>] | 2 |
| | Halfword, immediate offset | LDRH Rd, [Rn, #<imm>] | 2 |
| | Byte, immediate offset | LDRB Rd, [Rn, #<imm>] | 2 |
| | Word, register offset | LDR Rd, [Rn, Rm] | 2 |
| | Halfword, register offset | LDRH Rd, [Rn, Rm] | 2 |
| | Signed halfword, register offset | LDRSH Rd, [Rn, Rm] | 2 |
| | Byte, register offset | LDRB Rd, [Rn, Rm] | 2 |
| | Signed byte, register offset | LDRSB Rd, [Rn, Rm] | 2 |
| | PC-relative | LDR Rd, <label> | 2 |
| | SP-relative | LDR Rd, [SP, #<imm>] | 2 |
| | Multiple, excluding base | LDM Rn!, {<loreglist>} | 1 + N[1] |
| | Multiple, including base | LDM Rn, {<loreglist>} | 1 + N[1] |
| Store | Word, immediate offset | STR Rd, [Rn, #<imm>] | 2 |

**Table 355. Cortex M0- instruction summary**

| Operation | Description | Assembler | Cycles |
|---|---|---|---|
| Store | Halfword, immediate offset | STRH Rd, [Rn, #<imm>] | 2 |
| | Byte, immediate offset | STRB Rd, [Rn, #<imm>] | 2 |
| | Word, register offset | STR Rd, [Rn, Rm] | 2 |
| | Halfword, register offset | STRH Rd, [Rn, Rm] | 2 |
| | Byte, register offset | STRB Rd, [Rn, Rm] | 2 |
| | SP-relative | STR Rd, [SP, #<imm>] | 2 |
| | Multiple | STM Rn!, {<loreglist>} | 1 + N[1] |
| Push | Push | PUSH {<loreglist>} | 1 + N[1] |
| | Push with link register | PUSH {<loreglist>, LR} | 1 + N[1] |
| Pop | Pop | POP {<loreglist>} | 1 + N[1] |
| | Pop and return | POP {<loreglist>, PC} | 4 + N[2] |
| Branch | Conditional | B<cc> <label> | 1 or 3[3] |
| | Unconditional | B <label> | 3 |
| | With link | BL <label> | 4 |
| | With exchange | BX Rm | 3 |
| | With link and exchange | BLX Rm | 3 |
| Extend | Signed halfword to word | SXTH Rd, Rm | 1 |
| | Signed byte to word | SXTB Rd, Rm | 1 |
| | Unsigned halfword | UXTH Rd, Rm | 1 |
| | Unsigned byte | UXTB Rd, Rm | 1 |
| Reverse | Bytes in word | REV Rd, Rm | 1 |
| | Bytes in both halfwords | REV16 Rd, Rm | 1 |
| | Signed bottom half word | REVSH Rd, Rm | 1 |
| State change | Supervisor Call | SVC <imm> | -[4] |
| | Disable interrupts | CPSID i | 1 |
| | Enable interrupts | CPSIE i | 1 |
| | Read special register | MRS Rd, <specreg> | 4 |
| | Write special register | MSR <specreg>, Rn | 4 |
| Hint | Send event | SEV | 1 |
| | Wait for event | WFE | 2[5] |
| | Wait for interrupt | WFI | 2[5] |
| | Yield | YIELD[6] | 1 |
| | No operation | NOP | 1 |
| Barriers | Instruction synchronization | ISB | 4 |
| | Data memory | DMB | 4 |
| | Data synchronization | DSB | 4 |

[1] N is the number of elements.

[2] N is the number of elements in the stack-pop list including PC and assumes load or store does not generate a HardFault exception.

[3] 3 if taken, 1 if not taken.

[4] Cycle count depends on core and debug configuration.

[5] Excludes time spend waiting for an interrupt or event.

[6] Executes as NOP.

## 23.1 Abbreviations

**Table 356. Abbreviations**

| Acronym | Description |
|---|---|
| ADC | Analog-to-Digital Converter |
| AHB | Advanced High-performance Bus |
| AMBA | Advanced Microcontroller Bus Architecture |
| APB | Advanced Peripheral Bus |
| BOD | BrownOut Detection |
| GPIO | General Purpose Input/Output |
| PLL | Phase-Locked Loop |
| SPI | Serial Peripheral Interface |
| SSI | Serial Synchronous Interface |
| TTL | Transistor-Transistor Logic |
| UART | Universal Asynchronous Receiver/Transmitter |

## 23.2 References

**[1]** **ARM DUI 0497A —** Cortex-M0 Devices Generic User Guide

**[2]** **ARM DDI 0432C —** Cortex-M0 Revision r0p0 Technical Reference Manual

# 23.3 Legal information

## 23.3.1 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.

## 23.3.2 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.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

**Right to make changes —** NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use —** NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications —** Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control —** This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

## 23.3.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

**$I^2C$-bus —** logo is a trademark of NXP B.V.

## 23.4 Tables

## 23.5 Figures

## 23.6 Contents

### Chapter 1: LPC111x/LPC11Cxx Introductory information



### Chapter 2: LPC111x/LPC11Cxx Memory mapping



### Chapter 3: LPC111x/LPC11Cxx System configuration (SYSCON)



### Chapter 4: LPC111x/LPC11Cxx Power Monitor Unit (PMU)

## Chapter 5: LPC111x/LPC11Cxx Power profiles



## Chapter 6: LPC111x/LPC11Cxx Nested Vectored Interrupt Controller (NVIC)



## Chapter 7: LPC111x/LPC11Cxx I/O configuration (IOCONFIG)

## Chapter 8: LPC111x/LPC11Cxx Pin configuration



## Chapter 9: LPC111x/LPC11Cxx General Purpose I/O (GPIO)



## Chapter 10: LPC111x/LPC11Cxx UART



## Chapter 11: LPC111x/LPC11Cxx SPI0/1 with SSP

## Chapter 12: LPC111x/LPC11Cxx I2C-bus controller

## Chapter 13: LPC111x/LPC11Cxx C_CAN controller



## Chapter 14: LPC11Cxx C_CAN on-chip drivers

## Chapter 15: LPC111x/LPC11Cxx 16-bit counter/timer CT16B0/1



## Chapter 16: LPC111x/LPC11Cxx 32-bit counter/timer CT32B0/1



## Chapter 17: LPC111x/LPC11Cxx WatchDog Timer (WDT)

## Chapter 18: LPC111x/LPC11Cxx System tick timer (SysTick)



## Chapter 19: LPC111x/LPC11Cxx ADC



## Chapter 20: LPC111x/LPC11Cxx Flash programming firmware

## Chapter 21: LPC111x/LPC11Cxx Serial Wire Debug (SWD)



## Chapter 22: LPC111x/LPC11Cxx Appendix: ARM Cortex-M0 reference

## Chapter 23: 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: 4 March 2011**
**Document identifier: UM10398**