# UM10316

## LPC29xx User manual

Rev. 3 — 19 October 2010 User manual

## Document information

| Info | Content |
|---|---|
| Keywords | LPC2917/01; LPC2919/01; LPC2926; LPC2927; LPC2929; LPC2921; LPC2923; LPC2925; LPC2930; LPC2939 User Manual, ARM9, CAN, LIN, USB |
| Abstract | This document extends the LPC29xx data sheets with additional details to support both hardware and software development. It focuses on functional description and typical application use. |

## Revision history

| Rev | Date | Description |
|---|---|---|
| 3 | 20101019 | |
| Modifications: | | • Editorial updates to CGU, PMU, VIC, USB OTG, and Flash/EEPROM chapters. |
| | | • Table 103 “LPC2930 boot configuration” added. |
| | | • Part LPC2926 added. |
| | | • Editorial updates. |
| | | • Description of USBCmdCode register updated. |
| | | • Description of CMD bit in Set Device Status register updated. |
| 2 | 20091116 | |
| Modifications: | | • LIN chapter updated with description and register map of the UART mode. |
| | | • Corrected description of NMMEN bit (bit 0) in the RS485CTRL register: 0 = RS-485 mode disabled. 1 = RS-485 mode enabled. Default value: 0. |
| | | • LPC2930/39 pin configuration table: pin 29 description corrected; P1[31] changed to P1[30]. |
| | | • LPC2930/39 pin configuration table: $\overline{\text{USB\_OVRCR1}}$ function added to pin 31. |
| | | • Connection of ADC inputs to ADC channels corrected. |
| | | • UART function updated: The fractional divider can not be used in auto-baud mode. |
| | | • Editorial updates. |
| 1 | 20090522 | Initial LPC29xx user manual edition |

## Contact information

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

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

# UM10316

# Chapter 1: LPC29xx Introductory information

Rev. 3 — 19 October 2010 User manual

## 1. Introduction

The LPC29xx combine an 125 MHz ARM968E-S CPU core, Full Speed USB 2.0 OTG and device, CAN and LIN, up to 56 kB SRAM, up to 768 kB flash memory, external memory interface, two or three 10-bit ADCs, and multiple serial and parallel interfaces in a single chip targeted at consumer, industrial, medical, and communication. To optimize system power consumption, the LPC29xx has a very flexible Clock Generation Unit (CGU) that provides dynamic clock gating and scaling.

## 2. About this user manual

This document describes the following parts: LPC2917/01, LPC2919/01, LPC2921, LPC2923, LPC2925, LPC2926, LPC2927, LPC2929, LPC2930, and LPC2939. Differences between the various parts as they apply to each block or peripheral are listed at the beginning of each chapter. For an overview of features see Table 1–2.

## 3. General features

**Remark:** See Table 1–2 for feature details for each LPC29xx part.

- ARM968E-S processor running at frequencies of up to 125 MHz maximum.
- Multi-layer AHB system bus at 125 MHz with four separate layers.
- On-chip memory:
  - Two Tightly Coupled Memories (TCM), up to 32 kB Instruction (ITCM), up to 32 kB Data TCM (DTCM).
  - Two separate internal Static RAM (SRAM) instances; 32 kB SRAM and 16 kB SRAM.
  - 8 kB ETB SRAM.
  - Up to 768 kB flash-program memory with 16 kB EEPROM.
- Dual-master, eight-channel GPDMA controller on the AHB multilayer matrix which can be used with the SPI interfaces and the UARTs, as well as for memory-to-memory transfers including the TCM memories.
- External Static Memory Controller (SMC) with eight memory banks; up to 32-bit data bus; up to 24-bit address bus.
- Serial interfaces:
  - USB 2.0 full-speed device/OTG controller with dedicated DMA controller and on-chip PHY for device and Host (LPC2930/39 only) functions.
  - Two-channel CAN controller supporting Full-CAN and extensive message filtering
  - Two LIN master controllers with full hardware support for LIN communication.
  - Two 550 UARTs with 16-byte Tx and Rx FIFO depths, DMA support, and RS485 support.

  - Three full-duplex Q-SPIs with four slave-select lines; 16 bits wide; 8 locations deep; Tx FIFO and Rx FIFO.
  - Two I$^2$C-bus interfaces.
- Other peripherals:
  - Up to three ADCs: Two 10-bit ADCs, 8-channels each, with 3.3 V measurement range and one, 8-channel 10-bit ADC with 5.0 V measurement range provide a total of up to 24 analog inputs, with conversion times as low as 2.44 μs per channel. Each channel provides a compare function to minimize interrupts.
  - Multiple trigger-start option for all ADCs: timer, PWM, other ADC and external signal input.
  - Four 32-bit timers each containing four capture-and-compare registers linked to I/Os.
  - Four six-channel PWMs (Pulse-Width Modulators) with capture and trap functionality.
  - Two dedicated 32-bit timers to schedule and synchronize PWM and ADC.
  - Quadrature encoder interface that can monitor one external quadrature encoder.
  - 32-bit watchdog with timer change protection, running on safe clock.
- Up to 108 general-purpose I/O pins with programmable pull-up, pull-down, or bus keeper.
- Vectored Interrupt Controller (VIC) with 16 priority levels.
- Up to 24 level-sensitive external interrupt pins, including CAN and LIN wake-up features.
- Configurable clock-out pin for driving external system clocks.
- Processor wake-up from power-down via external interrupt pins; CAN or LIN activity.
- Flexible Reset Generator Unit (RGU) able to control resets of individual modules.
- Flexible Clock-Generation Unit (CGU) able to control clock frequency of individual modules:
  - On-chip very low-power ring oscillator; fixed frequency of 0.4 MHz; always on to provide a Safe_Clock source for system monitoring.
  - On-chip crystal oscillator with a recommended operating range from 10 MHz to 25 MHz - max. PLL input 25 MHz.
  - On-chip PLL allows CPU operation up to a maximum CPU rate of 125 MHz.
  - Generation of up to 11 base clocks.
  - Seven fractional dividers.
- Highly configurable system Power Management Unit (PMU):

  clock control of individual modules.

  allows minimization of system operating power consumption in any configuration.
- Standard ARM test and debug interface with real-time in-circuit emulator.
- Boundary-scan test supported.
- ETM/ETB debug functions with 8 kB of dedicated SRAM also accessible for application code and data storage.
- Dual power supply:

  - CPU operating voltage: 1.8 V ± 5 %.
  - I/O operating voltage: 2.7 V to 3.6 V; inputs tolerant up to 5.5 V.
- Available in 100-pin,144-pin, and 208-pin LQFP packages.
- –40 °C to 85 °C ambient operating temperature range.

## 4. Ordering information

**Table 1. Ordering information**

| Type number | Package | | |
|---|---|---|---|
| | Name | Description | Version |
| LPC2917FBD144/01 | LQFP144 | plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm | SOT486-1 |
| LPC2919FBD144/01 | LQFP144 | plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm | SOT486-1 |
| LPC2921FBD100 | LQFP100 | plastic low profile quad flat package; 100 leads; body 14 x 14 x 1.4 mm | SOT407-1 |
| LPC2923FBD100 | LQFP100 | plastic low profile quad flat package; 100 leads; body 14 x 14 x 1.4 mm | SOT407-1 |
| LPC2925FBD100 | LQFP100 | plastic low profile quad flat package; 100 leads; body 14 x 14 x 1.4 mm | SOT407-1 |
| LPC2926FBD144 | LQFP144 | plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm | SOT486-1 |
| LPC2927FBD144 | LQFP144 | plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm | SOT486-1 |
| LPC2929FBD144 | LQFP144 | plastic low profile quad flat package; 144 leads; body 20 × 20 × 1.4 mm | SOT486-1 |
| LPC2930FBD208 | LQFP208 | plastic low profile quad flat package; 208 leads; body 28 x 28 x 1.4 mm | SOT459-1 |
| LPC2939FBD208 | LQFP208 | plastic low profile quad flat package; 208 leads; body 28 x 28 x 1.4 mm | SOT459-1 |

## 4.1 Ordering options

Table 2. LPC29xx part overview

| Part number | Flash | SRAM (incl ETB) | TCM (I/D) | CAN | LIN/ UART | Pins | UART RS485 | SPI | 3V3 ADC | 5 V ADC | SMC | USB device | USB host | USB OTG | ETM | QEI | I2C-bus | Clkout pin |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPC2939 | 768 kB | 56 kB | 32/32 kB | 2 | 2 | 208 | 2 | 3 | 2 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| LPC2930 | - | 56 kB | 32/32 kB | 2 | 2 | 208 | 2 | 3 | 2 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| LPC2929 | 768 kB | 56 kB | 32/32 kB | 2 | 2 | 144 | 2 | 3 | 2 | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes |
| LPC2927 | 512 kB | 56 kB | 32/32 kB | 2 | 2 | 144 | 2 | 3 | 2 | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes |
| LPC2926 | 256 kB | 56 kB | 32/32 kB | 2 | 2 | 144 | 2 | 3 | 2 | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes |
| LPC2925 | 512 kB | 40 kB | 16/16 kB | 2 | 2 | 100 | 2 | 3 | 2 | No | No | Yes | No | No | Yes | Yes | Yes | Yes |
| LPC2923 | 256 kB | 24 kB | 16/16 kB | 2 | 2 | 100 | 2 | 3 | 2 | No | No | Yes | No | No | Yes | Yes | Yes | Yes |
| LPC2921 | 128 kB | 24 kB | 16/16 kB | 2 | 2 | 100 | 2 | 3 | 2 | No | No | Yes | No | No | Yes | Yes | Yes | Yes |
| LPC2919/01 | 768 kB | 56 kB | 16/16 kB | 2 | 2 | 144 | 2 | 3 | 2 | No | Yes | No | No | No | Yes | Yes | Yes | Yes |
| LPC2917/01 | 512 kB | 56 kB | 16/16 kB | 2 | 2 | 144 | 2 | 3 | 2 | No | Yes | No | No | No | Yes | Yes | Yes | Yes |

**Remark:** Note that parts LPC2927 and LPC2929 are not fully pin compatible with parts LPC2917/01 and LPC2919/01 or LPC2917 and LPC2919. On the LPC2927/29 the MSCSS and timer blocks have a reduced pinout.

## 4.2 Comparison with LPC2917/19 devices

Table 3. Feature comparison

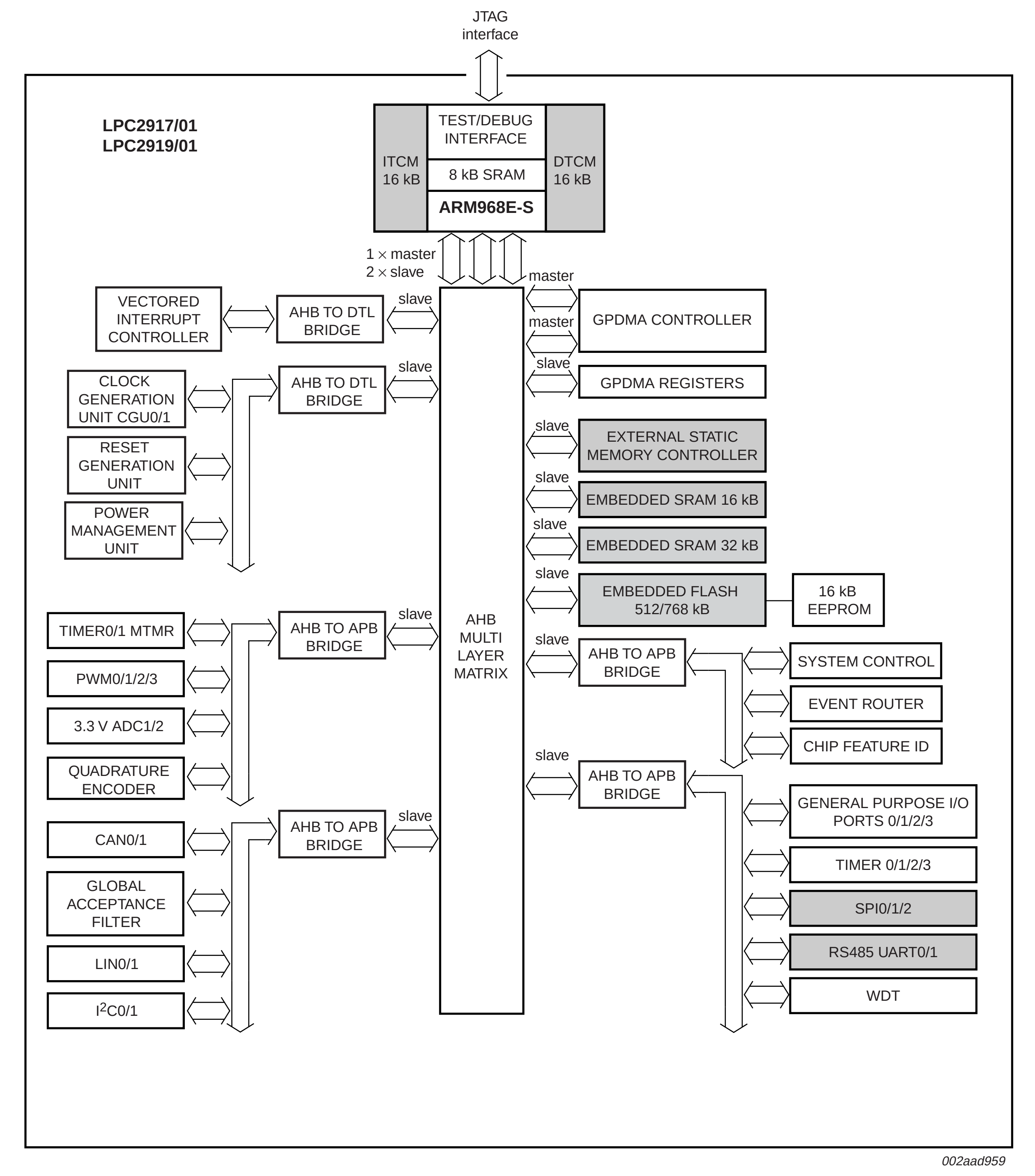
| Parts | GPDMA | UART RS485 mode | I2C | QEI | CAN | LIN | USB OTG/ device | Flash | EEPROM | SRAM total | ETB SRAM | 5V ADC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPC2917/19 | no | no | no | no | 2 | 2 | no | 512/768 kB | no | 80 kB | no | no |
| LPC2917/19/01 | yes | yes | yes | yes | 2 | 2 | no | 512/768 kB | yes | 88 kB | 8 kB | no |
| LPC2927/29 | yes | yes | yes | yes | 2 | 2 | yes | 512/768 kB | yes | 120 kB | 8 kB | yes |

### 4.3 LPC29xx USB options

Table 4. LPC29xx USB options

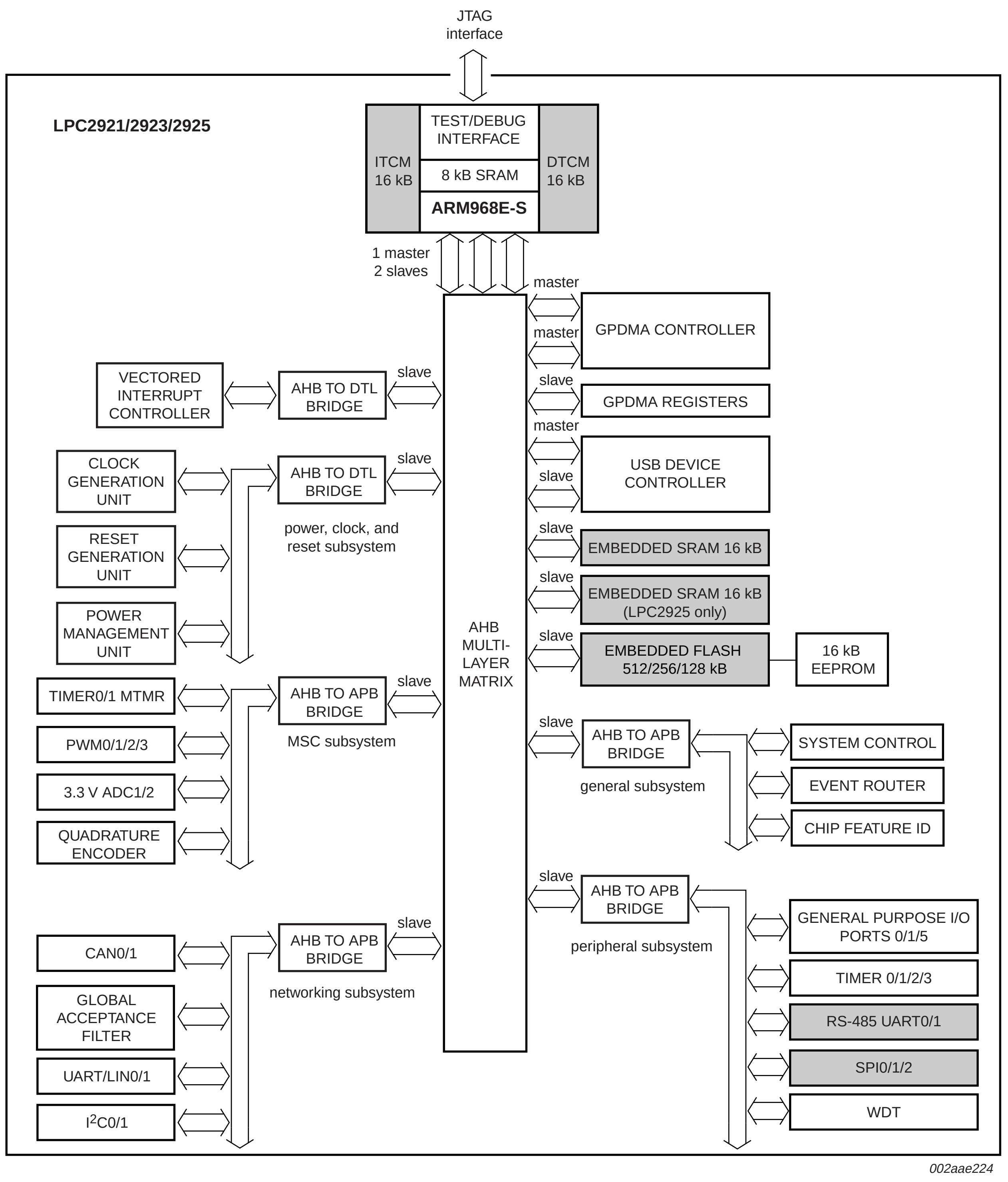
| Part | On-chip controller | | | Ports | On-chip PHY |
|---|---|---|---|---|---|
| | Device | Host | OTG | | |
| LPC2917/01 | - | - | - | - | - |
| LPC2919/01 | - | - | - | - | - |
| LPC2921 | Full-speed | - | - | 1 | Device |
| LPC2923 | Full-speed | - | - | 1 | Device |
| LPC2925 | Full-speed | - | - | 1 | Device |
| LPC2926 | Full-speed | - | Full-speed | 1 | Device |
| LPC2927 | Full-speed | - | Full-speed | 1 | Device |
| LPC2929 | Full-speed | - | Full-speed | 1 | Device |
| LPC2930 | Full-speed | Full-speed | Full-speed | 2 | Device, host |
| LPC2939 | Full-speed | Full-speed | Full-speed | 2 | Device, host |

## 5. Block diagram



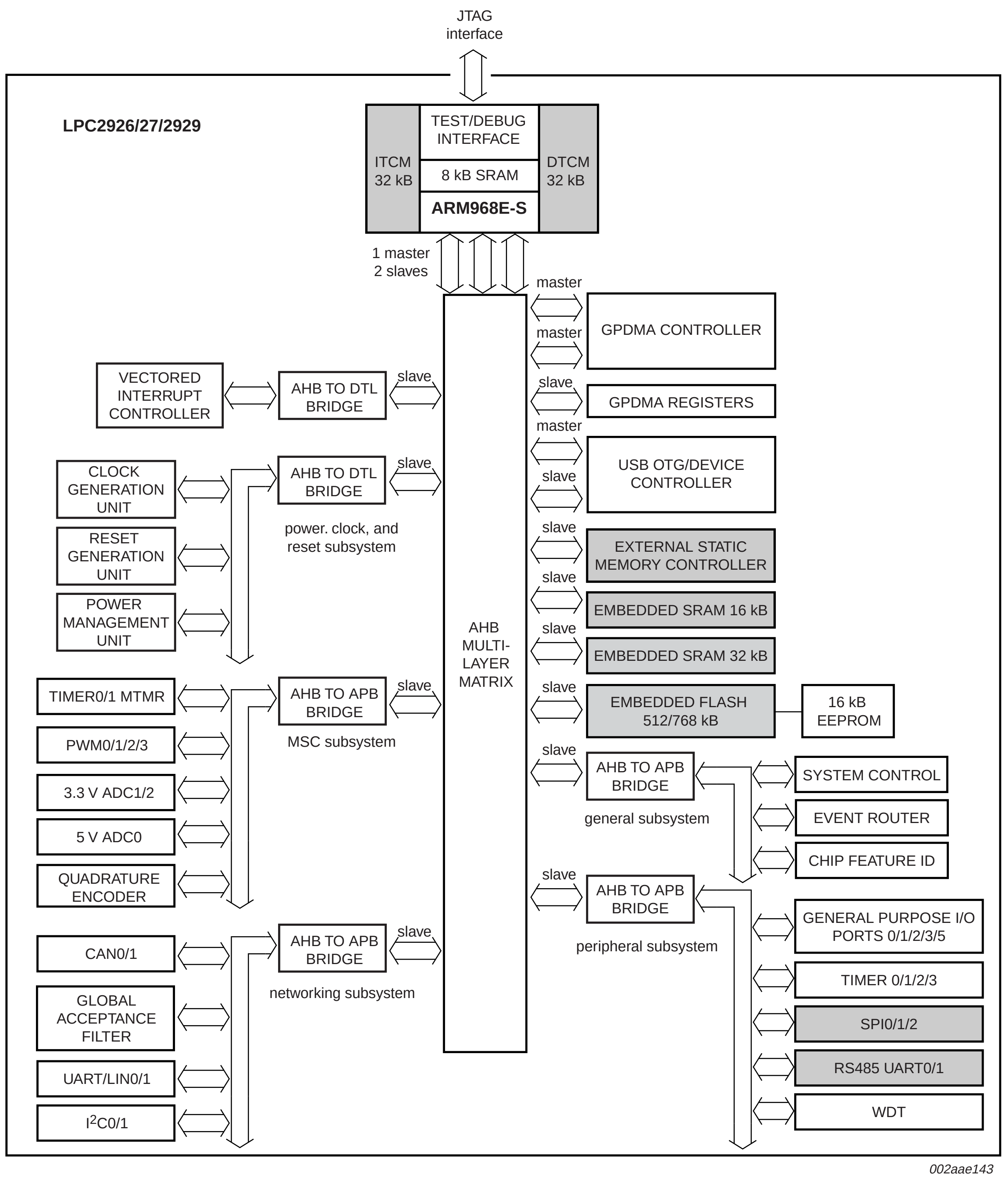
Grey-shaded blocks represent peripherals with connections to the GPDMA.

**Fig 1. LPC2917/19/01 block diagram**



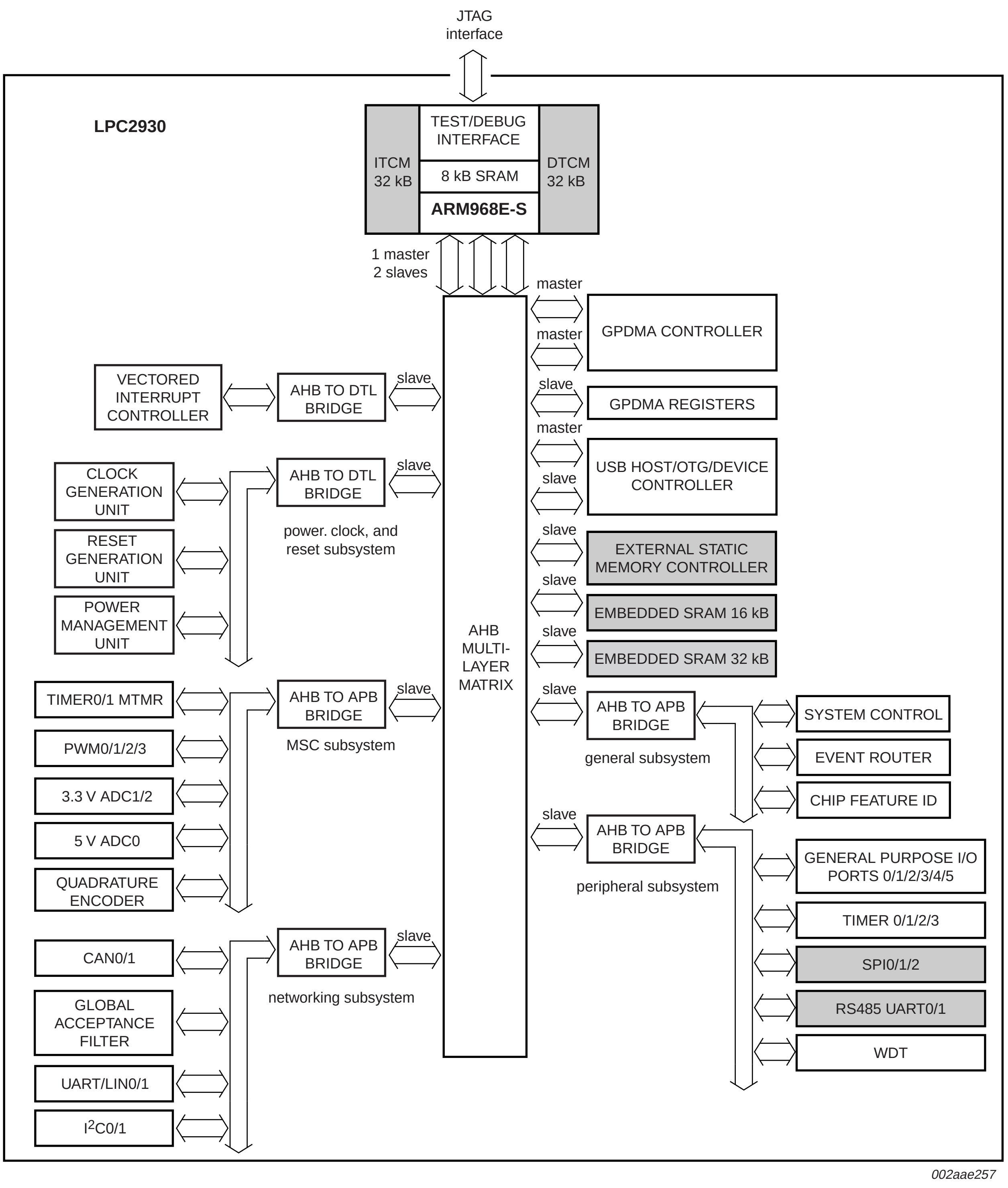
Grey-shaded blocks represent peripherals with connections to the GPDMA.

**Fig 2. LPC2921/23/25 block diagram**



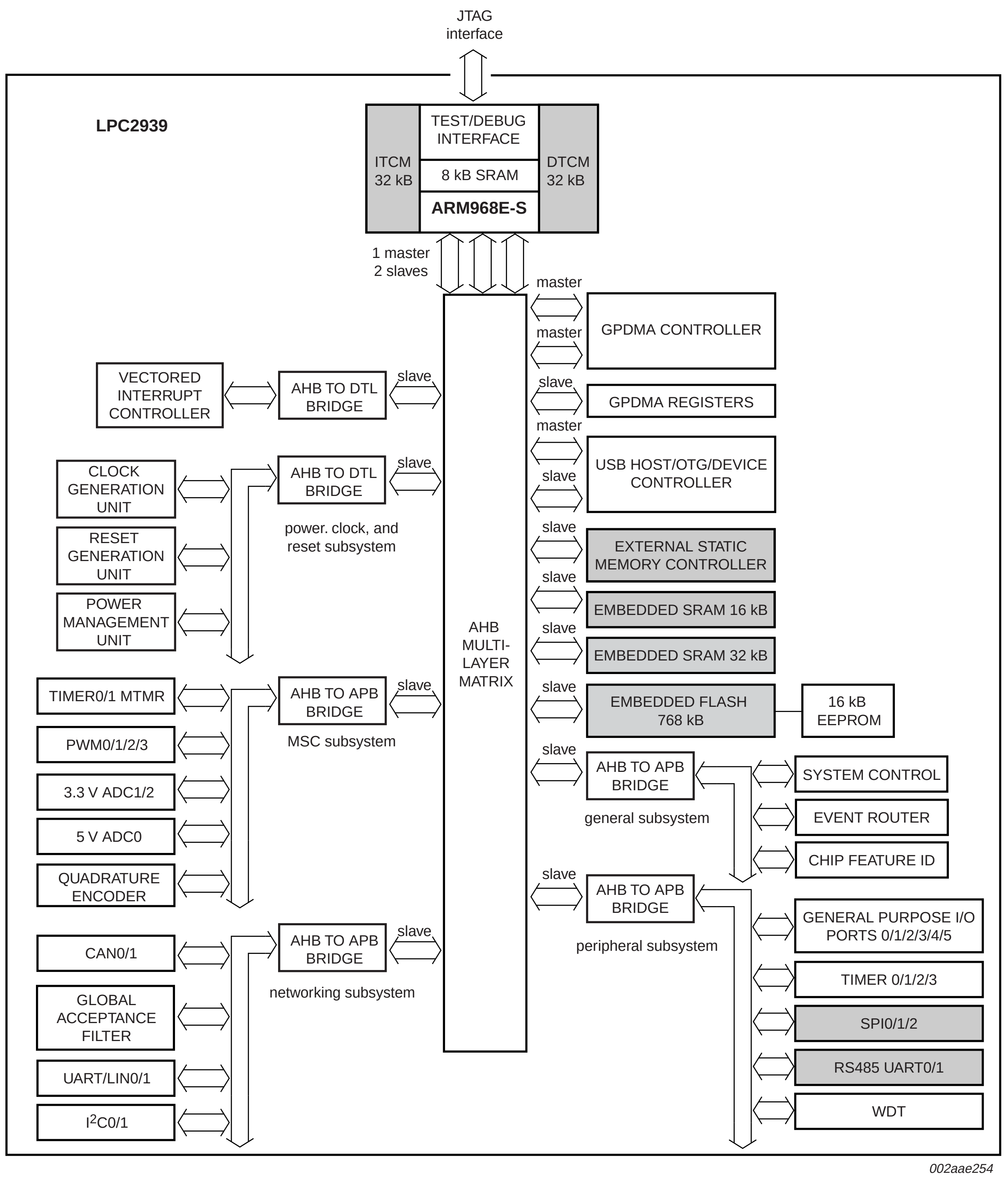
Grey-shaded blocks represent peripherals with connections to the GPDMA.

**Fig 3. LPC2926/27/29 block diagram**



Grey-shaded blocks represent peripherals with connections to the GPDMA.

**Fig 4. LPC2930 block diagram**



Grey-shaded blocks represent peripherals with connections to the GPDMA.

**Fig 5. LPC2939 block diagram**

## 6. Functional blocks

This chapter gives an overview of the functional blocks, clock domains, and power modes. See Table 1–2 for availability of peripherals and blocks for specific LPC29xx parts.

The functional blocks are explained in detail in the following chapters. Several blocks are gathered into subsystems and one or more of these blocks and/or subsystems are put into a clock domain. Each of these clock domains can be configured individually for power management (i.e. clock on or off and whether the clock responds to sleep and wake-up events).

Table 5. Functional blocks and clock domains

| Short | Description | Comment |
|---|---|---|
| **Clock domain AHB** | | |
| ARM | ARM9TDMI-S | 32-bit RISC processor |
| SMC | Static Memory Controller | For external (static) memory banks |
| SRAM | Internal Static Memory | - |
| **Clock domain Flash** | | |
| Flash | - | Internal Flash Memory |
| FMC | Flash Memory Controller | Controller for the internal flash memory |
| **Clock domain USB** | | |
| USB | USB OTG controller | - |
| **Clock domain DMA controller** | | |
| GPDMA | General Purpose DMA controller | |
| **Clock domain VIC** | | |
| VIC | Vectored Interrupt Controller | Prioritized/vectored interrupt handling |
| **Clock domain general subsystem** | | |
| CFID | Digital Chip ID | Identifies the device and its possibilities |
| ER | Event Router | Routes wake-up events and external interrupts (to CGU/VIC) |
| SCU | System Control Unit | Configures memory map and I/O functions |
| **Clock domain peripheral subsystem** | | |
| GPIO | General-Purpose Input/Output | Directly controls I/O pins |
| TMR | Timer | Provides match output and capture inputs |
| UART | Universal Asynchronous Receiver/Transmitter | Standard 550 serial port |
| WDT | Watchdog | Timer to guard (software) execution |
| SPI | Serial Peripheral Interface | Supports various industry-standard SPI protocols |
| **Clock domain modulation and sampling-control subsystem** | | |
| ADC | Analog-to-Digital Converter | 10-bit Analog-to-Digital Converter |
| PWM | Pulse-Width Modulator | Synchronized Pulse-Width Modulator |

Table 5. **Functional blocks and clock domains** *…continued*

| Short | Description | Comment |
|---|---|---|
| TMR | Timer | Dedicated Sampling and Control Timer |
| QEI | Quadrature encoder interface | - |
| **Clock domain networking subsystem** | | |
| CAN | Gateway | Includes acceptance filter |
| LIN | Master controller | LIN master controller |
| I2C | I2C-bus | |
| **Clock domain power control subsystem** | | |
| CGU0 | Clock Generation Unit | Controls clock sources and clock domains |
| CGU1 | clock generation unit | USB clocks and clock out |
| RGU | reset generation unit | - |
| PMU | power management unit | - |

## 7. Architectural overview

The LPC29xx consists of:

- An ARM968E-S processor with real-time emulation support
- An AMBA multi-layer Advanced High-performance Bus (AHB) for interfacing to the on-chip memory controllers
- Two DTL buses (an universal NXP interface) for interfacing to the interrupt controller and the Power, Clock and Reset Control cluster (also called subsystem).
- Three ARM Peripheral Buses (APB - a compatible superset of ARM's AMBA advanced peripheral bus) for connection to on-chip peripherals clustered in subsystems.
- One ARM Peripheral Bus for event router and system control.

The LPC29xx configures the ARM968E-S processor in little-endian byte order. All peripherals run at their own clock frequency to optimize the total system power consumption. The AHB2APB bridge used in the subsystems contains a write-ahead buffer one transaction deep. This implies that when the ARM968E-S issues a buffered write action to a register located on the APB side of the bridge, it continues even though the actual write may not yet have taken place. Completion of a second write to the same subsystem will not be executed until the first write is finished.

## 8. ARM968E-S processor

The ARM968E-S is a general purpose 32-bit RISC processor, which offers high performance and very low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than those of microprogrammed Complex Instruction Set Computers (CISC). This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective controller core.

Amongst the most compelling features of the ARM968E-S are:

- Separate directly connected instruction and data Tightly Coupled Memory (TCM) interfaces
- Write buffers for the AHB and TCM buses
- Enhanced 16 × 32 multiplier capable of single-cycle MAC operations and 16-bit fixed-point DSP instructions to accelerate signal-processing algorithms and applications.

Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. The ARM968E-S is based on the ARMv5TE five-stage pipeline architecture. Typically, in a three-stage pipeline architecture, while one instruction is being executed its successor is being decoded and a third instruction is being fetched from memory. In the five-stage pipeline additional stages are added for memory access and write-back cycles.

The ARM968E-S processor also employs a unique architectural strategy known as THUMB, which makes it ideally suited to high-volume applications with memory restrictions or to applications where code density is an issue.

The key idea behind THUMB is that of a super-reduced instruction set. Essentially, the ARM968E-S processor has two instruction sets:

- Standard 32-bit ARMv5TE set
- 16-bit THUMB set

The THUMB set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM's performance advantage over a traditional 16-bit controller using 16-bit registers. This is possible because THUMB code operates on the same 32-bit register set as ARM code.

THUMB code can provide up to 65 % of the code size of ARM, and 160 % of the performance of an equivalent ARM controller connected to a 16-bit memory system.

The ARM968E-S processor is described in detail in the ARM968E-S data sheet.

## 9. On-chip flash memory system

The LPC29xx includes up to 768 kB flash memory. This memory can be used for both code and data storage. Flash memory can be programmed in-system via a serial port (e.g., CAN).

## 10. On-chip static RAM

In addition to the two 16 kB or 32 kB TCMs, the LPC29xx includes two static RAM memories: one of up to 32 kB and one of 16 kB. Both may be used for code and/or data storage.

# UM10316

# Chapter 2: LPC29xx memory mapping

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The memory configuration varies for the different LPC29xx parts (see Table 2–6). In addition to the memory blocks, peripheral register blocks in memory region 7 are available only if the peripheral is implemented. See Table 2–7 for part specific registers. All other peripheral registers are available on all LPC29xx parts.

Table 6. LPC29xx memory configurations

| Part number | Flash | SRAM | | | TCM (I/D) | SMC |
|---|---|---|---|---|---|---|
| | | SRAM @ 0x8000 0000 | SRAM @ 0x8000 8000 | ETB (8 kB) | | |
| LPC2917/01 | 512 kB | 32 kB | 16 kB | yes | 16/16 kB | 8 banks, 16 MB each |
| LPC2919/01 | 768 kB | 32 kB | 16 kB | yes | 16/16 kB | 8 banks, 16 MB each |
| LPC2921 | 128 kB | 16 kB | - | yes | 16/16 kB | - |
| LPC2923 | 256 kB | 16 kB | - | yes | 16/16 kB | - |
| LPC2925 | 512 kB | 16 kB | 16 kB | yes | 16/16 kB | - |
| LPC2926 | 256 kB | 32 kB | 16 kB | yes | 32/32 kB | 8 banks, 16 MB each |
| LPC2927 | 512 kB | 32 kB | 16 kB | yes | 32/32 kB | 8 banks, 16 MB each |
| LPC2929 | 768 kB | 32 kB | 16 kB | yes | 32/32 kB | 8 banks, 16 MB each |
| LPC2930 | - | 32 kB | 16 kB | yes | 32/32 kB | 8 banks, 16 MB each |
| LPC2939 | 768 kB | 32 kB | 16 kB | yes | 32/32 kB | 8 banks, 16 MB each |

Table 7. LPC29xx configuration of peripheral registers

| Part number | peripheral cluster #2 | peripheral cluster #6 | | | AHB peripherals |
|---|---|---|---|---|---|
| | GPIO | ADC0 | ADC1 | ADC2 | USB |
| LPC291719/01 | GPIO0/1/2/3 | no | yes | yes | no |
| LPC2919/01 | GPIO0/1/2/3 | no | yes | yes | no |
| LPC2921 | GPIO0/1/5 | no | yes | yes | yes |
| LPC2923 | GPIO0/1/5 | no | yes | yes | yes |
| LPC2925 | GPIO0/1/5 | no | yes | yes | yes |
| LPC2927 | GPIO0/1/2/3/5 | yes | yes | yes | yes |
| LPC2929 | GPIO0/1/2/3/5 | yes | yes | yes | yes |
| LPC2930 | GPIO0/1/2/3/4/5 | yes | yes | yes | yes |
| LPC2939 | GPIO0/1/2/3/4/5 | yes | yes | yes | yes |

## 2. Memory-map view of the AHB

The LPC29xx uses an AHB multilayer bus with the CPU and the GPDMA as the bus masters. The AHB slaves are connected to the AHB-lite multilayer bus.The ARM968E-S CPU has access to all AHB slaves and hence to all address regions.

## 3. Memory-map regions

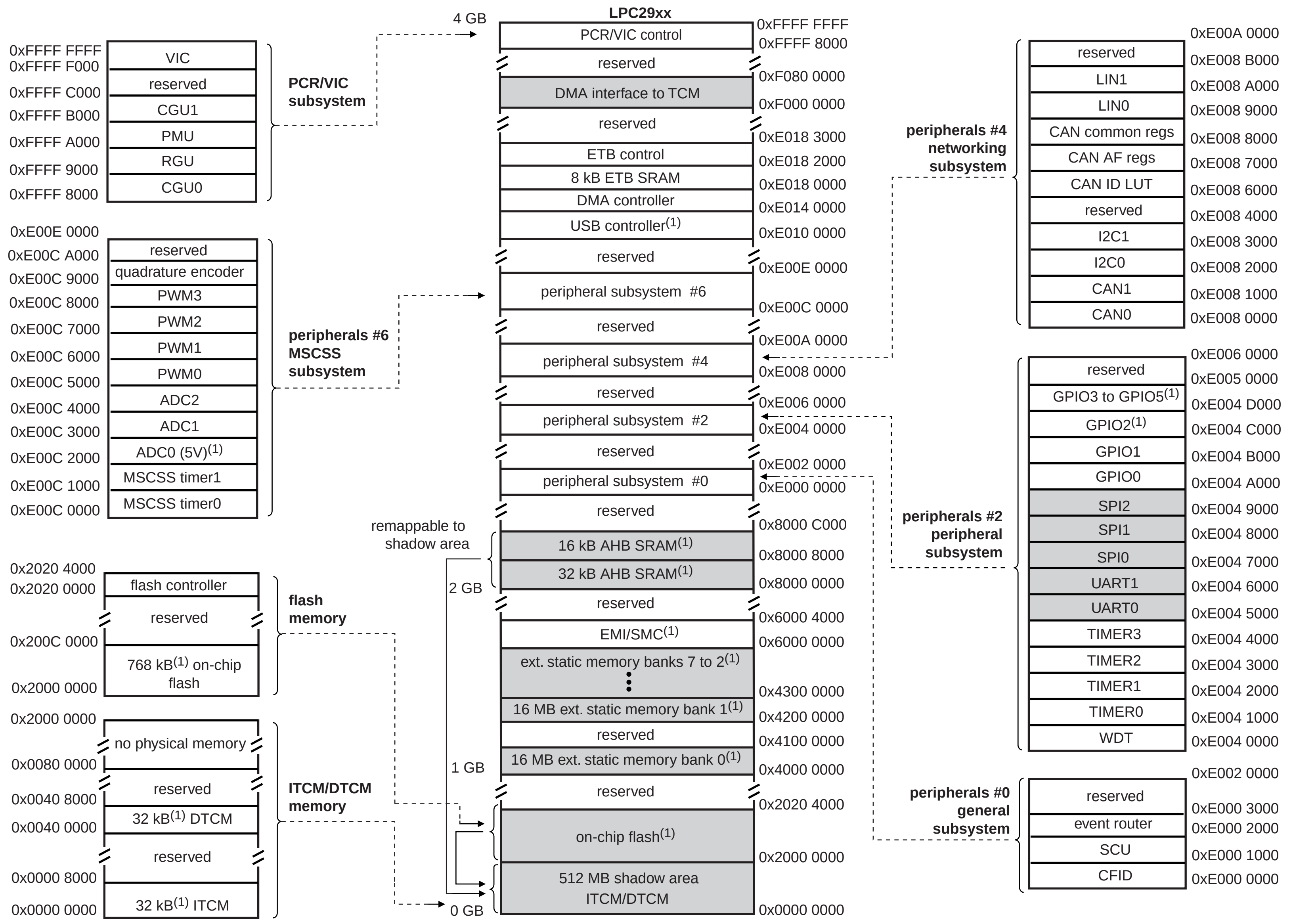
The ARM9 processor has a 4 GB of address space. The LPC29xx has divided this memory space into eight regions of 512 MB each. Each region is used for a dedicated purpose.

An exception to this is region 0; several of the other regions (or a part of it) can be shadowed in the memory map at this region. This shadowing can be controlled by software via the programmable re-mapping registers (see Table 6–64).

**Table 8. LPC29xx memory regions**

| Memory region # | Address | Description |
|---|---|---|
| 0 | 0x0000 0000 | TCM area and shadow area |
| 1 | 0x2000 0000 | embedded flash area |
| 2 | 0x4000 0000 | external static memory area |
| 3 | 0x6000 0000 | external static memory controller area |
| 4 | 0x8000 0000 | internal SRAM area |
| 5 | 0xA000 0000 | not used |
| 6 | 0xC000 0000 | not used |
| 7 | 0xE000 0000 | bus-peripherals area |

Figure 2–6 gives a graphical overview of the LPC29xx memory map.



(1) See Section 2–1 for part-specific implementation. Gray-shaded memory regions are accessible by the GPDMA controller.

**Fig 6. LPC29xx system memory map: graphical overview**

## 3.1 Region 0: TCM/shadow area

The ARM968E-S processor has its exception vectors located at address logic 0. Since flash is the only non-volatile memory available in the LPC29xx, the exception vectors in the flash must be located at address logic 0 at reset (AHB_RST).

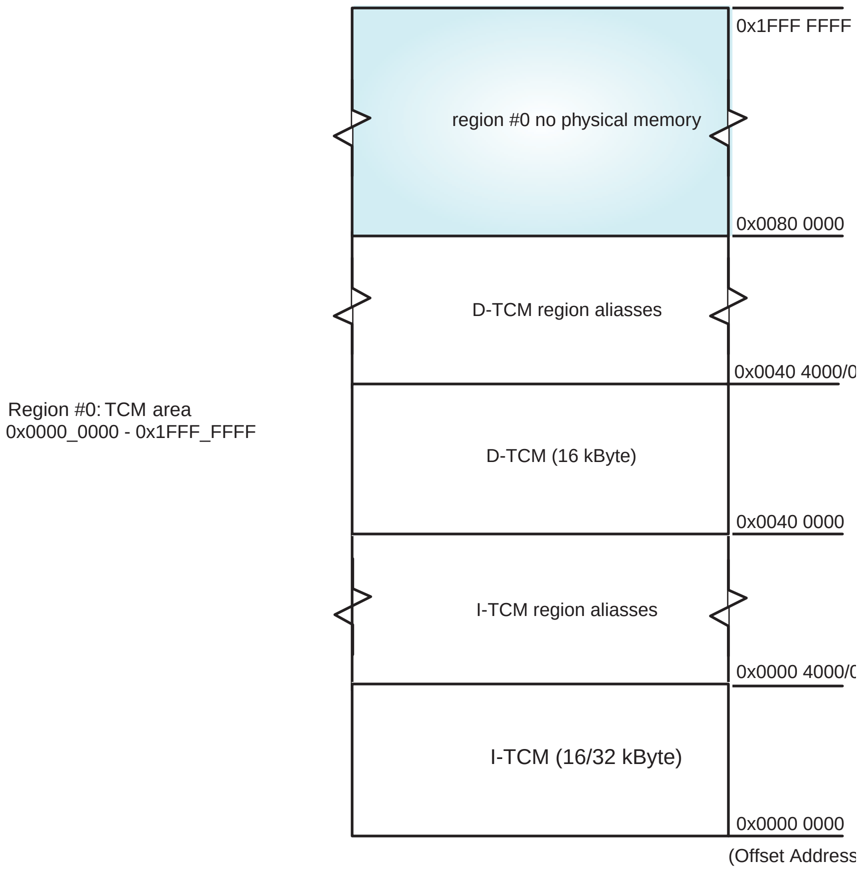


**Fig 7. Region 0 memory map**

After booting a choice must be made for region 0. When enabled, the Tightly Coupled Memories (TCMs) occupy fixed address locations in region 0 as indicated in Figure 2–6. Information on how to enable the TCMs can be found in the ARM documentation, see Ref. 31–2.

To enable memory re-mapping, the LPC29xx AHB system memory map provides a shadow area (region 0) starting at address logic 0. This is a virtual memory region, i.e. no actual memory is present at the shadow area addresses. A selectable region of the AHB system memory map is, apart from its own specific region, also accessible via this shadow area region.

After reset, the region 1 embedded flash area is always available at the shadow area. After booting, any other region of the AHB system memory map (e.g. internal SRAM) can be re-mapped to region 0 by means of the shadow memory mapping register. For more details about the shadow area see Table 6–64.

### 3.2 Region 1: embedded flash area

Figure 2–8 gives a graphical overview of the embedded flash memory map.

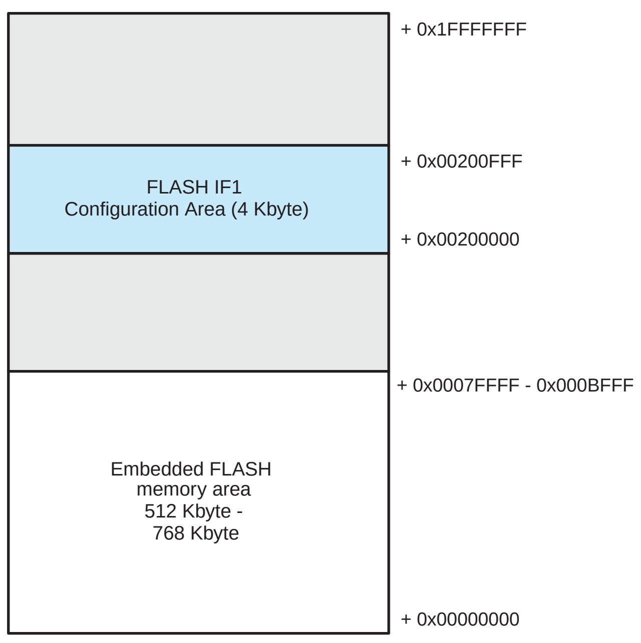


**Fig 8. Region 1 embedded flash memory**

Region 1 is reserved for the embedded flash. A data area of 2 Mbyte (to be prepared for a larger flash-memory instance) and a configuration area of 4 kB are reserved for each embedded flash instance. Although the LPC29xx contains only one embedded flash instance, the memory aperture per instance is defined at 4 Mbyte.

### 3.3 Region 2: external static memory area

Region 2 is reserved for the external static memory. The LPC29xx provides I/O pins for eight  bank-select signals and 24 address lines. This implies that eight memory banks of 16 Mbytes each can be addressed externally.

### 3.4 Region 3: external static memory controller area

The external Static-Memory Controller configuration area is located at region 3

### 3.5 Region 4: internal SRAM area

Figure 2–6 gives a graphical overview of the internal SRAM memory map.

Region 4 is reserved for internal SRAM. The LPC29xx has two internal SRAM instances. Instance #0 is 32 kB, instance #1 is 16 kB. See Section 7–1.

### 3.6 Regions 5 and 6

Regions 5 and 6 are not used.

### 3.7 Region 7: bus-peripherals area

Figure 2–6 gives a graphical overview of the bus-peripherals area memory map.

Region 7 is reserved for all stand-alone memory-mapped bus peripherals.

The lower part of region 7 is again divided into APB clusters, also referred to as subsystems in this User Manual. A APB cluster is typically used as the address space for a set of APB peripherals connected to a single AHB2APB bridge, the slave on the AHB system bus. The clusters are aligned on 256 kB boundaries. In the LPC29xx four APB clusters are in use: General SubSystem (GeSS), Peripheral SubSystem (PeSS), Networking SubSystem (IVNSS), and the Modulation and Sampling SubSystem (MSCSS). The APB peripherals are aligned on 4 kB boundaries inside the APB clusters.

The upper part of region 7 is used as the memory area where memory-mapped register interfaces of stand-alone AHB peripherals and a DTL cluster reside. Each of these is a slave on the AHB system bus. In the LPC29xx two such slaves are present: the Power, Clock and Reset subsystem (PCRSS) and the Vectored Interrupt Controller (VIC). The PCRSS is a DTL cluster in which the CGU, PMU and RGU are connected to the AHB system bus via an AHB2DTL adapter. The VIC is a DTL target connected to the AHB system bus via its own AHB2DTL adapter.

## 4. Memory-map operating concepts

The basic concept in the LPC29xx is that each memory area has a ‘natural’ location in the memory map. This is the address range for which code residing in that area is written. Each memory space remains permanently fixed in the same location, eliminating the need to have portions of the code designed to run in different address ranges.

Because of the location of the exception-handler vectors on the ARM9 processor (at addresses 0000 0000h through 0000 001Ch: see Table 2–9) By default, after reset, the embedded flash is mapped at address 0000 0000h to allow initial code to be executed and to perform the required initialization, which starts executing at 0000 0000h.

The LPC29xx generates the appropriate bus-cycle abort exception if an access is attempted for an address that is in a reserved or unused address region or unassigned peripheral spaces. For these areas both attempted data accesses and instruction fetches generate an exception. Note that write-access addresses should be word-aligned in ARM code or half-word aligned in Thumb code. Byte-aligned writes are performed as word or half-word aligned writes without error signalling.

Within the address space of an existing peripheral a data-abort exception is not generated in response to an access to an undefined address. Address decoding within each peripheral is limited to that needed to distinguish defined registers within the peripheral itself. Details of address aliasing within a peripheral space are not defined in the LPC29xx documentation and are not a supported feature.

Note that the ARM stores the pre-fetch abort flag along with the associated instruction (which will be meaningless) in the pipeline and processes the abort only if an attempt is made to execute the instruction fetched from the illegal address. This prevents the accidental aborts that could be caused by pre-fetches occurring when code is executed very near to a memory boundary.

Table 2–10 gives the base address overview of all peripherals:

**Table 9. Interrupt vectors address table**

| Address | Exception |
|---|---|
| 0x0000 0000 | Reset |
| 0x0000 0004 | Undefined instruction |
| 0x0000 0008 | Software interrupt |
| 0x0000 000C | Pre-fetch abort (instruction-fetch memory fault) |
| 0x0000 0010 | Data abort (data-access memory fault) |
| 0x0000 0014 | reserved |
| 0x0000 0018 | IRQ |
| 0x0000 001C | FIQ |

**Table 10. Peripherals base-address overview**

| Base address | Base name | AHB peripherals |
|---|---|---|
| **Memory region 0 to region 6** | | |
| 0x0000 0000 | | TCM memory |
| 0x2000 0000 | | Embedded flash memory |
| 0x2020 0000 | FMC RegBase | Embedded-flash controller configuration registers |
| 0x4000 0000 | | External static memory |
| 0x6000 0000 | SMC RegBase | External Static-Memory Controller configuration registers |
| 0x8000 0000 | | Internal SRAM memory |
| **APB Cluster 0: general subsystem** | | |
| 0xE000 0000 | CFID RegBase | Chip/feature ID register |
| 0xE000 1000 | SCU RegBase | System Control Unit |
| 0xE000 2000 | ER RegBase | Event Router |
| **APB Cluster 2: peripheral subsystem** | | |
| 0xE004 0000 | WDT RegBase | Watchdog Timer |
| 0xE004 1000 | TMR RegBase | Timer 0 |
| 0xE004 2000 | TMR RegBase | Timer 1 |
| 0xE004 3000 | TMR RegBase | Timer 2 |
| 0xE004 4000 | TMR RegBase | Timer 3 |
| 0xE004 5000 | UART RegBase | UART 0 |
| 0xE004 6000 | UART RegBase | UART 1 |
| 0xE004 7000 | SPI RegBase | SPI 0 |
| 0xE004 8000 | SPI RegBase | SPI 1 |
| 0xE004 9000 | SPI RegBase | SPI 2 |

**Table 10. Peripherals base-address overview** *…continued*

| Base address | Base name | AHB peripherals |
|---|---|---|
| 0xE004 A000 | GPIO RegBase | General-Purpose I/O 0 |
| 0xE004 B000 | GPIO RegBase | General-Purpose I/O 1 |
| 0xE004 C000 | GPIO RegBase | General-Purpose I/O 2 |
| 0xE004 D000 | GPIO RegBase | General-Purpose I/O 3 |
| 0xE004 E000 | GPIO RegBase | General-Purpose I/O 4 |
| 0xE004 F000 | GPIO RegBase | General-Purpose I/O 5 |
| **APB Cluster 4: networking subsystem** | | |
| 0xE008 0000 | CANC RegBase | CAN controller 0 |
| 0xE008 1000 | CANC RegBase | CAN controller 1 |
| 0xE008 2000 | I2C RegBase | I$^2$C0-bus interface |
| 0xE008 3000 | I2C Regbase | I$^2$C1-bus interface |
| 0xE008 6000 | CANAFM RegBase | CAN ID look-up table memory |
| 0xE008 7000 | CANAFR RegBase | CAN acceptance filter registers |
| 0xE008 8000 | CANCS RegBase | CAN central status registers |
| 0xE008 9000 | LIN RegBase | LIN master controller 0 |
| 0xE008 A000 | LIN RegBase | LIN master controller 1 |
| **APB Cluster 6: modulation and sampling-control subsystem** | | |
| 0xE00C 0000 | MTMR RegBase | MSCSS timer 0 |
| 0xE00C 1000 | MTMR RegBase | MSCSS timer 1 |
| 0xE00C 2000 | ADC RegBase | ADC 0 |
| 0xE00C 3000 | ADC RegBase | ADC 1 |
| 0xE00C 4000 | ADC RegBase | ADC 2 |
| 0xE00C 5000 | PWM RegBase | PWM 0 |
| 0xE00C 6000 | PWM RegBase | PWM 1 |
| 0xE00C 7000 | PWM RegBase | PWM 2 |
| 0xE00C 8000 | PWM RegBase | PWM 3 |
| 0xE00C 9000 | QEI RegBase | Quadrature encoder interface |
| **AHB peripherals: DMA controller, USB controller** | | |
| E010 0000 | USB RegBase | USB controller registers |
| E014 0000 | DMA RegBase | GPDMA controller registers |
| **Power, Clock and Reset control cluster** | | |
| FFFF 8000 | CGU RegBase | Clock Generation Unit |
| FFFF 9000 | RGU RegBase | Reset Generation Unit |
| FFFF A000 | PMU RegBase | Power Management Unit |
| **Vector interrupt controller** | | |
| FFFF F000 | VIC RegBase | Vectored Interrupt Controller |

# UM10316

## Chapter 3: LPC29xx Clock Generation Unit (CGU)

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

This chapter describes the base clock generation for all LPC29xx parts. CGU0 is identical for all parts. CGU1 is configuration dependent.

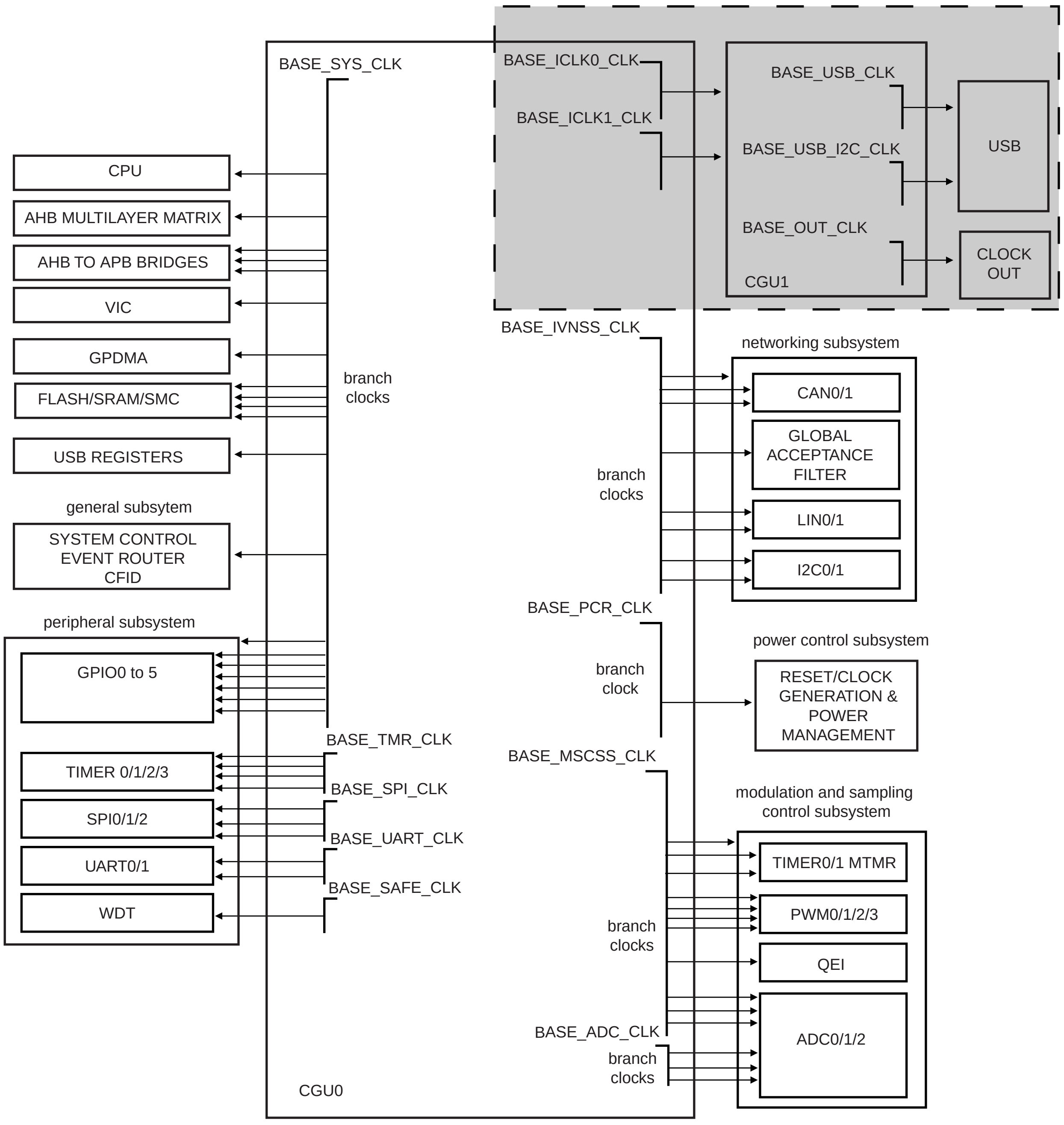
Table 11. LPC29xx base clock options

| Part | CGU0 base clocks | CGU1 base clocks |
|---|---|---|
| LPC2917/19/01 | see Table 3–12 | BASE_OUT_CLK |
| LPC2921/23/25 | see Table 3–12 | BASE_OUT_CLK, BASE_USB_CLK |
| LPC2926/27/29 | see Table 3–12 | BASE_OUT_CLK, BASE_USB_CLK, BASE_USB_I2C_CLK |
| LPC2939 | see Table 3–12 | BASE_OUT_CLK, BASE_USB_CLK, BASE_USB_I2C_CLK |
| LPC2930 | see Table 3–12 | BASE_OUT_CLK, BASE_USB_CLK, BASE_USB_I2C_CLK |

## 2. Introduction

The CGU0 is part of the Power Control, Clock, and Reset control (PCR) block and provides the clocks for all subsystems. A second, dedicated CGU1 provides the clocks for the USB block and a clock output. The CGU1 has two clock inputs to its PLL which are internally connected to two base clocks in the CGU0.

Both CGUs are functionally identical and have their own PLL and fractional divider registers.



The implementation of GPIO, ADC, and memory subsystem branch clocks varies for different LPC29xx parts.

**Fig 9. LPC29xx clock generation**

## 3. CGU0 functional description

The CGU0 uses a set of building blocks to generate the clock for the output branches. The building blocks are as follows:

- OSC1M (LP_OSC) – 1 MHz crystal oscillator
- XO50M – up to 25 MHz oscillator
- PL160M – PLL
- FDIV0..6 – 7 Frequency Dividers
- Output control

The following clock output branches are generated (Table 3–12):

**Table 12. CGU0 base clocks**

| Number | Name | Frequency (MHz) [1] | Description |
|---|---|---|---|
| 0 | BASE_SAFE_CLK | 0.4 | base safe clock (always on) for WDT |
| 1 | BASE_SYS_CLK | 125 | base system clock; ARM and AHB clock |
| 2 | BASE_PCR_CLK | 0.4 [2] | base PCR subsystem clock; for power control subsystem |
| 3 | BASE_IVNSS_CLK | 125 | base IVNSS subsystem clock for networking subsystem (CAN, LIN, and I2C) |
| 4 | BASE_MSCSS_CLK | 125 | base MSCSS subsystem clock for modulation and sampling control subsystem. |
| 5 | BASE_ICLK0_CLK | 125 | base internal clock 0, for CGU1 |
| 6 | BASE_UART_CLK | 125 | base UART clock |
| 7 | BASE_SPI_CLK | 50 | base SPI clock |
| 8 | BASE_TMR_CLK | 125 | base timers clock |
| 9 | BASE_ADC_CLK | 4.5 | base ADCs clock |
| 10 | test clock; reserved | - | this is an internal clock used for testing only. This clock is running at start-up and should be disabled in the PMU (see Table 5–51 for the test shell clock configuration registers). |
| 11 | BASE_ICLK1_CLK | 125 | base internal clock 1, for CGU1 |

[1] Maximum frequency that guarantees stable operation of the LPC29xx.

[2] Fixed to low-power oscillator.

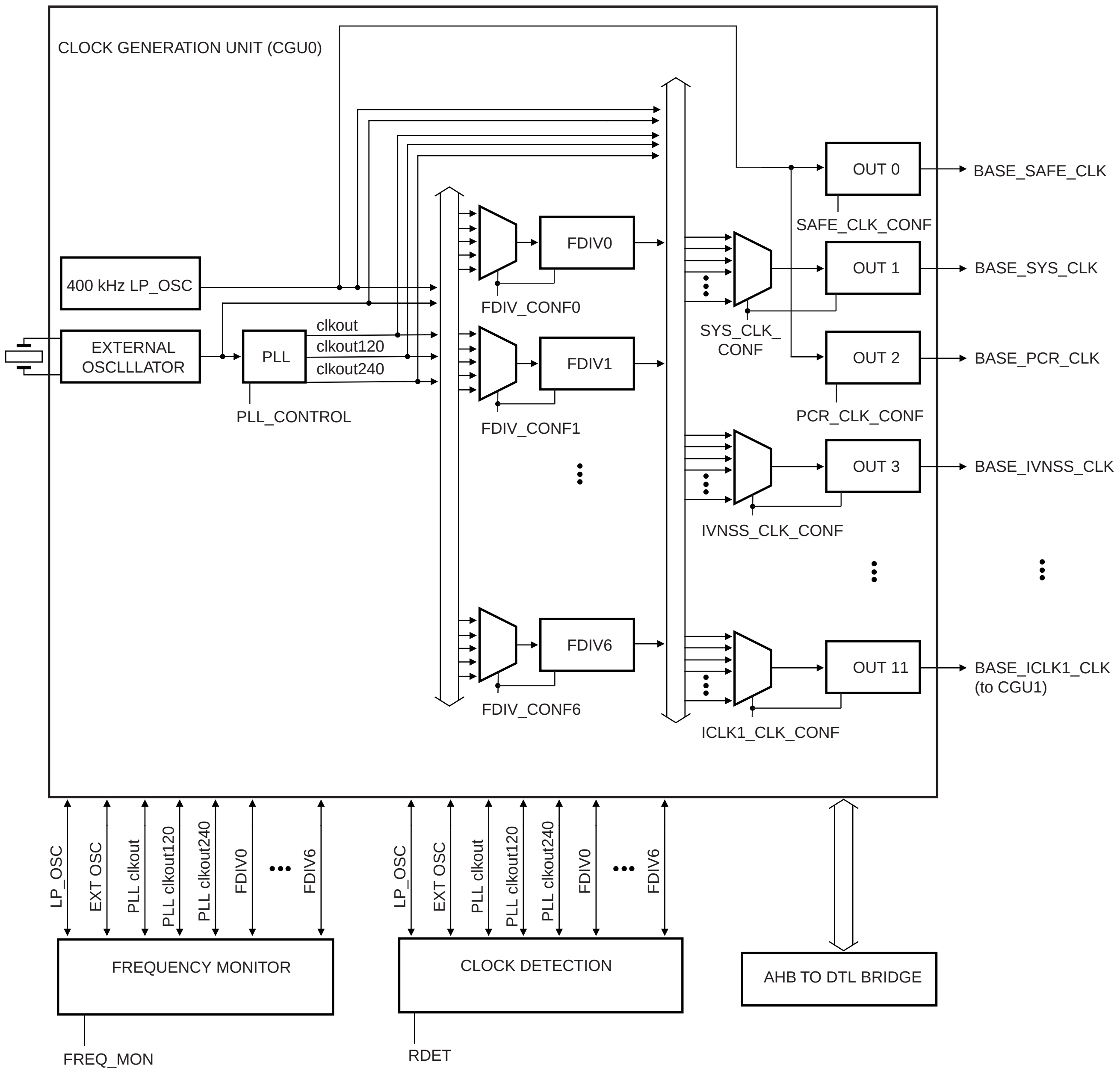


**Fig 10. Schematic representation of the CGU0**

The structure of the clock path of each clock output is shown in Figure 3–11.

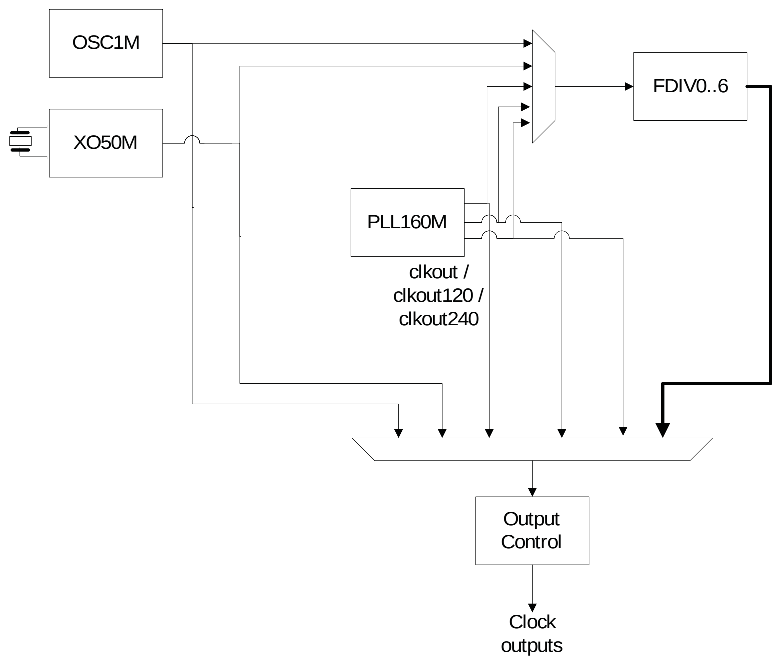


**Fig 11. Structure of the clock generation scheme**

## 3.1 Controlling the XO50M oscillator (external oscillator)

The XO50M oscillator can be disabled using the ENABLE field in the oscillator control register. Even when enabled, this can be bypassed using the BYPASS field in the same register. In this case the input of the OSC1M crystal is fed directly to the output.

The XO50M oscillator has an HF pin which selects the operating mode. For operation at higher frequencies (15-25 MHz), the XO50M oscillator HF must be enabled. For frequencies below that the pin must be disabled. Setting of the pin is controlled by the HF in the oscillator control register.

## 3.2 Controlling the PL160M PLL

The structure of the PLL clock path is shown in Figure 3–12.

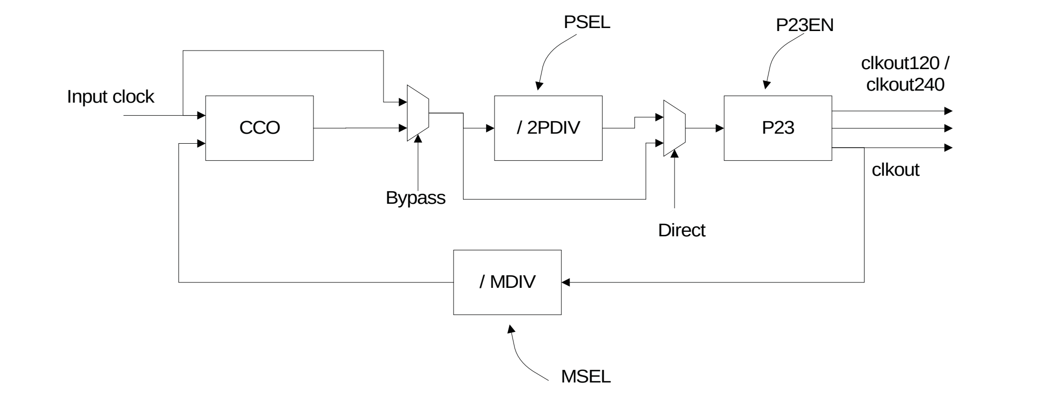


**Fig 12. PL160M PLL control mechanisms**

The PLL reference input clock is provided by the external oscillator (XO50M). The PLLs accept an input clock frequency in the range of 10 MHz to 25 MHz only. The input frequency can be directly routed to the post-divider using the BYPASS control. The post-divider can be bypassed using the DIRECT control.

The post-divider is controlled by settings of the field PSEL in the output control register. PSEL is a 2-bit value that selects a division between 1 and 8 in powers of 2.

The feedback divider is controlled by settings of the MSEL field in the output control register. The MSEL is a 5-bit value corresponding to the feedback divider minus 1. Thus, if MSEL is programmed to 0 the feedback divider is 1.

In normal mode the post-divider is enabled and the following relations are verified:

$F_{clkout} = MDIV \times F_{clkin} = F_{cco} / 2{\times}PDIV$

Values of the dividers are chosen with the following process:

1. Specify the input clock frequency $F_{clkin}$
2. Calculate M to obtain the desired output frequency $F_{clkout}$ with $M = F_{clkout} / F_{clkin}$
3. Find a value for P so that $F_{cco} = 2{\times}P / F_{clkout}$
4. Verify that all frequencies and divider values conform to the limits

In direct mode, the following relations are verified:

$F_{clkout} = M \times F_{clkin} = F_{cco}$

Unless the PLL is configured in bypass mode it must be locked before using it as a clock source. The PLL lock indication is read from the PLL status register.

Once the output clock is generated it is possible to use a three-phase output control which generates three clock signals separated in phase by 120°. This setting is controlled by field P23EN.

Settings to power down the PLL, controlled by field PD in the PLL control register, and safe switch setting controlled by the AUTOBLOK field are not shown in the illustration. Note that safe switching of the clock is not enabled at reset.

## 3.3 Controlling the frequency dividers

The seven frequency dividers are controlled by the FDIV0..6 registers.

The frequency divider divides the incoming clock by (L/D), where L and D are both 12-bit values, and attempts to produce a 50% duty-cycle. Each high or low phase is stretched to last approximately D/(L*2) input-clock cycles. When D/(L*2) is an integer the duty cycle is exactly 50%, otherwise it is an approximation.

The minimum division ratio is /2, so L should always be less than or equal to D/2. If not, or if L is equal to 0, the input clock is passed directly to the output without being divided.

## 3.4 Controlling the clock output

Once a source is selected for one of the clock branches the output clock can be further sub-divided using an output divider controlled by field IDIV in the clock-output configuration register.

Each clock-branch output can be individually controlled to power it down and perform safe switching between clock domains. These settings are controlled by the PD and AUTOBLOK fields respectively.

The clock output can trigger disabling of the clock branch on a specific polarity of the output. This is controlled via field RTX of the output-configuration register.

## 3.5 Reading the control settings

Each of the control registers is associated with a status register. These registers can be used to read the configured controls of each of the CGU building blocks.

## 3.6 Frequency monitor

The CGU includes a frequency-monitor mechanism which measures the clock pulses of one of the possible clock sources against the reference clock. The reference clock is the PCR block clock CLK_PCR.

When a frequency-monitor measurement begins two counters are started. The first starts from the specified number of reference-clock cycles (set in field RCNT) and counts down to 0: the second counts cycles of the monitored frequency starting from 0. The measurement is triggered by enabling it in field MEAS and stops either when the reference clock counter reaches 0 or the measured clock counter (in field FCNT) saturates.

The rate of the measured clock can be calculated using the formula:

Fmeas = Fcore $\times$ FCNTfinal / (RCNTinitial - RCNTfinal)

When the measurement is finished either FCNTfinal is equal to the saturated value of the counter (FCNT is a 14-bit value) or RCNTfinal is zero.

Measurement accuracy is influenced by the ratio between the clocks. For greater accuracy the frequency to measure should be closer to the reference clock.

## 3.7 Clock detection

All of the clock sources have a clock detector, the status of which can be read in a CGU register. This register indicates which sources have been detected.

If this is enabled, the absence of any clock source can trigger a hardware interrupt.

## 3.8 Bus disable

This safety feature is provided to avoid accidental changing of the clock settings. If it is enabled, access to all registers except the RBUS register (so that it can be disabled) is disabled and the clock settings cannot be modified.

## 3.9 Clock-path programming

The following flowchart shows the sequence for programing a complete clock path:

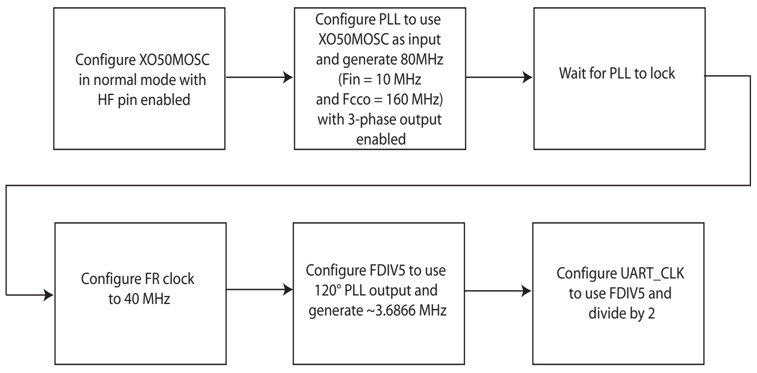

Configure XO50MOSC in normal mode with HF pin enabled
Configure PLL to use XO50MOSC as input and generate 80MHz (Fin = 10 MHz and Fcco = 160 MHz) with 3-phase output enabled
Wait for PLL to lock
Configure FR clock to 40 MHz
Configure FDIV5 to use 120° PLL output and generate ~3.6866 MHz
Configure UART_CLK to use FDIV5 and divide by 2

**Fig 13. Programming the clock path**

# 4. CGU1 functional description

The CGU1 block is functionally identical to the CGU0 block and generates two clocks for the USB interface and a dedicated output clock. The CGU1 block uses its own PLL and fractional dividers. The PLLs used in CGU0 and CGU1 are identical.

The clock input to the CGU1 PLL is provided by one of two base clocks generated in the CGU0: BASE_ICLK0_CLK or BASE_INT1CLK. The base clock not used for the PLL can be configured to drive the output clock directly.

The CGU1 provides the following three base clocks (Table 3–13):

**Table 13. CGU1 base clocks**

| Base clock | Parts of the device clocked by this branch clock |
|---|---|
| BASE_OUT_CLK | clock out pin |
| BASE_USB_CLK | USB clock |
| BASE_USB_I2C_CLK | USB OTG I2C clock |

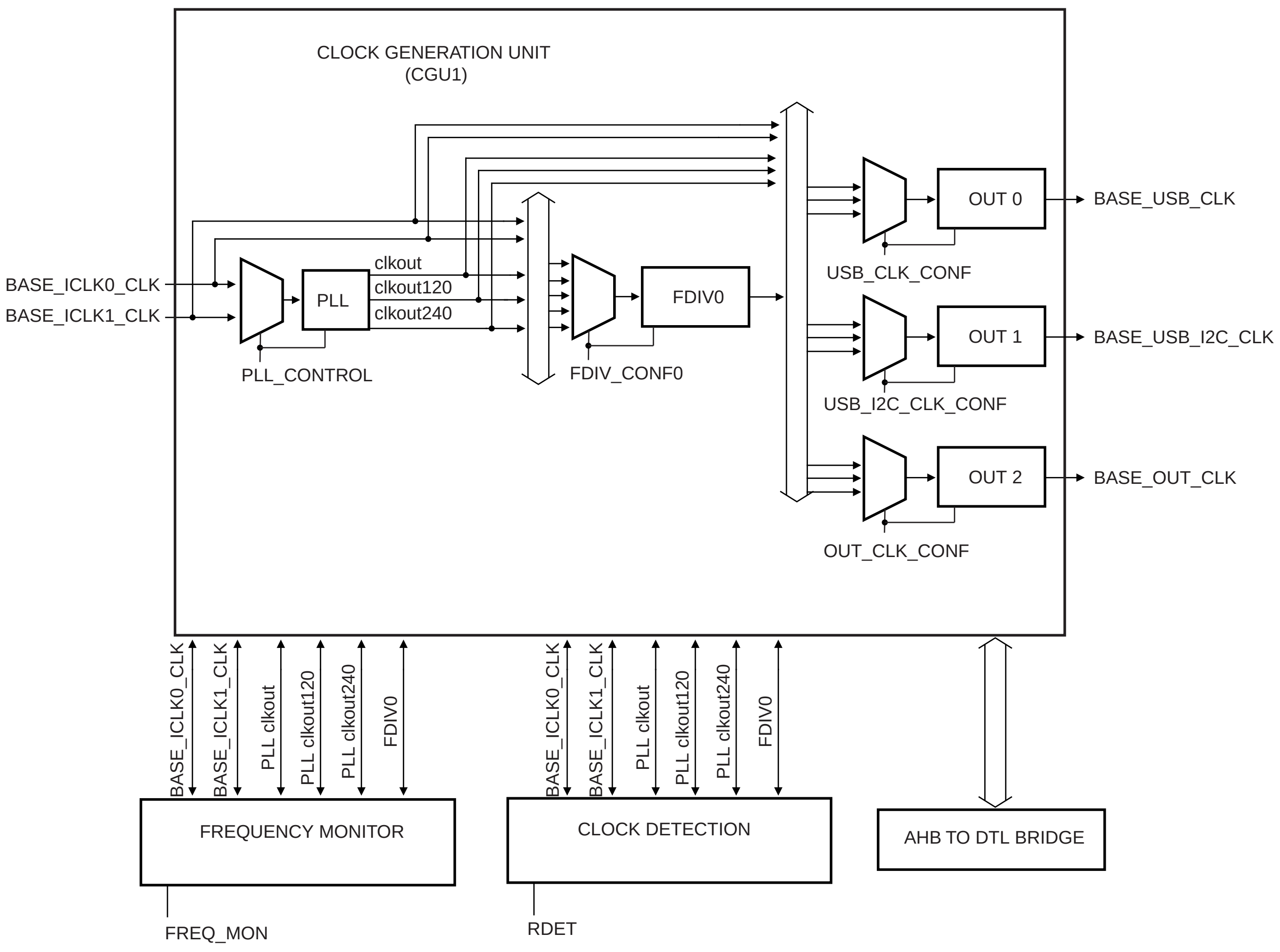


**Fig 14. Block diagram of the CGU1**

## 5. Register overview

**Remark:** Any clock-frequency adjustment has a direct impact on the timing of on-board peripherals such as the UARTs, SPI, Watchdog, timers, CAN controller, LIN master controller, ADCs, and flash memory interface.

**Table 14. Register overview: CGU0 (CGU0 base address: 0xFFFF 8000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| - | R | 0x000 | Reserved | 0x7100 0011 | - |
| - | R | 0x004 | Reserved | 0x0000 0000 | - |
| - | R | 0x008 | Reserved | 0x0C00 0000 | - |
| - | R | 0x00C | Reserved | - | - |
| FREQ_MON | R/W | 0x014 | Frequency monitor register | 0x0000 0000 | see Table 3–16 |
| RDET | R | 0x018 | Clock detection register | 0x0000 0FE3 | see Table 3–17 |

**Table 14. Register overview: CGU0 (CGU0 base address: 0xFFFF 8000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| XTAL_OSC_STATUS | R | 0x01C | Crystal-oscillator status register | 0x0000 0001 | see Table 3–18 |
| XTAL_OSC_CONTROL | R/W | 0x020 | Crystal-oscillator control register | 0x0000 0005 | see Table 3–19 |
| PLL_STATUS | R | 0x024 | PLL status register | 0x0005 1103 | see Table 3–20 |
| PLL_CONTROL | R/W | 0x028 | PLL control register | 0x0005 1103 | see Table 3–21 |
| FDIV_STATUS_0 | R | 0x02C | FDIV 0 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_0 | R/W | 0x030 | FDIV 0 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_1 | R | 0x034 | FDIV 1 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_1 | R/W | 0x038 | FDIV 1 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_2 | R | 0x03C | FDIV 2 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_2 | R/W | 0x040 | FDIV 2 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_3 | R | 0x044 | FDIV 3 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_3 | R/W | 0x048 | FDIV 3 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_4 | R | 0x04C | FDIV 4 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_4 | R/W | 0x050 | FDIV 4 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_5 | R | 0x054 | FDIV 5 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_5 | R/W | 0x058 | FDIV 5 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| FDIV_STATUS_6 | R | 0x05C | FDIV 6 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_6 | R/W | 0x060 | FDIV 6 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| SAFE_CLK_STATUS | R | 0x064 | Output-clock status register for BASE_SAFE_CLK | 0x0000 0000 | see Table 3–24 |
| SAFE_CLK_CONF | R/W | 0x068 | Output-clock configuration register for BASE_SAFE_CLK | 0x0000 0000 | see Table 3–25 |
| SYS_CLK_STATUS | R | 0x06C | Output-clock status register for BASE_SYS_CLK | 0x0000 0000 | see Table 3–26 |
| SYS_CLK_CONF | R/W | 0x070 | Output-clock configuration register for BASE_SYS_CLK | 0x0000 0000 | see Table 3–27 |
| PCR_CLK_STATUS | R | 0x074 | Output-clock status register for BASE_PCR_CLK | 0x0000 0000 | see Table 3–26 |
| PCR_CLK_CONF | R/W | 0x078 | Output-clock configuration register for BASE_PCR_CLK | 0x0000 0000 | see Table 3–27 |
| IVNSS_CLK_STATUS | R | 0x07C | Output-clock status register for BASE_IVNSS_CLK | 0x0000 0000 | see Table 3–26 |
| IVNSS_CLK_CONF | R/W | 0x080 | Output-clock configuration register for BASE_IVNSS_CLK | 0x0000 0000 | see Table 3–27 |
| MSCSS_CLK_STATUS | R | 0x084 | Output-clock status register for BASE_MSCSS_CLK | 0x0000 0000 | see Table 3–26 |
| MSCSS_CLK_CONF | R/W | 0x088 | Output-clock configuration register for BASE_MSCSS_CLK | 0x0000 0000 | see Table 3–27 |
| ICLK0_CLK_STATUS | R | 0x08C | Output-clock status register for BASE_ICLK0_CLK | 0x0000 0000 | see Table 3–26 |
| ICLK0_CLK_CONF | R/W | 0x090 | Output-clock configuration register for BASE_ICLK0_CLK | 0x0000 0000 | see Table 3–27 |
| UART_CLK_STATUS | R | 0x094 | Output-clock status register for BASE_UART_CLK | 0x0000 0000 | see Table 3–26 |

**Table 14. Register overview: CGU0 (CGU0 base address: 0xFFFF 8000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| UART_CLK_CONF | R/W | 0x098 | Output-clock configuration register for BASE_UART_CLK | 0x0000 0000 | see Table 3–27 |
| SPI_CLK_STATUS | R | 0x09C | Output-clock status register for BASE_SPI_CLK | 0x0000 0000 | see Table 3–26 |
| SPI_CLK_CONF | R/W | 0x0A0 | Output-clock configuration register for BASE_SPI_CLK | 0x0000 0000 | see Table 3–27 |
| TMR_CLK_STATUS | R | 0x0A4 | Output-clock status register for BASE_TMR_CLK | 0x0000 0000 | see Table 3–26 |
| TMR_CLK_CONF | R/W | 0x0A8 | Output-clock configuration register for BASE_TMR_CLK | 0x0000 0000 | see Table 3–27 |
| ADC_CLK_STATUS | R | 0x0AC | Output-clock status register for BASE_ADC_CLK | 0x0000 0000 | see Table 3–26 |
| ADC_CLK_CONF | R/W | 0x0B0 | Output-clock configuration register for BASE_ADC_CLK | 0x0000 0000 | see Table 3–27 |
| - | R | 0x0B4 | reserved | 0x0000 0000 | - |
| - | R/W | 0x0B8 | reserved | 0x0000 0000 | - |
| ICLK1_CLK_STATUS | R | 0x0BC | Output-clock status register for BASE_ICLK1_CLK | 0x0000 0000 | see Table 3–26 |
| ICLK1_CLK_CONF | R/W | 0x0C0 | Output-clock configuration register for BASE_ICLK1_CLK | 0x0000 0000 | see Table 3–27 |
| INT_CLR_ENABLE | W | 0xFD8 | Interrupt clear-enable register | 0x0000 0000 | see Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Interrupt set-enable register | 0x0000 0000 | see Table 10–92 |
| INT_STATUS | R | 0xFE0 | Interrupt status register | 0x0000 0FE3 | see Table 10–93 |
| INT_ENABLE | R | 0xFE4 | interrupt enable register | 0x0000 0000 | see Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | Interrupt clear-status register | 0x0000 0000 | see Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | Interrupt set-status register | 0x0000 0000 | see Table 10–96 |
| - | R | 0xFF0 | Reserved | - | - |
| BUS_DISABLE | R/W | 0xFF4 | Bus disable register | 0x0000 0000 | see Table 3–30 |
| - | R | 0xFF8 | Reserved | - | - |
| - | R | 0xFFC | Reserved | 0xA0A8 1000 | - |

**Table 15. Register overview: CGU1 (CGU1 base address: 0xFFFF B000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| reserved | R | 0x000 | Reserved | 0x7100 0011 | - |
| reserved | R | 0x004 | Reserved | 0x0000 0000 | - |
| reserved | R | 0x008 | Reserved | 0x0C00 0000 | - |
| reserved | R | 0x00C | Reserved | - | - |
| FREQ_MON | R/W | 0x014 | Frequency monitor register | 0x0000 0000 | see Table 3–16 |

**Table 15. Register overview: CGU1 (CGU1 base address: 0xFFFF B000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| RDET | R | 0x018 | Clock detection register | 0x0000 0FE3 | see Table 3–17 |
| PLL_STATUS | R | 0x01C | PLL status register | 0x0005 1103 | see Table 3–20 |
| PLL_CONTROL | R/W | 0x020 | PLL control register | 0x0005 1103 | see Table 3–21 |
| FDIV_STATUS_0 | R | 0x024 | FDIV 0 frequency-divider status register | 0x0000 1001 | see Table 3–22 |
| FDIV_CONF_0 | R/W | 0x028 | FDIV 0 frequency-divider control register | 0x0000 1001 | see Table 3–23 |
| USB_CLK_STATUS | R | 0x02C | Output-clock status register for BASE_USB_CLK | 0x0000 0000 | see Table 3–26 |
| USB_CLK_CONF | R/W | 0x030 | Output-clock configuration register for BASE_USB_CLK | 0x0000 0000 | see Table 3–27 |
| USB_I2C_CLK_STATUS | R | 0x034 | Output-clock status register for BASE_I2C_USB_CLK | 0x0000 0000 | see Table 3–26 |
| USB_I2C_CLK_CONF | R/W | 0x38 | Output-clock configuration register for BASE_I2C_USB_CLK | 0x0000 0000 | see Table 3–27 |
| OUT_CLK_STATUS | R | 0x03C | Output-clock status register for BASE_OUT_CLK | 0x0000 0000 | see Table 3–26 |
| OUT_CLK_CONF | R/W | 0x040 | Output-clock configuration register for BASE_OUT_CLK | 0x0000 0000 | see Table 3–27 |
| BUS_DISABLE | R/W | 0xFF4 | Bus disable register | 0x0000 0000 | see Table 3–30 |

## 5.1 Frequency monitor register

The CGU can report the relative frequency of any operating clock. The clock to be measured must be selected by software, while the fixed-frequency *BASE_PCR_CLK* is used as the reference frequency. A 14-bit counter then counts the number of cycles of the measured clock that occur during a user-defined number of reference-clock cycles. When the MEAS bit is set the measured-clock counter is reset to 0 and counts up, while the 9-bit reference-clock counter is loaded with the value in RCNT and then counts down towards 0. When either counter reaches its terminal value both counters are disabled and the MEAS bit is reset to 0. The current values of the counters can then be read out and the selected frequency obtained by the following equation:

$$fselected = \frac{Qselected}{(Qref[initial] - Qref[final])} \times fref$$

If RCNT is programmed to a value equal to the core clock frequency in kHz and reaches 0 before the FCNT counter saturates, the value stored in FCNT would then show the measured clock’s frequency in kHz without the need for any further calculation.

Note that the accuracy of this measurement can be affected by several factors. Quantization error is noticeable if the ratio between the two clocks is large (e.g. 100 kHz vs. 1kHz), because one counter saturates while the other still has only a small count value. Secondly, due to synchronization, the counters are not started and stopped at exactly the same time. Finally, the measured frequency can only be to the same level of precision as the reference frequency.

**Remark:** The clock selection in this register depends on whether the register is used for CGU0 or CGU1. In the CGU0, the low-power oscillator (LP_OSC) or the external crystal oscillator can be selected as input. In the CGU1, the two CGU0 base clocks BASE_ICLK0_CLK and BASE_ICLK1_CLK, can be selected instead. CGU1 has only one fractional divider register.

Table 16. FREQ_MON register bit description (FREQ_MON, address 0xFFFF 8014 (CGU0) and 0xFFFF B014 (CGU1))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | Clock-source selection for the clock to be measured. |
| | | | 0x00* | LP_OSC (CGU0) or BASE_ICLK0_CLK (CGU1) |
| | | | 0x01 | Crystal oscillator (CGU0) or BASE_ICLK1_CLK (CGU1) |
| | | | 0x02 | PLL |
| | | | 0x03 | PLL +120° |
| | | | 0x04 | PLL +240° |
| | | | 0x05 | FDIV0 (CGU0 and CGU1) |
| | | | 0x06 | FDIV1 (CGU0 only) |
| | | | 0x07 | FDIV2 (CGU0 only) |
| | | | 0x08 | FDIV3 (CGU0 only) |
| | | | 0x09 | FDIV4 (CGU0 only) |
| | | | 0x0A | FDIV5 (CGU0 only) |
| | | | 0x0B | FDIV6 (CGU0 only) |
| 23 | MEAS | R/W | | Measure frequency |
| | | | 0* | |
| 22 to 9 | FCNT | R | | Selected clock-counter value |
| | | | 0x0* | |
| 8 to 0 | RCNT | R/W | | Reference clock-counter value |
| | | | 0x0* | |

## 5.2 Clock detection register

Each clock generator has a clock detector associated with it to alert the system if a clock is removed or connected. The status register RDET can determine the current ‘clock-present’ status.

If enabled, interrupts are generated whenever ‘clock present’ changes status, so that an interrupt is generated if a clock changes from ‘present’ to ‘non-present’ or from ‘non-present’ to ‘present’.

**Remark:** The clock selection in this register depends on whether the register is used for CGU0 or CGU1. In the CGU0, the low-power oscillator (LP_OSC) or the external crystal oscillator can be selected as input. In the CGU1, the two CGU0 base clocks BASE_ICLK0_CLK and BASE_ICLK1_CLK, can be selected instead. In the CGU1, only one fractional divider register is used.

Table 17. RDET register bit description (RDET, address 0xFFFF 8018 (CGU0) or 0xFFFF B018 (CGU1))

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved |
| 11 | FDIV6_PRESENT | R | | Activity-detection register for FDIV 6 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 10 | FDIV5_PRESENT | R | | Activity-detection register for FDIV 5 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 9 | FDIV4_PRESENT | R | | Activity-detection register for FDIV 4 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 8 | FDIV3_PRESENT | R | | Activity-detection register for FDIV 3 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 7 | FDIV2_PRESENT | R | | Activity-detection register for FDIV 2 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 6 | FDIV1_PRESENT | R | | Activity-detection register for FDIV 1 (CGU0 only) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 5 | FDIV0_PRESENT | R | | Activity-detection register for FDIV 0 (CGU0 and CGU1) |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 4 | PLL240_PRESENT | R | | Activity-detection register for 240°-shifted PLL output |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 3 | PLL120_PRESENT | R | | Activity-detection register for 120°-shifted PLL output |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 2 | PLL_PRESENT | R | | Activity-detection register for normal PLL output |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |

**Table 17. RDET register bit description (RDET, address 0xFFFF 8018 (CGU0) or 0xFFFF B018 (CGU1))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 | XTAL_PRESENT (CGU0) or BASE_ICLK0_CLK_PRESENT (CGU1) | R | | Activity-detection register for crystal -oscillator output |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |
| 0 | LP_OSC_PRESENT (CGU0) or BASE_ICLK1_CLK_PRESENT (CGU1) | R | | Activity-detection register for LP_OSC |
| | | | 1* | Clock present |
| | | | 0 | Clock not present |

## 5.3 Crystal-oscillator status register (CGU0)

The register XTAL_OSC_STATUS reflects the status bits for the crystal oscillator.

**Table 18. XTAL_OSC_STATUS register bit description (XTAL_OSC_STATUS, address 0xFFFF 801C)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved |
| 2 | HF | R | | Oscillator HF pin |
| | | | 1* | Oscillator high-frequency mode (crystal or external clock source above 10 MHz) |
| | | | 0 | Oscillator low-frequency mode (crystal or external clock source below 20 MHz) |
| 1 | BYPASS | R | | Configure crystal operation or external clock input pin XIN_OSC |
| | | | 0 | Operation with crystal connected |
| | | | 1* | Bypass mode. Use this mode when an external clock source is used instead of a crystal |
| 0 | ENABLE | R | | Oscillator-pad enable |
| | | | 0 | Power-down |
| | | | 1* | Enable |

## 5.4 Crystal oscillator control register (CGU0)

The register XTAL_OSC_CONTROL contains the control bits for the crystal oscillator. Following a change of ENABLE bit in XTAL_OSC_CONTROL register requires a read in XTAL_OSC_STATUS to confirm ENABLE bit is indeed changed.

**Table 19. XTAL_OSC_CONTROL register bit description (XTAL_OSC_CONTROL, address 0xFFFF 8020)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved |

**Table 19. XTAL_OSC_CONTROL register bit description** *…continued*(XTAL_OSC_CONTROL, address 0xFFFF 8020)
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 2 | HF | R/W | | Oscillator HF pin |
| | | | 1* | Oscillator high-frequency mode (crystal or external clock source 15 to 25 MHz)[2] |
| | | | 0 | Oscillator low-frequency mode (crystal or external clock source 1 to 20 MHz)[2] |
| 1 | BYPASS | R/W | | Configure crystal operation or external-clock input pin XIN_OSC[1] |
| | | | 0* | Operation with crystal connected |
| | | | 1 | Bypass mode. Use this mode when an external clock source is used instead of a crystal |
| 0 | ENABLE | R/W | | Oscillator-pad enable[1] |
| | | | 0 | Power-down |
| | | | 1* | Enable |

[1] Do not change the BYPASS and ENABLE bits in one write-action: this will result in unstable device operation!

[2] For between 15 MHz to 20 MHz the state of the HF pin is don't care, see also the crystal specification notes in Ref. 31–1. Section 11 (Oscillator).

## 5.5 PLL status register (CGU0 and CGU1)

The register PLL_STATUS reflects the status bits of the PLL.

**Table 20. PLL_STATUS register bit description (PLL_STATUS, address 0xFFFF 8024 (CGU0) and 0xFFFF B024 (CGU1))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 0 | LOCK | R | | Indicates if the PLL is in lock or not. |
| | | | 1 | In lock |
| | | | 0* | Not in lock |

## 5.6 PLL control register (CGU0 and CGU1)

The PLL_CONTROL register contains the control bits for the PLL. In the CGU0, only the crystal oscillator is allowed as an input to the PLL. In the CGU1, both internal base clocks, BASE_ICLK0_CLK and BASE_ICLK1_CLK, can be inputs to the PLL.

### Post-divider ratio programming

The division ratio of the post-divider is controlled by PSEL[0:1] in the PLL_CONTROL register. The division ratio is twice the value of P. This guarantees an output clock with a 50% duty cycle.

### Feedback-divider ratio programming

The feedback-divider division ratio is controlled by MSEL[4:0] in the PLL_CONTROL register. The division ratio between the PLL output clock and the input clock is the decimal value on MSEL[4:0] plus one.

### Frequency selection, mode 1 (normal mode)

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

The output frequency of the PLL is given by the following equation:

$$fclkoutPLL = Mfclkin = \frac{fcco}{(2 \cdot P)}$$

To select the appropriate values for M and P:

1. Specify the input clock frequency $f_{clkin}$
2. Calculate M to obtain the desired output frequency $f_{clkout\ PLL}$ with $M = f_{clkout}/f_{clkin}$
3. Find a value for P so that $f_{cco} = 2 \times P \times f_{clkout}$
4. Verify that all frequencies and divider values conform to the limits specified.

### Frequency selection, mode 2 (direct CCO mode)

In this mode the post-divider is bypassed and the CCO clock is sent directly to the output(s), leading to the following frequency equation:

$$fclkout = Mfclkin = fcco$$

To select the appropriate values for M and P:

1. Specify the input clock frequency $f_{clkin}$
2. Calculate M to obtain the desired output frequency $f_{clkout}$ with $M = f_{clkout}/f_{clkin}$
3. Verify that all frequencies and divider values conform to the limits specified.

Note that although the post-divider is not used, it still runs in this mode. To reduce current consumption to the lowest possible value it is recommended to set PSEL[1:0] to ’00’. This sets the post-divider to divide by two, which causes it to consume the least amount of current.

**Table 21. PLL_CONTROL register bit description (PLL_CONTROL, address 0xFFFF 8028 (CGU0) and 0xFFFF B028 (CGU1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | Clock-source Selection for clock generator to be connected to the input of the PLL. |
| | | | 0x00* | Not used (CGU0) or BASE_ICLK0_CLK (CGU1) |
| | | | 0x01 | Crystal oscillator (CGU0) or BASE_ICLK1_CLK (CGU1) |
| | | | 0x02 to 0xFF | Not used |

**Table 21. PLL_CONTROL register bit description (PLL_CONTROL, address 0xFFFF 8028 (CGU0) and 0xFFFF B028 (CGU1))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 23 to 16 | MSEL[4:0] | R/W | | Feedback-divider division ratio (M)[1] |
| | | | 00000 | 1 |
| | | | 00001 | 2 |
| | | | 00010 | 3 |
| | | | 00011 | 4 |
| | | | 00100* | 5 |
| | | | : | : |
| | | | 11111 | 32 |
| 15 to 12 | reserved | R | | Reserved |
| 11 | AUTOBLOK | W | 1 | Enables auto-blocking of clock when programming changes |
| | | | 0 | No action |
| 10 | reserved | R | - | Reserved |
| 9 and 8 | PSEL[1:0] | R/W | | Post-divider division ratio (2P)[1] |
| | | | 00 | 2 |
| | | | 01* | 4 |
| | | | 10 | 8 |
| | | | 11 | 16 |
| 7 | DIRECT | R/W | | Direct CCO clock output control |
| | | | 0* | Clock output goes through post-divider |
| | | | 1 | Clock signal goes directly to outputs |
| 6 to 3 | reserved | R | | Reserved |
| 2 | P23EN | R/W | | Three-phase output mode control |
| | | | 0* | PLL +120° and PLL +240° outputs disabled |
| | | | 1 | PLL +120° and PLL +240° outputs enabled |
| 1 | BYPASS | R/W | | Input-clock bypass control |
| | | | 0 | CCO clock sent to post-dividers (only for test modes) |
| | | | 1* | PLL input clock sent to post-dividers |
| 0 | PD | R/W | | Power-down control |
| | | | 0 | Normal mode |
| | | | 1* | Power-down mode[2] |

[1] Changing the divider ratio while the PLL is running is not recommended. Since there is no way of synchronizing the change of the MSEL and PSEL values with the divider 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.

[2] To power down the PLL, P23EN bit should also be set to 0.

## 5.7 Frequency divider status register

There is one status register FDIV_STATUS_n for each frequency divider (n = 0..6 for CGU0). Note that there is only one frequency divider in the CGU1. The status bits reflect the inputs to the FDIV as driven from the control register

**Table 22. FDIV_STATUS_n register bit description (FDIV_STATUS_0 to 6, address 0xFFFF 802C/34/3C/44/4C/54/5C (CGU0) and FDIV_STATUS_0, address 0xFFFF B024 (CGU1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R | | Selected source clock for FDIV n |
| | | | 0x00* | LP_OSC (CGU0) or (BASE_ICLK0_CLK) (CGU1) |
| | | | 0x01 | Crystal oscillator (CGU0) or (BASE_ICLK1_CLK) (CGU1) |
| | | | 0x02 | PLL |
| | | | 0x03 | PLL +120$^0$ |
| | | | 0x04 | PLL +240$^0$ |
| | | | 0x05 to 0xFF | Not used |
| 23 to 12 | LOAD | R | | Load value |
| | | | 0x1* | |
| 11 to 0 | DENOMINATOR | R | | Denominator or modulo value. |
| | | | 0x1* | |

## 5.8 Frequency divider configuration register

There is one control register FDIV_CONF_n for each frequency divider (n = 0..6).

The frequency divider divides the incoming clock by (LOAD/DENOMINATOR), where LOAD and DENOMINATOR are both 12-bit values programmed in the control register FDIV_CONTROL_n.

Essentially the output clock generates ‘LOAD’ positive edges during every ‘DENOMINATOR’ cycle of the input clock. An attempt is made to produce a 50% duty-cycle. Each high or low phase is stretched to last approximately DENOMINATOR/(LOAD*2) input clock cycles. When DENOMINATOR/(LOAD*2) is an integer the duty cycle is exactly 50%: otherwise the waveform will only be an approximation. It will be close to 50% for relatively large non-integer values of DENOMINATOR/(LOAD*2), but not for small values.

The minimum division ratio is divide-by-2, so LOAD should always be less than or equal to (DENOMINATOR/2). If this is not true, or if LOAD is equal to 0, the input clock is passed directly to the output with no division.

Table 23. FDIV_CONF_n register bit description (FDIV_CONF_n, address 0xFFFF 8030/38/40/48/50/58/60 (CGU0) and FDIV_CONF_0, address 0xFFFF B028 (CGU1))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | Selected source clock for FDIV n |
| | | | 0x00* | LP_OSC (CGU0) or (BASE_ICLK0_CLK) (CGU1) |
| | | | 0x01 | Crystal oscillator (CGU0) or (BASE_ICLK1_CLK) (CGU1) |
| | | | 0x02 | PLL |
| | | | 0x03 | PLL +120$^0$ |
| | | | 0x04 | PLL +240$^0$ |
| | | | 0x05 to 0xFF | Invalid |
| 23 to 12 | LOAD | R/W | | Load value |
| | | | 0x1* | |
| 11 to 0 | DENOMINATOR | R/W | | Denominator or modulo value. |
| | | | 0x1* | |

## 5.9 Output-clock status register for BASE_SAFE_CLK and BASE_PCR_CLK

There is one status register for each CGU output clock generated. All output generators have the same register bits. Exceptions are the output generators for BASE_SAFE_CLK and BASE_PCR_CLK, which are described here. For the other outputs, see Section 3–5.11.

Table 24. SAFE_CLK_STATUS (address 0xFFFF 8064), PCR_CLK_STATUS (address 0xFFFF 0074) register bit description
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved |
| 4 to 2 | IDIV | R | 000* | Integer divide value |
| 1 to 0 | reserved | R | - | Reserved. |

## 5.10 Output-clock configuration register for BASE_SAFE_CLK and BASE_PCR_CLK

There is one configuration register for each CGU output clock generated. All output generators have the same register bits. An exception is the output generators for BASE_SAFE_CLK and BASE_PCR_CLK, which are described here. For the other outputs see Section 3–5.12.

Table 25. SAFE_CLK_CONF (address 0xFFFF 8068), PCR_CLK_CONF (address 0xFFFF 8078) register bit description

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | Selected source clock |
| | | | 0x0* | LP_OSC |
| | | | 0x01 to 0xFF | Invalid: the hardware will not accept these values when written |
| 23 to 5 | reserved | R | - | Reserved; do not modify, read as logic 0, write as logic 0 |
| 4 to 2 | IDIV | R/W | 000* | Integer divide value |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |

## 5.11 Output-clock status register for CGU0 clocks

There is one status register for each CGU output clock generated. All output generators have the same register bits. Exceptions are the output generators for BASE_SAFE_CLK and BASE_PCR_CLK, see Section 3–5.9.

Table 26. XX_CLK_STATUS register bit description (XX = SYS (address 0xFFFF 806C), IVNSS (address 0xFFFF 807C), MSCSS (address 0xFFFF 8084), UART (address 0xFFFF 8094), SPI (address 0xFFFF 809C), TMR (address 0xFFFF 80A4), ADC (address 0xFFFF 80AC))

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved |
| 4 to 2 | IDIV | R | 000* | Integer divide value |
| 1 | RTX | R | 0* | Clock-disable polarity |
| 0 | PD | R | 0* | Power-down clock slice |

## 5.12 Output-clock configuration register for CGU0 clocks

There is one configuration register for each CGU output clock generated. All output generators have the same register bits. Exceptions are the output generators for BASE_SAFE_CLK and BASE_PCR_CLK, see Section 3–5.10.

XX = SYS, IVNSS, MSCSS, UART, SPI, TMR or ADC, ICLK0/1_CLK

Each output generator takes in one input clock and sends one clock out of the CGU. In between the clock passes through an integer divider and a clock control block. A clock blocker/switch block connects to the clock control block.

The integer divider has a 3-bit control signal, IDIV, and divides the incoming clock by any value from 1 through 8. The divider value is equal to (IDIV + 1); if IDIV is equal to zero, the incoming clock is passed on directly to the next stage. When the input to the integer divider has a 50% duty cycle the divided output will have a 50% duty cycle for all divide values. If the incoming duty cycle is not 50% only even divide values will produce an output clock with a 50% duty cycle.

**Table 27. XX_CLK_CONF register bit description (XX = SYS (address 0xFFFF 8070), IVNSS (address 0xFFFF 8080), MSCSS (address 0xFFFF 8088), UART (address 0xFFFF 8098), SPI (address 0xFFFF 80A0), TMR (address 0xFFFF 80A8), ADC (address 0xFFFF 80B0))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | selected source clock |
| | | | 0x00* | LP_OSC |
| | | | 0x01 | Crystal oscillator[1] |
| | | | 0x02 | PLL |
| | | | 0x03 | PLL +120$^0$ |
| | | | 0x04 | PLL +240$^0$ |
| | | | 0x05 | FDIV0 |
| | | | 0x06 | FDIV1 |
| | | | 0x07 | FDIV2 |
| | | | 0x08 | FDIV3 |
| | | | 0x09 | FDIV4 |
| | | | 0x0A | FDIV5 |
| | | | 0x0B | FDIV6 |
| 23 to 12 | reserved | R | - | Reserved |
| 11 | AUTOBLOK | W | - | Enables auto-blocking of clock when programming changes |
| 10 to 5 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 4 to 2 | IDIV | R/W | 000* | Integer divide value |
| 1 | reserved | R/W | 0* | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 0 | PD | R/W | 0* | Power-down clock slice |

[1] At reset release, the JTAGSEL pin is sampled. If it is LOW (ARM debug), the crystal oscillator (XO50M) will be selected as source for BASE_SYS_CLK.

## 5.13 Output-clock status register for CGU1 clocks

There is one status register for each CGU1 output clock generated. All output generators have the same register bits.

**Table 28. XX_CLK_STATUS register bit description (XX = USB_CLK (address 0xFFFF B02C), USB_I2C (address 0xFFFF B034), OUT_CLK (address 0xFFFF B03C))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved |
| 4 to 2 | IDIV | R | 000* | Integer divide value |
| 1 | RTX | R | 0* | Clock-disable polarity |
| 0 | PD | R | 0* | Power-down clock slice |

## 5.14 Output-clock configuration register for CGU1 clocks

There is one configuration register for each CGU1 output clock generated. All output generators have the same register bits. The CGU1 output clock can be generated directly from the two CGU0 base clocks BASE_ICLK0_CLK and BASE_ICLK1_CLK or from the CGU1 PLL.

Each output generator takes in one input clock and sends one clock out of the CGU. In between the clock passes through an integer divider and a clock control block. A clock blocker/switch block connects to the clock control block.

The integer divider has a 3-bit control signal, IDIV, and divides the incoming clock by any value from 1 through 8. The divider value is equal to (IDIV + 1); if IDIV is equal to zero, the incoming clock is passed on directly to the next stage. When the input to the integer divider has a 50% duty cycle the divided output will have a 50% duty cycle for all divide values. If the incoming duty cycle is not 50% only even divide values will produce an output clock with a 50% duty cycle.

**Table 29. XX_CLK_CONF register bit description (XX = USB_CLK (address 0xFFFF B030), USB_I2C_CLK (address 0xFFFF B038), OUT_CLK (address 0xFFFF B040))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | CLK_SEL | R/W | | selected source clock |
| | | | 0x00* | BASE_ICLK0_CLK |
| | | | 0x01 | BASE_ICLK1_CLK |
| | | | 0x02 | PLL |
| | | | 0x03 | PLL +120$^0$ |
| | | | 0x04 | PLL +240$^0$ |
| | | | 0x05 | FDIV0 |
| | | | 0x06 - 0x0B | reserved |
| 23 to 12 | reserved | R | - | Reserved |
| 11 | AUTOBLOK | W | - | Enables auto-blocking of clock when programming changes |
| 10 to 5 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 4 to 2 | IDIV | R/W | 000* | Integer divide value |
| 1 | reserved | R/W | 0* | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 0 | PD | R/W | 0* | Power-down clock slice |

[1] When JTAG = 1, crystal Oscillator will be the default value for the BASE_SYS_CLK

## 5.15 Bus disable register

The BUS_DISABLE register prevents any disabled register in the CGU0 from being written to.

**Table 30. BUS_DISABLE register bit description (BUS_DISABLE, address 0xFFFF 8FF4 (CGU0) and 0xFFFF BFF4 (CGU1))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 0 | RRBUS | R/W | | Bus write-disable bit |
| | | | 1 | No writes to registers within CGU are possible (except the BUS_DISABLE register) |
| | | | 0* | Normal operation |

## 5.16 CGU0 interrupt bit description

Table 3–31 gives the interrupts for the CGU0. The first column gives the bit number in the interrupt registers. For a general explanation of the interrupt concept and a description of the registers see Section 10–5.

**Table 31. CGU interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 12 | unused | Unused |
| 11 | FDIV6 | FDIV 6 activity state change |
| 10 | FDIV5 | FDIV 5 activity state change |
| 9 | FDIV4 | FDIV 4 activity state change |
| 8 | FDIV3 | FDIV 3 activity state change |
| 7 | FDIV2 | FDIV 2 activity state change |
| 6 | FDIV1 | FDIV 1 activity state change |
| 5 | FDIV0 | FDIV 0 activity state change |
| 4 | PL160M240 | PLL +240° activity state change |
| 3 | PL160M120 | PLL +120° activity state change |
| 2 | PL160M | PLL activity state change |
| 1 | crystal | Crystal-oscillator activity state change |
| 0 | LP_OSC | Ring-oscillator activity state change |

# UM10316

## Chapter 4: LPC29xx Reset Generation Unit (RGU)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. The USB reset is not available on the LPC2917/19/01 parts.

## 2. Introduction

The RGU is part of the Power Control, Clock, and Reset Subsystem (PCRSS) together with the CGU (see Section 3–3) and PMU.

## 3. RGU functional description

The RGU allows generation of independent reset signals for the following outputs:

Table 32. Reset output configuration

| Reset output | Reset source | Parts of the device reset when activated |
|---|---|---|
| POR_RST | power-on reset module | LP_OSC; source for RGU_RST |
| RGU_RST | POR_RST, RST_N pin | RGU internal; source for PCR_RST |
| PCR_RST | RGU_RST, WATCHDOG | PCR (Power, Clock, and Reset) internal; source for COLD_RST |
| COLD_RST | PCR_RST | parts with COLD_RST as reset source below |
| WARM_RST | COLD_RST | parts with WARM_RST as reset source below |
| SCU_RST | COLD_RST | SCU |
| CFID_RST | COLD_RST | CFID |
| FMC_RST | COLD_RST | embedded Flash-Memory Controller (FMC) |
| EMC_RST | COLD_RST | embedded SRAM-Memory Controller |
| SMC_RST | COLD_RST | external Static-Memory Controller (SMC) |
| GESS_A2V_RST | WARM_RST | GeSS AHB-to-APB bridge |
| PESS_A2V_RST | WARM_RST | PeSS AHB-to-APB bridge |
| GPIO_RST | WARM_RST | all GPIO modules |
| UART_RST | WARM_RST | all UART modules |
| TMR_RST | WARM_RST | all Timer modules in PeSS |
| SPI_RST | WARM_RST | all SPI modules |
| IVNSS_A2V_RST | WARM_RST | IVNSS AHB-to-APB bridge |
| IVNSS_CAN_RST | WARM_RST | all CAN modules including Acceptance filter |
| IVNSS_LIN_RST | WARM_RST | all LIN modules |
| MSCSS_A2V_RST | WARM_RST | MSCSS AHB to APB bridge |
| MSCSS_PWM_RST | WARM_RST | all PWM modules |
| MSCSS_ADC_RST | WARM_RST | all ADC modules |
| MSCSS_TMR_RST | WARM_RST | all Timer modules in MSCSS |
| I2C_RST | WARM_RST | all I2C modules |

**Table 32. Reset output configuration** *…continued*

| Reset output | Reset source | Parts of the device reset when activated |
|---|---|---|
| QEI_RST | WARM_RST | Quadrature encoder |
| DMA_RST | WARM_RST | GPDMA controller |
| USB_RST | WARM_RST | USB controller |
| VIC_RST | WARM_RST | Vectored Interrupt Controller (VIC) |
| AHB_RST | WARM_RST | CPU and AHB Bus infrastructure |

Generation of reset outputs is controlled using registers RESET_CTRL0 and RESET_CTRL1. Note that a POR reset can also be triggered by software.

The RGU monitors the reset cause for each reset output. The reset cause can be retrieved with two levels of granularity.

The first level is monitored by the RESET_STATUS0 to 3 registers and indicates one of the following reset causes (see Table 4–37 to Table 4–40):

- No reset has taken place
- Watchdog reset
- Reset generated by software via RGU register
- Other cause

The second level of granularity is monitored by one individual register for each reset output in which the detailed reset cause is indicated (see Table 4–43 to Table 4–47). Detailed reset causes depend on the reset hierarchy:

- POR reset (does not have a reset source register as it can only be activated by POR)
- RGU reset
- Watchdog reset
- PCR (Power control, Clock, and Reset Subsystem) reset
- Cold reset
- Warm reset

## 3.1 Reset hierarchy

The different types of system reset can be ordered according to their scope. The hierarchy is as follows (see Table 4–33):

1. POR reset: resets everything in the microcontroller.
2. External reset: resets everything in the microcontroller except the OSC 1M oscillator.
3. RGU reset: resets RGU and then has the same effect as Watchdog reset.
4. Watchdog-triggered reset: triggers PCR reset.
5. PCR reset: triggers cold reset and resets Watchdog and flash controller.
6. Cold reset: triggers warm reset and resets GPIO, external memory controller, flash controller, SRAM controller, the SCU, and the CFID.
7. Warm reset: Resets non-memory peripherals (UART, ADC, I2C, timers, etc.). Does **not** reset memory controllers, SCU, CFID or Watchdog.

Table 33. Reset priority

| Priority | Reset | OSC1M | RGU | WDT | GPIO | SCU | Flash controller | CFID | Memory controllers (SRAM,SMC) | all other peripherals |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | POR | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 2 | EXT RESET | no | yes | yes | yes | yes | yes | yes | yes | yes |
| 3 | RGU | no | yes | yes | yes | yes | yes | yes | yes | yes |
| 4 | WDT | no | no | yes | yes | yes | yes | yes | yes | yes |
| 5 | PCR | no | no | yes | yes | yes | yes | yes | yes | yes |
| 6 | Cold | no | no | no | yes | yes | yes | yes | yes | yes |
| 7 | Warm | no | no | no | no | no | no | no | no | yes |

## 4. Register overview

Table 34. Register overview: RGU (base address: 0xFFFF 9000)

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| RESET_CTRL0 | W | 0x100 | Reset control register 0 | - | see Table 4–35 |
| RESET_CTRL1 | W | 0x104 | Reset control register 1 | - | see Table 4–36 |
| RESET_STATUS0 | R/W | 0x110 | Reset status register 0 | 0x140 | see Table 4–37 |
| RESET_STATUS1 | R/W | 0x114 | Reset status register 1 | 0x0 | see Table 4–38 |
| RESET_STATUS2 | R/W | 0x118 | Reset status register 2 | 0x5555 5555 | see Table 4–39 |
| RESET_STATUS3 | R/W | 0x11C | Reset status register 3 | 0x5555 5555 | see Table 4–40 |
| RST_ACTIVE_STATUS0 | R | 0x150 | Reset-Active Status register 0 | 0xFFFF FFFF | see Table 4–41 |
| RST_ACTIVE_STATUS1 | R | 0x154 | Reset-Active Status register 1 | 0xFFFF FFFF | see Table 4–42 |
| RGU_RST_SRC | R/W | 0x404 | Source register for RGU reset | 0x0000 0000 | see Table 4–43 |
| PCR_RST_SRC | R/W | 0x408 | Source register for PCR reset | 0x0000 0000 | see Table 4–44 |
| COLD_RST_SRC | R/W | 0x40C | Source register for COLD reset | 0x0000 0010 | see Table 4–45 |
| WARM_RST_SRC | R/W | 0x410 | Source register for WARM reset | 0x0000 0020 | see Table 4–46 |
| SCU_RST_SRC | R/W | 0x480 | Source register for SCU reset | 0x0000 0020 | see Table 4–46 |
| CFID_RST_SRC | R/W | 0x484 | Source register for CFID reset | 0x0000 0020 | see Table 4–46 |
| FMC_RST_SRC | R/W | 0x490 | Source register for EFC reset | 0x0000 0020 | see Table 4–46 |
| EMC_RST_SRC | R/W | 0x494 | Source register for EMC reset | 0x0000 0020 | see Table 4–46 |
| SMC_RST_SRC | R/W | 0x498 | Source register for SMC reset | 0x0000 0020 | see Table 4–46 |
| GESS_A2V_RST_SRC | R/W | 0x4A0 | Source register for GeSS AHB2APB bridge reset | 0x0000 0040 | see Table 4–47 |
| PESS_A2V_RST_SRC | R/W | 0x4A4 | Source register for PeSS AHB2APB bridge reset | 0x0000 0040 | see Table 4–47 |
| GPIO_RST_SRC | R/W | 0x4A8 | Source register for GPIO reset | 0x0000 0040 | see Table 4–47 |
| UART_RST_SRC | R/W | 0x4AC | Source register for UART reset | 0x0000 0040 | see Table 4–47 |
| TMR_RST_SRC | R/W | 0x4B0 | Source register for Timer reset | 0x0000 0040 | see Table 4–47 |
| SPI_RST_SRC | R/W | 0x4B4 | Source register for SPI reset | 0x0000 0040 | see Table 4–47 |

Table 34. Register overview: RGU (base address: 0xFFFF 9000) *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| IVNSS_A2V_RST_SRC | R/W | 0x4B8 | Source register for IVNSS AHB2APB bridge reset | 0x0000 0040 | see Table 4–47 |
| IVNSS_CAN_RST_SRC | R/W | 0x4BC | Source register for IVNSS CAN reset | 0x0000 0040 | see Table 4–47 |
| IVNSS_LIN_RST_SRC | R/W | 0x4C0 | Source register for IVNSS LIN reset | 0x0000 0040 | see Table 4–47 |
| MSCSS_A2V_RST_SRC | R/W | 0x4C4 | Source register for MSCSS AHB2APB bridge reset | 0x0000 0040 | see Table 4–47 |
| MSCSS_PWM_RST_SRC | R/W | 0x4C8 | Source register for MSCSS PWM reset | 0x0000 0040 | see Table 4–47 |
| MSCSS_ADC_RST_SRC | R/W | 0x4CC | Source register for MSCSS ADC reset | 0x0000 0040 | see Table 4–47 |
| MSCSS_TMR_RST_SRC | R/W | 0x4D0 | Source register for MSCSS Timer reset | 0x0000 0040 | see Table 4–47 |
| I2C_RST_SRC | R/W | 0x4D4 | Source register for I2C reset | 0x0000 0040 | see Table 4–47 |
| QEI_RST_SRC | R/W | 0x4D8 | Source register for QEI reset | 0x0000 0040 | see Table 4–47 |
| DMA_RST_SRC | R/W | 0x4DC | Source register for DMA reset | 0x0000 0040 | see Table 4–47 |
| USB_RST_SRC | R/W | 0x4E0 | Source register for USB reset | 0x0000 0040 | see Table 4–47 |
| VIC_RST_SRC | R/W | 0x4F0 | Source register for VIC reset | 0x0000 0040 | see Table 4–47 |
| AHB_RST_SRC | R/W | 0x4F4 | Source register for AHB reset | 0x0000 0040 | see Table 4–47 |
| BUS_DISABLE | R/W | 0xFF4 | Bus-disable register | 0x0000 0000 | see Table 4–48 |
| reserved | R | 0xFF8 | Reserved | 0x0000 0000 | |
| reserved | R | 0xFFC | Reserved | 0xA098 1000 | |

## 4.1 RGU reset control register

The RGU reset control register allows software to activate and release individual reset outputs. Each bit corresponds to an individual reset output, and writing a ‘1’ activates that output. The reset output is automatically de-activated after a fixed delay period.

Table 35. RESET_CONTROL0 register bit description(RESET_CONTROL0, address 0xFFFF 9100)
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved; do not modify, write as logic 0 |
| 4 | WARM_RST_CTRL | W | - | Activate WARM_RST |
| 3 | COLD_RST_CTRL | W | - | Activate COLD_RST |
| 2 | PCR_RST_CTRL | W | - | Activate PCR_RST |
| 1 | RGU_RST_CTRL | W | - | Activate RGU_RST |
| 0 | reserved | R | - | Reserved; do not modify. Write as logic 0 |

Table 36. RESET_CONTROL1 register bit description (RESET_CONTROL1, 0xFFFF 9104)
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 and 30 | reserved | R | - | Reserved; do not modify, write as logic 0 |
| 29 | AHB_RST_CTRL | W | - | Activate AHB_RST |
| 28 | VIC_RST_CTRL | W | - | Activate VIC_RST |

**Table 36. RESET_CONTROL1 register bit description (RESET_CONTROL1, 0xFFFF 9104)** *…continued*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 27 to 25 | reserved | R | - | Reserved; do not modify. Write as logic 0 |
| 24 | USB | W | - | Activate USB_RST |
| 23 | DMA_RST_CTRL | W | - | Activate DMA_RST |
| 22 | MSCSS_QEI_RST_CTRL | W | - | Activate MSCSS_QEI_RST |
| 21 | IVNSS_I2C_RST_CTRL | W | - | Activate IVNSS_I2C_RST |
| 20 | MSCSS_TMR_RST_CTRL | W | - | Activate MSCSS_TMR_RST |
| 19 | MSCSS_ADC_RST_CTRL | W | - | Activate MSCSS_ADC_RST |
| 18 | MSCSS_PWM_RST_CTRL | W | - | Activate MSCSS_PWM_RST |
| 17 | MSCSS_A2V_RST_CTRL | W | - | Activate MSCSS_A2V_RST |
| 16 | IVNSS_LIN_RST_CTRL | W | - | Activate IVNSS_LIN_RST |
| 15 | IVNSS_CAN_RST_CTRL | W | - | Activate IVNSS_CAN_RST |
| 14 | IVNSS_A2V_RST_CTRL | W | - | Activate IVNSS_A2V_RST |
| 13 | SPI_RST_CTRL | W | - | Activate SPI_RST |
| 12 | TMR_RST_CTRL | W | - | Activate TMR_RST |
| 11 | UART_RST_CTRL | W | - | Activate UART_RST |
| 10 | GPIO_RST_CTRL | W | - | Activate GPIO_RST |
| 9 | PESS_A2V_RST_CTRL | W | - | Activate PESS_A2V_RST |
| 8 | GESS_A2V_RST_CTRL | W | - | Activate GESS_A2V_RST |
| 7 | reserved | R | - | Reserved; do not modify. Write as logic 0 |
| 6 | SMC_RST_CTRL | W | - | Activate SMC_RST |
| 5 | EMC_RST_CTRL | W | - | Activate EMC_RST |
| 4 | FMC_RST_CTRL | W | - | Activate FMC_RST |
| 3 and 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | CFID_RST_CTRL | W | - | Activate CFID_RST |
| 0 | SCU_RST_CTRL | W | - | Activate SCU_RST |

## 4.2 RGU reset status register

The reset status register shows which source (if any) caused the last reset activation per individual reset output of the RGU. When one (or more) inputs of the RGU caused the Reset Output to go active (indicated by value’01’), the respective **_RST_SRC register can be read, see Section 4–4.4. The register is cleared by writing 0000 0000h to it.

**Table 37. RESET_STATUS0 register bit description (RESET_STATUS0, address 0xFFFF 9110)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 9 and 8 | WARM_RST_STAT | R/W | | Status of warm reset |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 7 and 6 | COLD_RST_STAT | R/W | | Status of cold reset |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 5 and 4 | PCR_RST_STAT | R/W | | Status of PCR reset |
| | | | 00* | No reset activated since RGU last came out of reset |
| | | | 01 | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 3 and 2 | RGU_RST_STAT | R/W | | Status of RGU reset |
| | | | 00* | No reset activated since RGU last came out of reset |
| | | | 01 | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 1 and 0 | POR_RST_STAT | R/W | | Status of POR reset |
| | | | 00* | No reset activated since RGU last came out of reset |
| | | | 01 | Power On Reset |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

**Table 38. RESET_STATUS1 register bit description (RESET_STATUS1, address 0xFFFF 9114)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 39. RESET_STATUS2 register bit description (RESET_STATUS2, address 0xFFFF 9118)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 and 30 | IVNSS_CAN_RST_STAT | R/W | | Reset IVNSS CAN status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 29 and 28 | IVNSS_A2V_RST_STAT | R/W | | Reset IVNSS AHB2APB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 27 and 26 | SPI_RST_STAT | R/W | | Reset SPI status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 25 and 24 | TMR_RST_STAT | R/W | | Reset Timer status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 23 and 22 | UART_RST_STAT | R/W | | Reset UART status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 21 and 20 | GPIO_RST_STAT | R/W | | Reset GPIO status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

**Table 39. RESET_STATUS2 register bit description (RESET_STATUS2, address 0xFFFF 9118)** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 19 and 18 | PESS_A2V_RST_STAT | R/W | | Reset PeSS AHB2APB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 17 and 16 | GESS_A2V_RST_STAT | R/W | | Reset GeSS AHB2APB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 15 and 14 | reserved | R | 01* | Reserved; do not modify. |
| 13 and 12 | SMC_RST_STAT | R/W | | Reset SMC status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 11 and 10 | EMC_RST_STAT | R/W | | Reset EMC status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 9 and 8 | FMC_RST_STAT | R/W | | Reset FMC status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 7 to 4 | reserved | R | 0x05* | Reserved |
| 3 and 2 | CFID_RST_STAT | R/W | | Reset CFID status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

Table 39. RESET_STATUS2 register bit description (RESET_STATUS2, address 0xFFFF 9118) *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 and 0 | SCU_RST_STAT | R/W | | Reset SCU status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

Table 40. RESET_STATUS3 register bit description (RESET_STATUS3, address 0xFFFF 911C)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 28 | reserved | R | 0x05* | Reserved; do not modify. Read as logic 0 |
| 27 and 26 | AHB_RST_STAT | R/W | | Reset AHB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 25 and 24 | VIC_RST_STAT | R/W | | Reset INTC status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 23 to 18 | reserved | R | 0x15* | Reserved; do not modify. Read as logic 0 |
| 17 and 16 | USB_STAT | R/W | | Reset USB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 15 and 13 | DMA_STAT | R/W | | Reset DMA status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

**Table 40. RESET_STATUS3 register bit description (RESET_STATUS3, address 0xFFFF 911C)** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 13 and 12 | MSCSS_QEI_STAT | R/W | | Reset MSCSS QEI status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 11 and 10 | IVNSCC_I2C_STAT | R/W | | Reset IVNSCC I2C status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 9 and 8 | MSCSS_TMR_RST_STAT | R/W | | Reset MSCSS Timer status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 7 and 6 | MSCSS_ADC_RST_STAT | R/W | | Reset MSCSS ADC status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 5 and 4 | MSCSS_PWM_RST_STAT | R/W | | Reset MSCSS PWM status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |
| 3 and 2 | MSCSS_A2V_RST_STAT | R/W | | Reset MSCSS AHB2APB status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

**Table 40. RESET_STATUS3 register bit description (RESET_STATUS3, address 0xFFFF 911C)** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 and 0 | IVNSS_LIN_RST_STAT | R/W | | Reset IVNSS LIN status |
| | | | 00 | No reset activated since RGU last came out of reset |
| | | | 01* | Input reset to the RGU |
| | | | 10 | Reserved |
| | | | 11 | Reset control register |

## 4.3 RGU reset active status register

The reset active status register shows the current value of the reset outputs of the RGU. Note that the resets are active LOW.

**Table 41. RST_ACTIVE_STATUS0 register bit description (RST_ACTIVE_STATUS0, address 0xFFFF 9150)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved; do not modify |
| 4 | WARM_RST_STAT | R | 1* | Current state of WARM_RST |
| 3 | COLD_RST_STAT | R | 1* | Current state of COLD_RST |
| 2 | PCR_RST_STAT | R | 1* | Current state of PCR_RST |
| 1 | RGU_RST_STAT | R | 1* | Current state of RGU_RST |
| 0 | POR_RST_STAT | R | 1* | Current state of POR_RST |

**Table 42. RST_ACTIVE_STATUS1 register bit description (RST_ACTIVE_STATUS1, address 0xFFFF 9154)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 and 30 | reserved | R | - | Reserved; do not modify |
| 29 | AHB_RST_STAT | R | 1* | Current state of AHB_RST |
| 28 | VIC_RST_STAT | R | 1* | Current state of VIC_RST |
| 27 to 25 | reserved | R | - | Reserved; do not modify |
| 24 | USB_RST_STAT | W | - | Current state of DMA_RST |
| 23 | DMA_RST_STAT | W | - | Current state of DMA_RST |
| 22 | MSCSS_QEI_RST_STAT | W | - | Current state of MSCSS_QEI_RST |
| 21 | IVNSS_I2C_RST_STAT | W | - | Current state of IVNSS_I2C_RST |
| 20 | MSCSS_TMR_RST_STAT | R | 1* | Current state of MSCSS_TMR_RST |
| 19 | MSCSS_ADC_RST_STAT | R | 1* | Current state of MSCSS_ADC_RST |
| 18 | MSCSS_PWM_RST_STAT | R | 1* | Current state of MSCSS_PWM_RST |
| 17 | MSCSS_A2V_RST_STAT | R | 1* | Current state of MSCSS_A2V_RST |
| 16 | IVNSS_LIN_RST_STAT | R | 1* | Current state of IVNSS_LIN_RST |
| 15 | IVNSS_CAN_RST_STAT | R | 1* | Current state of IVNSS_CAN_RST |
| 14 | IVNSS_A2V_RST_STAT | R | 1* | Current state of IVNSS_A2V_RST |

**Table 42. RST_ACTIVE_STATUS1 register bit description (RST_ACTIVE_STATUS1, address 0xFFFF 9154)** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 13 | SPI_RST_STAT | R | 1* | Current state of SPI_RST |
| 12 | TMR_RST_STAT | R | 1* | Current state of TMR_RST |
| 11 | UART_RST_STAT | R | 1* | Current state of UART_RST |
| 10 | GPIO_RST_STAT | R | 1* | Current state of GPIO_RST |
| 9 | PESS_A2V_RST_STAT | R | 1* | Current state of PESS_A2V_RST |
| 8 | GESS_A2V_RST_STAT | R | 1* | Current state of GESS_A2V_RST |
| 7 | reserved | R | - | Reserved; do not modify |
| 6 | SMC_RST_STAT | R | 1* | Current state of SMC_RST |
| 5 | EMC_RST_STAT | R | 1* | Current state of EMC_RST |
| 4 | FMC_RST_STAT | R | 1* | Current state of FMC_RST |
| 3 and 2 | reserved | R | - | Reserved; do not modify |
| 1 | CFID_RST_STAT | R | 1* | Current state of CFID_RST |
| 0 | SCU_RST_STAT | R | 1* | Current state of SCU_RST |

## 4.4 RGU reset source registers

The reset source register indicates for each RGU reset output which specific reset input caused it to go active.

### POR reset

**Remark:** The POR_RST reset output of the RGU does not have a source register as it can only be activated by the POR reset module.

### RGU reset

The following reset source register description is applicable to the RGU reset output of the RGU, which is activated by the RST_N input pin or the POR reset, see Table 10–90. To be able to detect the source of the next PCR reset the register should be cleared by writing a 1 after read.

**Table 43. RGU_RST_SRC register bit description (RGU_RST_SRC, address 0xFFFF 9404)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | RSTN_PIN | R/W | 0* | Reset activated by external input reset |
| 0 | POR | R/W | 0* | Reset activated by power-on-reset |

### PCR reset

The following reset source register description is applicable to the PCR reset output of the RGU, which is activated by the Watchdog Timer or the RGU reset, see Table 10–90. To be able to detect the source of the next PCR reset the register should be cleared by writing a 1 after read.

**Table 44. PCR_RST_SRC register bit description (PCR_RST_SRC, address 0xFFFF 9408)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 3 | WDT_TMR | R/W | 0* | Reset activated by Watchdog timer (WDT) |
| 2 | RGU | R/W | 0* | Reset activated by RGU reset |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

### Cold reset

The following reset source register description is applicable for the COLD reset output of the RGU, that is activated by the PCR reset, see Table 10–90. To be able to detect the source of the next COLD reset the register should be cleared by writing a 0 after read.

**Table 45. COLD_RST_SRC register bit description (COLD_RST_SRC, address 0xFFFF 940C)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 4 | PCR | R/W | 1* | Reset activated by PCR reset |
| 3 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

### Peripherals activated by cold reset

The following reset source register description is applicable to all the reset outputs of the RGU that are activated by the COLD reset, see Table 10–90. To be able to detect the next reset the register should be cleared by writing a 0 after read.

**Table 46. XX_RST_SRC register bit description (WARM_RST_SRC to SMC_RST_SRC, addresses 0xFFFF 9410 to 0xFFFF 9498)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 | COLD | R/W | 1* | Reset activated by COLD reset |
| 4 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

### Peripherals activated by warm reset

The following reset source register description is applicable to all the reset outputs of the RGU that are activated by the WARM reset, see Table 10–90. To be able to detect the next reset the register should be cleared by writing a 0 after read.

**Table 47. YY_RST_SRC register bit description (GESS_A2V_RST_SRC to AHB_RST_SRC, address 0xFFFF 94A0 to 0xFFFF 94F4)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 7 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 6 | WARM | R/W | 1* | Reset activated by WARM reset |
| 5 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 4.5 RGU bus-disable register

The BUS_DISABLE register prevents any register in the CGU from being written to.

**Table 48. BUS_DISABLE register bit description (BUS_DISABLE, address 0xFFFF 9FF4)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | RRBUS | R/W | | Bus write-disable bit |
| | | | 1 | No writes to registers within RGU are possible (except the BUS_DISABLE register) |
| | | | 0* | Normal operation |

# UM10316

## Chapter 5: LPC29xx Power Management Unit (PMU)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The implementation of some branch clocks for power control depends on the peripheral and memory configuration of each LPC29xx part, see Table 5–49. All other branch clocks are available in all LPC29xx parts.

**Table 49. Branch clocks implemented in LPC29xx (x = CLK_CFG_ or CLK_STAT_)**

| Part | SRAM | | Flash | USB | | | GPIO | ADC | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | xRAM0 | xRAM1 | xFMC | xUSB_CLK | xUSB_ I2C_CLK | xUSB | xGPIO | xADC0, xADC0_ APB | xADC1, xADC1_ APB | xADC2, xADC2_ APB |
| LPC2917/19/01 | yes | yes | yes | no | no | no | 0/1/2/3 | no | yes | yes |
| LPC2921/23 | yes | no | yes | yes | no | yes | 0/1/5 | no | yes | yes |
| LPC2925 | yes | yes | yes | yes | no | yes | 0/1/5 | no | yes | yes |
| LPC2926/27/29 | yes | yes | yes | yes | yes | yes | 0/1/2/3/5 | yes | yes | yes |
| LPC2930 | yes | yes | yes[1] | yes | yes | yes | 0/1/2/3/4/5 | yes | yes | yes |
| LPC2939 | yes | yes | yes | yes | yes | yes | 0/1/2/3/4/5 | yes | yes | yes |

[1] The flash clock is connected to the boot ROM for the flashless LPC2930. The clock can be switched off to conserve power after the boot process has completed.

## 2. Introduction

The PMU is part of the Power Control and Reset Subsystem (PCRSS) together with the CGU0/1 (see Section 3–2) and RGU (see Section 4–2).

## 3. PMU functional description

**Table 50. Branch clock overview**

*Legend:*
*'1' Indicates that the related register bit is tied off to logic HIGH, all writes are ignored*
*'0' Indicates that the related register bit is tied off to logic LOW, all writes are ignored*
*'+' Indicates that the related register bit is readable and writable*

| Base clock | Branch clock name/clock leafs | Implemented switch on/off mechanism | | |
|---|---|---|---|---|
| | | WAKE-UP | AUTO | RUN |
| BASE_SAFE_CLK | CLK_SAFE | 0 | 0 | 1 |

**Table 50. Branch clock overview** *…continued*

*Legend:*

*‘1’ Indicates that the related register bit is tied off to logic HIGH, all writes are ignored*

*‘0’ Indicates that the related register bit is tied off to logic LOW, all writes are ignored*

*‘+’ Indicates that the related register bit is readable and writable*

| Base clock | Branch clock name/clock leafs | Implemented switch on/off mechanism | | |
|---|---|---|---|---|
| | | WAKE-UP | AUTO | RUN |
| BASE_SYS_CLK | CLK_SYS_CPU | + | + | 1 |
| | CLK_SYS | + | + | 1 |
| | CLK_SYS_PCR | + | + | 1 |
| | CLK_SYS_FMC[1] | + | + | + |
| | CLK_SYS_RAM0 | + | + | + |
| | CLK_SYS_RAM1 | + | + | + |
| | CLK_SYS_SMC | + | + | + |
| | CLK_SYS_GESS | + | + | + |
| | CLK_SYS_VIC | + | + | + |
| | CLK_SYS_PESS | + | + | + |
| | CLK_SYS_GPIO0 | + | + | + |
| | CLK_SYS_GPIO1 | + | + | + |
| | CLK_SYS_GPIO2 | + | + | + |
| | CLK_SYS_GPIO3 | + | + | + |
| | CLK_SYS_IVNSS_A | + | + | + |
| | CLK_SYS_MSCSS_A | + | + | + |
| | CLK_SYS_GPIO4 | + | + | + |
| | CLK_SYS_GPIO5 | + | + | + |
| | CLK_SYS_DMA | + | + | + |
| | CLK_SYS_USB | + | + | + |
| BASE_PCR_CLK | CLK_PCR_SLOW | + | + | 1 |
| BASE_IVNSS_CLK | CLK_IVNSS_APB | + | + | + |
| | CLK_IVNSS_CANCA | + | + | + |
| | CLK_IVNSS_CANC0 | + | + | + |
| | CLK_IVNSS_CANC1 | + | + | + |
| | CLK_IVNSS_I2C0 | + | + | + |
| | CLK_IVNSS_I2C1 | + | + | + |
| | CLK_IVNSS_LIN0 | + | + | + |
| | CLK_IVNSS_LIN1 | + | + | + |
| BASE_MSCSS_CLK | CLK_MSCSS_APB | + | + | + |
| | CLK_MSCSS_MTMR0 | + | + | + |
| | CLK_MSCSS_MTMR1 | + | + | + |
| | CLK_MSCSS_PWM0 | + | + | + |
| | CLK_MSCSS_PWM1 | + | + | + |
| | CLK_MSCSS_PWM2 | + | + | + |
| | CLK_MSCSS_PWM3 | + | + | + |

**Table 50. Branch clock overview** *...continued*

*Legend:*
*'1' Indicates that the related register bit is tied off to logic HIGH, all writes are ignored*
*'0' Indicates that the related register bit is tied off to logic LOW, all writes are ignored*
*'+' Indicates that the related register bit is readable and writable*

| Base clock | Branch clock name/clock leafs | Implemented switch on/off mechanism | | |
|---|---|---|---|---|
| | | WAKE-UP | AUTO | RUN |
| BASE_MSCSS_CLK | CLK_MSCSS_ADC0_APB | + | + | + |
| | CLK_MSCSS_ADC1_APB | + | + | + |
| | CLK_MSCSS_ADC2_APB | + | + | + |
| | CLK_MSCSS_QEI | + | + | + |
| BASE_OUT_CLK | CLK_OUT_CLK | + | + | + |
| BASE_UART_CLK | CLK_UART0 | + | + | + |
| | CLK_UART1 | + | + | + |
| BASE_SPI_CLK | CLK_SPI0 | + | + | + |
| | CLK_SPI1 | + | + | + |
| | CLK_SPI2 | + | + | + |
| BASE_TMR_CLK | CLK_TMR0 | + | + | + |
| | CLK_TMR1 | + | + | + |
| | CLK_TMR2 | + | + | + |
| | CLK_TMR3 | + | + | + |
| BASE_ADC_CLK | CLK_ADC0 | + | + | + |
| | CLK_ADC1 | + | + | + |
| | CLK_ADC2 | + | + | + |
| BASE_TEST_CLK | CLK_TSSHELL | - | - | - |
| BASE_USB_I2C_CLK | CLK_USB_I2C | + | + | + |
| BASE_USB_CLK | CLK_USB | + | + | + |

[1] The flash clock is connected to the boot ROM for the flashless LPC2930. The clock can be switched off to conserve power after the boot process has completed.

The PMU allows definition of the power mode for each individual clock leaf. The clock leaves are divided into branches as follows:

## 3.1 PMU clock-branch run mode

- the clock should be running
- the clock leaf should be disabled by the AHB automatic-switching setting
- the leaf should follow the system in entering sleep mode and waiting for a wake-up

All these settings can be controlled via the corresponding registers CLK_CFG_<leaf>.

The following clock leaves are exceptions to the general rule:

- CLK_SYS_CPU – cannot be disabled.
- CLK_SYS – cannot be disabled.
- CLK_SYS_PCR – cannot be disabled.

Clocks that have been programmed to enter sleep mode follow the chosen setting of the PD field in register PM. This means that with a single write-action all of these domains can be set either to sleep or to wake up.

Since application of configuration settings may not be instantaneous, the current setting can be read in register CLK_STAT_<leaf>. The registers CLK_STAT_<leaf> indicate the configured settings and in field STATEM_STAT the current setting. The possible states are:

- run – normal clock enabled.
- wait – request has been sent to AHB to disable the clock but is waiting to be granted.
- sleep0 – clock disabled.
- sleep1 – clock disabled and request removed.

### 3.2 PMU clock branch overview

Within each clock branch the PMU keeps an overview of the power state of the separate leaves. This indication can be used to determine whether the clock to a branch can be safely disabled. This overview is kept in register BASE_STAT and contains one bit per clock branch.

## 4. Register overview

**Table 51. Register overview: PMU (base address: 0xFFFF A000)**

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| PM | R/W | 0x000 | 0x0000 0000 | Power mode register | see Table 5–52 |
| BASE_STAT | R | 0x004 | 0x0000 1FFF | Base-clock status register | see Table 5–53 |
| CLK_CFG_SAFE | R/W | 0x100 | 0x0000 0001 | Safe-clock configuration register | see Table 5–54 |
| CLK_STAT_SAFE | R | 0x104 | 0x0000 0001 | Safe-clock status register | see Table 5–55 |
| CLK_CFG_CPU | R/W | 0x200 | 0x0000 0001 | CPU-clock configuration register | see Table 5–54 |
| CLK_STAT_CPU | R | 0x204 | 0x0000 0001 | CPU-clock status register | see Table 5–55 |
| CLK_CFG_SYS | R/W | 0x208 | 0x0000 0001 | System-clock configuration register | see Table 5–54 |
| CLK_STAT_SYS | R | 0x20C | 0x0000 0001 | System-clock status register | see Table 5–55 |
| CLK_CFG_PCR | R/W | 0x210 | 0x0000 0001 | System-clock_pcr configuration register | see Table 5–54 |
| CLK_STAT_PCR | R | 0x214 | 0x0000 0001 | System-clock_pcr status register | see Table 5–55 |
| CLK_CFG_FMC | R/W | 0x218 | 0x0000 0001 | Flash-clock configuration register | see Table 5–54 |
| CLK_STAT_FMC | R | 0x21C | 0x0000 0001 | Flash-clock status register | see Table 5–55 |
| CLK_CFG_RAM0 | R/W | 0x220 | 0x0000 0001 | AHB clock to embedded memory controller 0 configuration register | see Table 5–54 |
| CLK_STAT_RAM0 | R | 0x224 | 0x0000 0001 | AHB clock to embedded memory controller 0 status register | see Table 5–55 |
| CLK_CFG_RAM1 | R/W | 0x228 | 0x0000 0001 | AHB clock to embedded memory controller 1 configuration register | see Table 5–54 |
| CLK_STAT_RAM1 | R | 0x22C | 0x0000 0001 | AHB clock to embedded memory controller 1 status register | see Table 5–55 |

**Table 51. Register overview: PMU (base address: 0xFFFF A000)** *…continued*

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| CLK_CFG_SMC | R/W | 0x230 | 0x0000 0001 | AHB clock to Static Memory Controller configuration register | see Table 5–54 |
| CLK_STAT_SMC | R | 0x234 | 0x0000 0001 | AHB clock to Static Memory Controller status register | see Table 5–55 |
| CLK_CFG_GESS | R/W | 0x238 | 0x0000 0001 | AHB/APB clock to GeSS module configuration register | see Table 5–54 |
| CLK_STAT_GESS | R | 0x23C | 0x0000 0001 | AHB/APB clock to GeSS module status register | see Table 5–55 |
| CLK_CFG_VIC | R/W | 0x240 | 0x0000 0001 | AHB/DTL clock to interrupt controller configuration register | see Table 5–54 |
| CLK_STAT_VIC | R | 0x244 | 0x0000 0001 | AHB/DTL clock to interrupt controller status register | see Table 5–55 |
| CLK_CFG_PESS | R/W | 0x248 | 0x0000 0001 | AHB/APB clock to PeSS module configuration register | see Table 5–54 |
| CLK_STAT_PESS | R | 0x24C | 0x0000 0001 | AHB/APB clock to PeSS module status register | see Table 5–55 |
| CLK_CFG_GPIO0 | R/W | 0x250 | 0x0000 0001 | APB clock to General-Purpose I/O 0 configuration register | see Table 5–54 |
| CLK_STAT_GPIO0 | R | 0x254 | 0x0000 0001 | APB clock to General-Purpose I/O 0 status register | see Table 5–55 |
| CLK_CFG_GPIO1 | R/W | 0x258 | 0x0000 0001 | APB clock to General-Purpose I/O 1 configuration register | see Table 5–54 |
| CLK_STAT_GPIO1 | R | 0x25C | 0x0000 0001 | APB clock to General-Purpose I/O 1 status register | see Table 5–55 |
| CLK_CFG_GPIO2 | R/W | 0x260 | 0x0000 0001 | APB clock to General-Purpose I/O 2 configuration register | see Table 5–54 |
| CLK_STAT_GPIO2 | R | 0x264 | 0x0000 0001 | APB clock to General-Purpose I/O 2 status register | see Table 5–55 |
| CLK_CFG_GPIO3 | R/W | 0x268 | 0x0000 0001 | APB clock to General-Purpose I/O 3 status register | see Table 5–54 |
| CLK_STAT_GPIO3 | R | 0x26C | 0x0000 0001 | APB clock to General-Purpose I/O 3 status register | see Table 5–55 |
| CLK_CFG_IVNSS_A | R/W | 0x270 | 0x0000 0001 | AHB clock to IVNSS module-configuration register | see Table 5–54 |
| CLK_STAT_IVNSS_A | R | 0x274 | 0x0000 0001 | AHB clock to IVNSS module-status register | see Table 5–55 |
| CLK_CFG_MSCSS_A | R/W | 0x278 | 0x0000 0001 | AHB/APB clock to MSCSS module-configuration register | see Table 5–54 |
| CLK_STAT_MSCSS_A | R | 0x27C | 0x0000 0001 | AHB/APB clock to MSCSS module-status register | see Table 5–55 |
| CLK_CFG_GPIO4 | R/W | 0x280 | 0x0000 0001 | APB clock to General-Purpose I/O 4 status register | see Table 5–54 |
| CLK_STAT_GPIO4 | R | 0x284 | 0x0000 0001 | APB clock to General-Purpose I/O 4 status register | see Table 5–55 |
| CLK_CFG_GPIO5 | R/W | 0x288 | 0x0000 0001 | APB clock to General-Purpose I/O 5 status register | see Table 5–54 |

Table 51. Register overview: PMU (base address: 0xFFFF A000) *…continued*

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| CLK_STAT_GPIO5 | R | 0x28C | 0x0000 0001 | APB clock to General-Purpose I/O 5 status register | see Table 5–55 |
| CLK_CFG_DMA | R/W | 0x290 | 0x0000 0001 | GPDMA clock configuration register | see Table 5–54 |
| CLK_STAT_DMA | R | 0x294 | 0x0000 0001 | GPDMA clock status register | see Table 5–55 |
| CLK_CFG_USB | R/W | 0x298 | 0x0000 0001 | USB register interface clock configuration register | see Table 5–54 |
| CLK_STAT_USB | R | 0x29C | 0x0000 0001 | USB register interface status register | see Table 5–55 |
| CLK_CFG_PCR_IP | R/W | 0x300 | 0x0000 0001 | IP clock to PCR module configuration-register | see Table 5–54 |
| CLK_STAT_PCR_IP | R | 0x304 | 0x0000 0001 | IP clock to PCR module-status register | see Table 5–55 |
| CLK_CFG_IVNSS_APB | R/W | 0x400 | 0x0000 0001 | APB clock to IVNSS module-configuration register | see Table 5–54 |
| CLK_STAT_IVNSS_APB | R | 0x404 | 0x0000 0001 | APB clock to IVNSS module status-register | see Table 5–55 |
| CLK_CFG_CANCA | R/W | 0x408 | 0x0000 0001 | IP clock to CAN gateway acceptance-filter configuration register | see Table 5–54 |
| CLK_STAT_CANCA | R | 0x40C | 0x0000 0001 | IP clock to CAN gateway acceptance-filter status register | see Table 5–55 |
| CLK_CFG_CANC0 | R/W | 0x410 | 0x0000 0001 | IP clock to CAN gateway 0 configuration register | see Table 5–54 |
| CLK_STAT_CANC0 | R | 0x414 | 0x0000 0001 | IP clock to CAN gateway 0 status register | see Table 5–55 |
| CLK_CFG_CANC1 | R/W | 0x418 | 0x0000 0001 | IP clock to CAN gateway 1 configuration register | see Table 5–54 |
| CLK_STAT_CANC1 | R | 0x41C | 0x0000 0001 | IP clock to CAN gateway 1 status register | see Table 5–55 |
| CLK_CFG_I2C0 | R/W | 0x420 | 0x0000 0001 | IP clock to I2C0 configuration register | see Table 5–54 |
| CLK_STAT_I2C0 | R | 0x424 | 0x0000 0001 | IP clock to I2C0 status register | see Table 5–55 |
| CLK_CFG_I2C1 | R/W | 0x428 | 0x0000 0001 | IP clock to I2C1 configuration register | see Table 5–54 |
| CLK_STAT_I2C1 | R | 0x42C | 0x0000 0001 | IP clock to I2C1 status register | see Table 5–55 |
| - | - | 0x430 - 0x43C | - | reserved | - |
| CLK_CFG_LIN0 | R/W | 0x440 | 0x0000 0001 | IP clock to LIN controller 0 configuration register | see Table 5–54 |
| CLK_STAT_LIN0 | R | 0x444 | 0x0000 0001 | IP clock to LIN controller 0 status register | see Table 5–55 |
| CLK_CFG_LIN1 | R/W | 0x448 | 0x0000 0001 | IP clock to LIN controller 1 configuration register | see Table 5–54 |
| CLK_STAT_LIN1 | R | 0x44C | 0x0000 0001 | IP clock to LIN controller 1 status register | see Table 5–55 |
| - | - | 0x450 -0x4FC | - | reserved | - |
| CLK_CFG_MSCSS_APB | R/W | 0x500 | 0x0000 0001 | APB clock to MSCSS module-configuration register | see Table 5–54 |

**Table 51. Register overview: PMU (base address: 0xFFFF A000)** *...continued*

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| CLK_STAT_MSCSS_APB | R | 0x504 | 0x0000 0001 | APB clock to MSCSS module-status register | see Table 5–55 |
| CLK_CFG_MTMR0 | R/W | 0x508 | 0x0000 0001 | IP clock to timer 0 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_MTMR0 | R | 0x50C | 0x0000 0001 | IP clock to timer 0 in MSCSS status register | see Table 5–55 |
| CLK_CFG_MTMR1 | R/W | 0x510 | 0x0000 0001 | IP clock to timer 1 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_MTMR1 | R | 0x514 | 0x0000 0001 | IP clock to timer 1 in MSCSS status register | see Table 5–55 |
| CLK_CFG_PWM0 | R/W | 0x518 | 0x0000 0001 | IP clock to PWM 0 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_PWM0 | R | 0x51C | 0x0000 0001 | IP clock to PWM 0 in MSCSS status register | see Table 5–55 |
| CLK_CFG_PWM1 | R/W | 0x520 | 0x0000 0001 | IP clock to PWM 1 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_PWM1 | R | 0x524 | 0x0000 0001 | IP clock to PWM 1 in MSCSS status register | see Table 5–55 |
| CLK_CFG_PWM2 | R/W | 0x528 | 0x0000 0001 | IP clock to PWM 2 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_PWM2 | R | 0x52C | 0x0000 0001 | IP clock to PWM 2 in MSCSS status register | see Table 5–55 |
| CLK_CFG_PWM3 | R/W | 0x530 | 0x0000 0001 | IP clock to PWM 3 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_PWM3 | R | 0x534 | 0x0000 0001 | IP clock to PWM 3 in MSCSS status register | see Table 5–55 |
| CLK_CFG_ADC0_APB | R/W | 0x538 | 0x0000 0001 | APB clock to ADC 0 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_ADC0_APB | R | 0x53C | 0x0000 0001 | APB clock to ADC 0 in MSCSS status register | see Table 5–55 |
| CLK_CFG_ADC1_APB | R/W | 0x540 | 0x0000 0001 | APB clock to ADC 1 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_ADC1_APB | R | 0x544 | 0x0000 0001 | APB clock to ADC 1 in MSCSS status register | see Table 5–55 |
| CLK_CFG_ADC2_APB | R/W | 0x548 | 0x0000 0001 | APB clock to ADC 2 in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_ADC2_APB | R | 0x54C | 0x0000 0001 | APB clock to ADC 2 in MSCSS status register | see Table 5–55 |
| CLK_CFG_QEI_APB | R/W | 0x550 | 0x0000 0001 | APB clock to QEI in MSCSS configuration register | see Table 5–54 |
| CLK_STAT_QEI_APB | R | 0x554 | 0x0000 0001 | APB clock to QEI in MSCSS status register | see Table 5–55 |
| reserved | R/W | 0x558 - 0x5FF | 0x0000 0001 | Reserved | - |
| CLK_CFG_OUT_CLK | R/W | 0x600 | 0x0000 0001 | clock out configuration register | see Table 5–54 |
| CLK_STAT_OUT_CLK | R | 0x604 | 0x0000 0001 | clock out status register | see Table 5–55 |

Table 51. Register overview: PMU (base address: 0xFFFF A000) *…continued*

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| CLK_CFG_UART0 | R/W | 0x700 | 0x0000 0001 | IP clock to UART-0 configuration register | see Table 5–54 |
| CLK_STAT_UART0 | R | 0x704 | 0x0000 0001 | IP clock to UART-0 status register | see Table 5–55 |
| CLK_CFG_UART1 | R/W | 0x708 | 0x0000 0001 | IP clock to UART 1 configuration register | see Table 5–54 |
| CLK_STAT_UART1 | R | 0x70C | 0x0000 0001 | IP clock to UART 1 status register | see Table 5–55 |
| CLK_CFG_SPI0 | R/W | 0x800 | 0x0000 0001 | IP clock to SPI 0 configuration register | see Table 5–54 |
| CLK_STAT_SPI0 | R | 0x804 | 0x0000 0001 | IP clock to SPI 0 status register | see Table 5–55 |
| CLK_CFG_SPI1 | R/W | 0x808 | 0x0000 0001 | IP clock to SPI 1 configuration register | see Table 5–54 |
| CLK_STAT_SPI1 | R | 0x80C | 0x0000 0001 | IP clock to SPI 1 status register | see Table 5–55 |
| CLK_CFG_SPI2 | R/W | 0x810 | 0x0000 0001 | IP clock to SPI 2 configuration register | see Table 5–54 |
| CLK_STAT_SPI2 | R | 0x814 | 0x0000 0001 | IP clock to SPI 2 status register | see Table 5–55 |
| CLK_CFG_TMR0 | R/W | 0x900 | 0x0000 0001 | IP clock to Timer 0 configuration register | see Table 5–54 |
| CLK_STAT_TMR0 | R | 0x904 | 0x0000 0001 | IP clock to Timer 0 status register | see Table 5–55 |
| CLK_CFG_TMR1 | R/W | 0x908 | 0x0000 0001 | IP clock to Timer 1 configuration register | see Table 5–54 |
| CLK_STAT_TMR1 | R | 0x90C | 0x0000 0001 | IP clock to Timer 1 status register | see Table 5–55 |
| CLK_CFG_TMR2 | R/W | 0x910 | 0x0000 0001 | IP clock to Timer 2 configuration register | see Table 5–54 |
| CLK_STAT_TMR2 | R | 0x914 | 0x0000 0001 | IP clock to Timer 2 status register | see Table 5–55 |
| CLK_CFG_TMR3 | R/W | 0x918 | 0x0000 0001 | IP clock to Timer 3 configuration register | see Table 5–54 |
| CLK_STAT_TMR3 | R | 0x91C | 0x0000 0001 | IP clock to Timer 3 status register | see Table 5–55 |
| CLK_CFG_ADC0 | R/W | 0xA00 | 0x0000 0001 | IP clock to ADC 0 status register | see Table 5–54 |
| CLK_STAT_ADC0 | R | 0xA04 | 0x0000 0001 | IP clock to ADC 0 status register | see Table 5–55 |
| CLK_CFG_ADC1 | R/W | 0xA08 | 0x0000 0001 | IP clock to ADC 1 status register | see Table 5–54 |
| CLK_STAT_ADC1 | R | 0xA0C | 0x0000 0001 | IP clock to ADC 1 status register | see Table 5–55 |
| CLK_CFG_ADC2 | R/W | 0xA10 | 0x0000 0001 | IP clock to ADC 2 configuration register | see Table 5–54 |
| CLK_STAT_ADC2 | R | 0xA14 | 0x0000 0001 | IP clock to ADC 2 status register | see Table 5–55 |
| CLK_CFG_TSSHELL | R/W | 0xB00 | 0x0000 0001 | IP clock to test clock configuration register.<br><br>**Remark:** This is an internal clock used for testing only. It is running at start-up and should be disabled using this register. | see Table 5–54 |
| CLK_STAT_TSSHELL | R | 0xB04 | 0x0000 0001 | IP clock to test clock status register | see Table 5–55 |
| CLK_CFG_USB_I2C | R/W | 0xC00 | 0x0000 0001 | IP clock to USB I2C configuration register | see Table 5–54 |
| CLK_STAT_USB_I2C | R | 0xC04 | 0x0000 0001 | IP clock to USB I2C status register | see Table 5–55 |
| CLK_CFG_USB_CLK | R/W | 0xD00 | 0x0000 0001 | IP clock to USB CLK configuration register | see Table 5–54 |

**Table 51. Register overview: PMU (base address: 0xFFFF A000)** *…continued*

| Name | Access | Address offset | Reset value | Description | Reference |
|---|---|---|---|---|---|
| CLK_STAT_USB_CLK | R | 0xD04 | 0x0000 0001 | IP clock to USB CLK status register | see Table 5–55 |
| reserved | - | 0xFF8 | 0x0000 0000 | Reserved | |
| reserved | - | 0xFFC | 0xA0B6 0000 | Reserved | |

## 4.1 Power mode register (PM)

This register contains a single bit, PD, which when set disables all output clocks with wake-up enabled. Clocks disabled by the power-down mechanism are reactivated when a wake-up interrupt is detected or when a 0 is written to the PD bit.

**Table 52. PM register bit description (PM, address 0xFFFF A000)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | PD | R/W | | Initiate power-down mode: |
| | | | 1 | Clocks with wake-up mode enabled (WAKEUP=1) are disabled |
| | | | 0* | Normal operation |

## 4.2 Base-clock status register

Each bit in this register indicates whether the specified base clock can be safely switched off. A logic zero indicates that all branch clocks generated from this base clock are disabled, so the base clock can also be switched off. A logic 1 value indicates that there is still at least one branch clock running.

**Table 53. BASE_STAT register bit description (BASE_STAT, address 0xFFFF A004)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 13 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 12 | BASE12_STA | R | 1* | Indicator for BASE_USB_CLK |
| 11 | BASE11_STAT | R | 1* | Indicator for BASE_USB_I2C_CLK |
| 10 | BASE10_STAT | R | 1* | Indicator for BASE_CLK_TESTSHELL |
| 9 | BASE9_STAT | R | 1* | Indicator for BASE_ADC_CLK |
| 8 | BASE8_STAT | R | 1* | Indicator for BASE_TMR_CLK |
| 7 | BASE7_STAT | R | 1* | Indicator for BASE_SPI_CLK |
| 6 | BASE6_STAT | R | 1* | Indicator for BASE_UART_CLK |
| 5 | BASE5_STAT | R | 1* | Indicator for BASE_OUT_CLK |
| 4 | BASE4_STAT | R | 1* | Indicator for BASE_MSCSS_CLK |
| 3 | BASE3_STAT | R | 1* | Indicator for BASE_IVNSS_CLK |
| 2 | BASE2_STAT | R | 1* | Indicator for BASE_PCR_CLK |
| 1 | BASE1_STAT | R | 1* | Indicator for BASE_SYS_CLK |
| 0 | BASE0_STAT | R | 1* | Indicator for BASE_SAFE_CLK |

## 4.3 PMU clock configuration register for output branches

Each generated output clock from the PMU has a configuration register.

**Table 54. CLK_CFG_XXX register bit description (CLK_CFG_SAFE to CLK_CFG_USB_CLK, addresses 0xFFFF A100 to 0xFFFF AD00)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 2 | WAKEUP[1] | R/W | 1 | The branch clock is 'wake-up enabled'. When the PD bit in the Power Mode register (see Section 5–4.1) is set, and clocks which are wake-up enabled are switched off. These clocks will be switched on if a wake-up event is detected or if the PD bit is cleared. If register bit AUTO is set, the AHB disable protocol must complete before the clock is switched off. |
| | | | 0* | PD bit has no influence on this branch clock |
| 1 | AUTO[1] | R/W | 1 | Enable auto (AHB disable mechanism). The PMU initiates the AHB disable protocol before switching the clock off. This protocol ensures that all AHB transactions have been completed before turning the clock off |
| | | | 0* | No AHB disable protocol is used. |
| 0 | RUN[2] | R/W | 1* | The WAKEUP, PD (and AUTO) control bits determine the activation of the branch clock. If register bit AUTO is set the AHB disable protocol must complete before the clock is switched off. |
| | | | 0 | Branch clock switched off |

[1] Tied off to logic LOW for some branch clocks. All writes are ignored for those with tied bits.

[2] Tied off to logic HIGH for some branch clocks. All writes are ignored for those with tied bits.

## 4.4 Status register for output branch clock

Like the configuration register, each generated output clock from the PMU has a status register. When the configuration register of an output clock is written to the value of the actual hardware signals may not be updated immediately. This may be due to the auto or wake-up mechanism. The status register shows the current value of these signals.

Table 55. CLK_STAT_XXX register bit description (CLK_STAT_SAFE to CLK_STAT_USB_CLK, addresses 0xFFFF A104 to 0xFFFF AD04)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 and 8 | SM | R | | Status of state machine controlling the clock-enable signal |
| | | | 00* | RUN = clock enabled |
| | | | 01 | WAIT = request sent to AHB master to disable clock. Waiting for AHB master to grant the request |
| | | | 10 | SLEEP1 = clock disabled and request removed |
| | | | 11 | SLEEP0 = clock disabled |
| 7 to 3 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 2 | WS | R | | Wake-up mechanism enable status |
| | | | 1 | Enabled |
| | | | 0* | Not enabled |
| 1 | AS | R | | Auto (AHB disable mechanism) enable status |
| | | | 1 | Enabled |
| | | | 0* | Not enabled |
| 0 | RS | R | | Run-enable status |
| | | | 1* | Enabled |
| | | | 0 | Not enabled |

# UM10316

## Chapter 6: LPC29xx System Control Unit (SCU)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. See Table 6–56 for available GPIO pins for the port selection registers and for registers that are part specific.

**Table 56. LPC29xx SCU usage**

| Part number | GPIO port 0 | GPIO port 1 | GPIO port 2 | GPIO port 3 | GPIO port 4 | GPIO port 5 | USB D+/- port selection registers | USB SSMM3/ SMP3 registers |
|---|---|---|---|---|---|---|---|---|
| LPC2917/19/01 | P0[31:0] | P1[31:0] | P2[27:0] | P3[15:0] | - | - | - | no |
| LPC2921/23/25 | P0[31:0] | P1[27:0] | - | - | - | P5[19:18] | for P5[19:18] | yes |
| LPC2926/27/29 | P0[31:0] | P1[27:0] | P2[27:0] | P3[15:0] | - | P5[19:18] | for P5[19:18] | yes |
| LPC2930 | P0[31:0] | P1[31:0] | P2[27:0] | P3[15:0] | P4[23:0] | P5[19:0] | for P5[19:16] | yes |
| LPC2939 | P0[31:0] | P1[31:0] | P2[27:0] | P3[15:0] | P4[23:0] | P5[19:0] | for P5[19:16] | yes |

## 2. Introduction

The SCU controls some device functionality that is not part of any other block. Settings made in the SCU influence the complete system.

The SCU manages the port selection registers. The function of each I/O pin can be configured. Not all peripherals of the device can be used at the same time, so the desired functions are chosen by selecting a function for each I/O pin.

In addition, memory mapping features and AHB priority settings are controlled by the SCU.

## 3. Register overview

The System Control Unit registers are shown in Table 6–57.

**Table 57. Register overview: SCU (base address: 0xE000 1000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SFSP0_BASE | R/W | 0x000 | Function-select port 0 base address | 0x0000 0000 | Table 6–58 |
| SFSP1_BASE | R/W | 0x100 | Function-select port 1 base address | 0x0000 0000 | Table 6–58 |
| SFSP2_BASE | R/W | 0x200 | Function-select port 2 base address | 0x0000 0000 | Table 6–58 |
| SFSP3_BASE | R/W | 0x300 | Function-select port 3 base address | 0x0000 0000 | Table 6–58 |
| SFSP4_BASE | R/W | 0x400 | Function-select port 4 base address | 0x0000 0000 | Table 6–58 |

**Table 57. Register overview: SCU (base address: 0xE000 1000)** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SFSP5_BASE | R/W | 0x500 | Function-select port 5 base address | 0x0000 0000 | Table 6–58 |
| SFSP5_16 | R/W | 0x540 | Function select port 5 pin 16 (USB port 2, USB_D−2) | 0x0000 0000 | Table 6–60 |
| - | - | 0x544 | reserved | 0x0000 0000 | - |
| SFSP5_18 | R/W | 0x548 | Function select port 5 pin 18 (USB port 2, USB_D−1) | 0x0000 0000 | Table 6–61 |
| - | - | 0x54C | - | 0x0000 0000 | - |
| SEC_DIS | R/W | 0xB00 | Security disable register | | Table 6–62 |
| SEC_STA | R | 0xB04 | Security status register | | Table 6–63 |
| SSMM0 | R/W | 0xC00 | Shadow memory mapping register for ARM | 0x2000 0000 | Table 6–64 |
| SSMM1 | R/W | 0xC04 | Shadow memory mapping register for master DMA0 | 0x2000 0000 | Table 6–64 |
| SSMM2 | R/W | 0xC08 | Shadow memory mapping register for master DMA1 | 0x2000 0000 | Table 6–64 |
| SSMM3 | R/W | 0xC0C | Shadow memory mapping register for master USB | 0x2000 0000 | Table 6–64 |
| SMP0 | R | 0xD00 | Master priority ARM | 0x0000 0000 | Table 6–65 |
| SMP1 | R | 0xD04 | Master priority DMA0 | 0x0000 0000 | Table 6–65 |
| SMP2 | R | 0xD08 | Master priority DMA1 | 0x0000 0000 | Table 6–65 |
| SMP3 | R | 0xD0C | Master priority USB | 0x0000 0000 | Table 6–65 |
| - | R | 0xFF4 | Reserved; do not modify. Read as logic 0 | 0x0000 0000 | - |
| - | R | 0xFFC | Reserved; do not modify. Read as logic 0 | 0xA09B 2000 | - |

## 3.1 SCU port function select registers

The port function select register configures the pin functions individually on the corresponding I/O port. For an overview of pinning, see Section 11–2. Each port pin has its individual register. Each port has its SFSPn_BASE register as defined above in Table 6–57. n runs from 0 to 4, m runs from 0 to 31. For port 5, m runs from 0 to 15.

Table 6–58 shows the address locations of the SFSPn_m registers within a port memory space as indicated by SFSPn_BASE.

**Table 58. SCU port function select register overview (base address: 0xE000 1000 (port 0), 0xE000 1100 (port 1), 0xE000 1200 (port 2), 0xE000 1300 (port3), 0xE000 1400 (port4), 0xE000 1500 (port 5))**

*Ports not pinned out are reserved; do not modify, read as logic 0.*

| Name | Address offset | Access | Reset value | Description | Reference |
|---|---|---|---|---|---|
| SFSPn_0 | 0x00 | R/W | 0x0000 0000 | Function-select port n, pin 0 register | see Table 6–59 |
| SFSPn_1 | 0x04 | R/W | 0x0000 0000 | Function-select port n, pin 1 register | see Table 6–59 |
| SFSPn_2 | 0x08 | R/W | 0x0000 0000 | Function-select port n, pin 2 register | see Table 6–59 |
| SFSPn_3 | 0x0C | R/W | 0x0000 0000 | Function-select port n, pin 3 register | see Table 6–59 |
| SFSPn_4 | 0x10 | R/W | 0x0000 0000 | Function-select port n, pin 4 register | see Table 6–59 |
| SFSPn_5 | 0x14 | R/W | 0x0000 0000 | Function-select port n, pin 5 register | see Table 6–59 |
| SFSPn_6 | 0x18 | R/W | 0x0000 0000 | Function-select port n, pin 6 register | see Table 6–59 |
| SFSPn_7 | 0x1C | R/W | 0x0000 0000 | Function-select port n, pin 7 register | see Table 6–59 |
| SFSPn_8 | 0x20 | R/W | 0x0000 0000 | Function-select port n, pin 8 register | see Table 6–59 |
| SFSPn_9 | 0x24 | R/W | 0x0000 0000 | Function-select port n, pin 9 register | see Table 6–59 |
| SFSPn_10 | 0x28 | R/W | 0x0000 0000 | Function-select port n, pin 10 register | see Table 6–59 |
| SFSPn_11 | 0x2C | R/W | 0x0000 0000 | Function-select port n, pin 11 register | see Table 6–59 |
| SFSPn_12 | 0x30 | R/W | 0x0000 0000 | Function-select port n, pin 12 register | see Table 6–59 |
| SFSPn_13 | 0x34 | R/W | 0x0000 0000 | Function-select port n, pin 13 register | see Table 6–59 |
| SFSPn_14 | 0x38 | R/W | 0x0000 0000 | Function-select port n, pin 14 register | see Table 6–59 |
| SFSPn_15 | 0x3C | R/W | 0x0000 0000 | Function-select port n, pin 15 register | see Table 6–59 |
| SFSPn_16 | 0x40 | R/W | 0x0000 0000 | Function-select port n, pin 16 register | see Table 6–59 |
| SFSPn_17 | 0x44 | R/W | 0x0000 0000 | Function-select port n, pin 17 register | see Table 6–59 |
| SFSPn_18 | 0x48 | R/W | 0x0000 0000 | Function-select port n, pin 18 register | see Table 6–59 |
| SFSPn_19 | 0x4C | R/W | 0x0000 0000 | Function-select port n, pin 19 register | see Table 6–59 |
| SFSPn_20 | 0x50 | R/W | 0x0000 0000 | Function-select port n, pin 20 register | see Table 6–59 |
| SFSPn_21 | 0x54 | R/W | 0x0000 0000 | Function-select port n, pin 21 register | see Table 6–59 |

**Table 58. SCU port function select register overview (base address: 0xE000 1000 (port 0), 0xE000 1100 (port 1), 0xE000 1200 (port 2), 0xE000 1300 (port3), 0xE000 1400 (port4), 0xE000 1500 (port 5))** *…continued*

*Ports not pinned out are reserved; do not modify, read as logic 0.*

| Name | Address offset | Access | Reset value | Description | Reference |
|---|---|---|---|---|---|
| SFSPn_22 | 0x58 | R/W | 0x0000 0000 | Function-select port n, pin 22 register | see Table 6–59 |
| SFSPn_23 | 0x5C | R/W | 0x0000 0000 | Function-select port n, pin 23 register | see Table 6–59 |
| SFSPn_24 | 0x60 | R/W | 0x0000 0000 | Function-select port n, pin 24 register | see Table 6–59 |
| SFSPn_25 | 0x64 | R/W | 0x0000 0000 | Function-select port n, pin 25 register | see Table 6–59 |
| SFSPn_26 | 0x68 | R/W | 0x0000 0000 | Function-select port n, pin 26 register | see Table 6–59 |
| SFSPn_27 | 0x6C | R/W | 0x0000 0000 | Function-select port n, pin 27 register | see Table 6–59 |
| SFSPn_28 | 0x70 | R/W | 0x0000 0000 | Function-select port n, pin 28 register | see Table 6–59 |
| SFSPn_29 | 0x74 | R/W | 0x0000 0000 | Function-select port n, pin 29 register | see Table 6–59 |
| SFSPn_30 | 0x78 | R/W | 0x0000 0000 | Function-select port n, pin 30 register | see Table 6–59 |
| SFSPn_31 | 0x7C | R/W | 0x0000 0000 | Function-select port n, pin 31 register | see Table 6–59 |

Table 6–59 shows the bit assignment of the SFSPn_m registers (n runs from 0 to 4, m runs from 0 to 31. For port 5, m runs from 0 to 15).

**Remark:** Note that on Reset the ADC pins P0[23] to P0[8] are set to digital inputs without internal pull-up/down on reset. This guarantees that these pins are 5 V tolerant after reset, even though the analog inputs to ADC1 and ADC2 are not. The default pad type is analog input for all other port pins (except P5[19:16]).

**Table 59. SFSPn_m register bit description (base address: 0xE000 1000 (port 0), 0xE000 1100 (port 1), 0xE000 1200 (port 2), 0xE000 1300 (port3), 0xE000 1400 (port4), 0xE000 1500 (port 5))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved. Read as logic 0 |
| 4 to 2 | PAD_TYPE[1] | R/W | | Input pad type |
| | | | 000*[2] | Analog input[3] |
| | | | 001 | Digital input without internal pull up/down |
| | | | 010 | Not allowed |
| | | | 011 | Digital input with internal pull up[4] |
| | | | 100 | Not allowed |
| | | | 101 | Digital input with internal pull down |
| | | | 110 | Not allowed |
| | | | 111 | Digital input with bus keeper |

Table 59. SFSPn_m register bit description (base address: 0xE000 1000 (port 0), 0xE000 1100 (port 1), 0xE000 1200 (port 2), 0xE000 1300 (port3), 0xE000 1400 (port4), 0xE000 1500 (port 5)) *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 to 0 | FUNC_SEL[1:0] | R/W | | Function-select; for the function-to-port-pin mapping tables[5] |
| | | | 00* | Select pin function 0 |
| | | | 01 | Select pin function 1 |
| | | | 10 | Select pin function 2 |
| | | | 11 | Select pin function 3 |

[1] These bits control the input section of the I/O buffer. The FUNC_SEL bits will define if a pin is input or output depending on the function selected. For GPIO mode the direction is controlled by the direction register, see Table 16–201. Note that for functions of type input, the input pad type must be set correctly in addition to the FUNC_SEL bits.

[2] The reset value for port pins P0[23:8] is 001 (digital input without internal pull-up/down). This guarantees that the ADC pins are 5 V tolerant after reset even though the analog pad of ADC1 and ADC2 is not 5 V tolerant.

[3] The ‘analog’ connection towards the ADC is always enabled. Use PAD_TYPE = 000 when used as analog input to avoid the input buffer oscillating on slow analog-signal transitions or noise. The digital input buffer is switched off.

[4] When pull-up is activated the input is **not** 5 V -tolerant.

[5] Each pin has up to four functions.

Setting the FUNC_SEL bits in the SFSP5_16 register also determines the function of port 5[17]. If the USB_D–2 function is selected for P5[16], P5[17] is automatically assigned to the USB_D+2 function. If P5[16] is GPIO, P5[17] is assigned to GPIO as well.

Table 60. SFSP5_16 function select register bit description (SFSP5_16, address 0xE000 1540)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 2 | reserved | R | - | Reserved. Read as logic 0 |
| 1 to 0 | FUNC_SEL[1:0] | R/W | | Function-select; for the function-to-port-pin mapping tables |
| | | | 00* | Select pin function GPIO on P5[16] |
| | | | 01 | Select pin USB_D–2 |
| | | | 10 | reserved |
| | | | 11 | reserved |

Setting the FUNC_SEL bits in the SFSP5_18 register also determines the function of port 5[19]. If the USB_D–1 function is selected for P5[18], P5[19] is automatically assigned to the USB_D+1 function. If P5[18] is selected GPIO, P5[19] is assigned to GPIO as well.

**Table 61. SFSP5_18 function select register bit description (SFSP_5_18, address 0xE000 1548)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31:5 | - | - | - | reserved |
| 4 | VBUS | R/W | | USB mode |
| | | | 0 | port 1 in host or device mode |
| | | | 1 | port 1 in OTG mode |
| 3 to 2 | - | - | - | reserved |
| 1 to 0 | FUNC_SEL[1:0] | R/W | | Function-select; for the function-to-port-pin mapping tables |
| | | | 00* | Select pin function GPIO on P5[18] |
| | | | 01 | Select pin USB_D–1 |
| | | | 10 | reserved |
| | | | 11 | reserved |

### 3.1.1 Functional description

The digital I/O pins of the device are divided into four ports. For each pin of these ports one out of four functions can be chosen. Refer to Figure 6–15 for a schematic representation of an I/O-pin. The I/O functionality is dependent on the application.

The function of an I/O can be changed ‘on the fly’ during run-time. By default it is assigned to function 0, which is the GPIO. For each pin of these ports a programmable pull-up and pull-down resistor (R) is present.

**Remark:** Even though the default function is GPIO, the pad type has to be set to digital in the SFSPn_m registers in order to use the GPIO functionality (see Table 6–59).

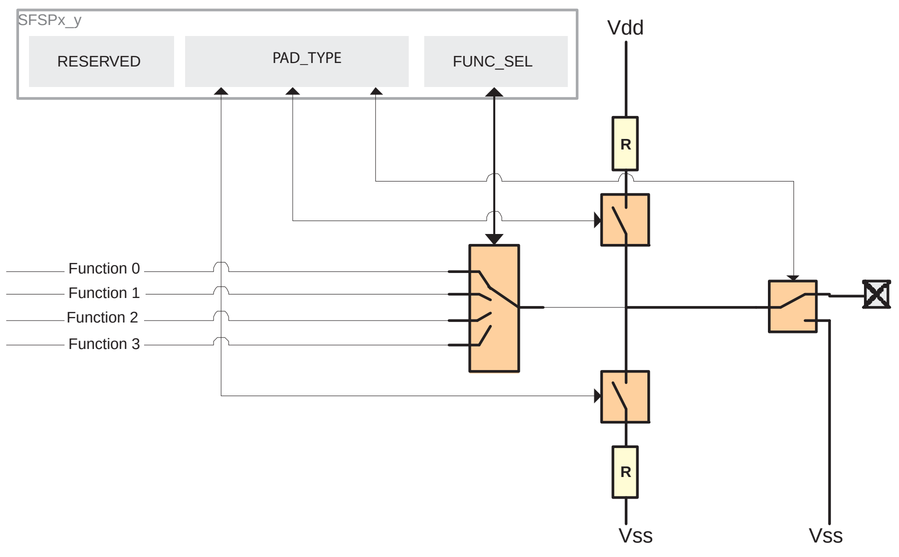


**Fig 15. Schematic representation of an I/O pin**

## 3.2 JTAG security registers

**Table 62. Security disable register bit description (SEC_DIS, address 0xE000 1B00)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31:2 | - | - | - | reserved |
| 1 | DIS | R/W | | JTAG security enable/disable |
| | | | 1 | Disables JTAG security and clears bit 1 on SEC_STA |
| | | | 0 | enables JTAG security |
| 0 | - | - | - | reserved |

**Table 63. Security status register bit description (SEC_STA, address 0xE000 1B04)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31:2 | - | - | - | reserved |
| 1 | DIS | R | | JTAG security status |
| | | | 1 | JTAG security enabled |
| | | | 0 | JTAG security disabled |
| 0 | - | - | - | reserved |

## 3.3 Shadow memory mapping registers

The shadow memory mapping register defines which part of the memory region is present in the shadow memory area. The shadow memory mapping start address is the pointer within a region indicating the shadowing to the shadow area starting at location 0000 0000h. In this way a whole region or only a part of the flash, SRAM or external memory bank can be remapped to the shadow area.

The SSMM0 register defines the memory mapping seen by the ARM CPU master, the SSMM1 and SSMM2 register defines the memory mapping for the DMA0 and DMA1 masters, and the SSMM3 register for the USB master.

**Table 64. SSMMx register bit description (SSMM0/1/2/3, addresses: 0xE000 1C00, 0xE000 1C04, 0xE000 1C08, 0xE000 1C0C)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | SMMSA[21:0] | R/W | 0x2000 0000* | shadow memory map start address; memory start address for mapping (a part of) a region to the shadow area; the start address is aligned on 1 kB boundaries and therefore the lowest 10 bits must be always logic 0 |
| 9 to 0 | reserved | - | - | reserved; do not modify, read as logic 0, write<br>as logic 0 |

## 3.4 AHB master priority registers

By default, AHB access is scheduled round-robin. However, the AHB access priority of each of the AHB bus masters can be set by writing the priority integer value (highest priority = 1, lowest priority = 4) to the master’s priority register SMPn.

All masters with the same priority are scheduled on a round-robin basis.

**Table 65. SMPx register bit description (SMP0/1/2/3, addresses: 0xE000 1D00 (ARM), 0xE000 1D04 (DMA0), 0xE000 1D08 (DMA1), 0xE000 1D0C (USB))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31:3 | - | - | - | reserved |
| 2:0 | PRIO | R/W | 0x0 | AHB priority (1: highest, 4: lowest) |

# UM10316

## Chapter 7: LPC29xx Chip Feature ID (CFID)

**Rev. 3 — 19 October 2010** **User manual**

## 1. Introduction

The CFID module contains registers that show the functionality of the chip. It contains an ID to identify the silicon and four registers containing information about the features enabled or disabled for each part in the LPC29xx series.

## 2. Register overview

**Table 66. Register overview: CFID (base address 0xE000 0000)**

| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| CHIPID | R | 0x000 | chip ID | 0x209C E02B |
| FEAT0 | R | 0x100 | package information register | part dependent (see Table 7–67) |
| FEAT1 | R | 0x104 | chip configuration register 1 | |
| FEAT2 | R | 0x108 | chip feature configuration register 2 | |
| FEAT3 | R | 0x10C | chip configuration register 3 | |

**Table 67. CFID reset values**

| Part | CHIPID | FEAT0 | FEAT1 | FEAT2 | FEAT3[1] |
|---|---|---|---|---|---|
| LPC2917/01 | 0x209C E02B | 0xFFFF FFF4 | 0xE6E6 FF3F | 0xDF07 FF08 | 0xFFFF FCF0 |
| LPC2919/01 | 0x209C E02B | 0xFFFF FFF4 | 0xE6E6 FF3F | 0xDF0B FF08 | 0xFFFF FCF8 |
| LPC2921 | 0x209C E02B | 0xFFFF FFF2 | 0xE5E5 FF03 | 0xDF01 FF08 | 0xFFFF FA84 |
| LPC2923 | 0x209C E02B | 0xFFFF FFF2 | 0xE5E5 FF03 | 0xDF03 FF08 | 0xFFFF FA88 |
| LPC2925 | 0x209C E02B | 0xFFFF FFF2 | 0xE5E5 FF0F | 0xDF07 FF08 | 0xFFFF FAD0 |
| LPC2926 | 0x209C E02B | 0xFFFF FFF4 | 0xE6E6 FF3F | 0xDF07 FF08 | 0xFFFF FFF1 |
| LPC2927 | 0x209C E02B | 0xFFFF FFF4 | 0xE6E6 FF3F | 0xDF07 FF08 | 0xFFFF FFF1 |
| LPC2929 | 0x209C E02B | 0xFFFF FFF4 | 0xE6E6 FF3F | 0xDF0B FF08 | 0xFFFF FFF9 |
| LPC2930 | 0x209C E02B | 0xFFFF FFF5 | 0xE6E6 FF3F | 0xDF00 FF00 | 0xFFFF FFE3 |
| LPC2939 | 0x209C E02B | 0xFFFF FFF5 | 0xE6E6 FF3F | 0xDF0B FF08 | 0xFFFF FFF9 |

[1] Factory setting. The reset value of the FEAT3 register depends on the setting of the JTAG security bit (see Table 7–68).

Registers CHIPID and FEAT0 to FEAT2 are factory preprogrammed and read-only. The FEAT3 register contains the value of the JTAG security bit (see Table 7–68). The JTAG security bit can be defined by the user by writing to the index sector (see Section 28–2.6.3). The factory default setting allows JTAG access.

**Table 68. FEAT3 register bit description (FEAT3, address 0xE000 010C)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | JTAGSEC | R | | The setting of this bit is determined by the setting of the JTAG security in the flash index sector (see Section 28–2.6.3). |
| | | | 1 | JTAG security disabled. JTAG access allowed. |
| | | | 0 | JTAG security enabled. No JTAG access. |
| 30:0 | - | R | see Table 7–67 | reserved |

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. Not all event sources are connected to pins. Table 8–69 shows the event router connections for each LPC29xx part.

Table 69. External event router connections

| Part | External interrupt pins | USB | CAN/LIN (n = 0,1) | I2C (n = 0,1) | SPI (m = 0,1, 2) | UART (n = 0,1) |
|---|---|---|---|---|---|---|
| LPC2921/23/25 | EI[0:3] | - | RXDCn; RXDLn | SCLn | SDIm | RXDn |
| LPC2917/19/01 | EI[0:7] | - | RXDCn; RXDLn | SCLn | SDIm | RXDn |
| LPC2926/27/29 | EI[0:7] | USB_SCL1 | RXDCn; RXDLn | SCLn | SDIm | RXDn |
| LPC2930 | EI[0:7] | USB_SCL1 | RXDCn; RXDLn | SCLn | SDIm | RXDn |
| LPC2939 | EI[0:7] | USB_SCL1 | RXDCn; RXDLn | SCLn | SDIm | RXDn |

## 2. Event router functional description

The Event Router provides bus-controlled routing of input events to the VIC for use as interrupt or wake-up signals to the CGU. Event inputs are connected to internal peripherals and to external interrupt pins. All event inputs are described in Table 8–70.

The CAN and LIN receive-pin events can be used as extra external interrupt pins when CAN and/or LIN functionality is not needed.

A schematic representation of the Event Router is shown in Figure 8–16.

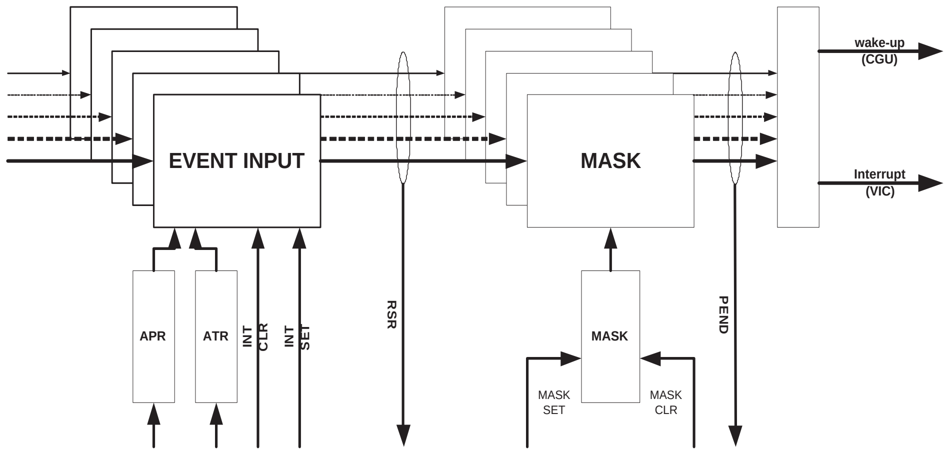


Fig 16. Schematic representation of the Event Router

Input events are processed in event slices; one for each event signal. Each of these slices generates one event signal and is visible in the RSR (Raw Status Register). These events are then AND-ed with enables from the MASK register to give PEND (PENDing register) event status. If one or more events are pending the output signals are active.

An event input slice is controlled through bits in the APR (Activation Polarity Register), the ATR (Activation Type Register), INT_SET (INTerrupt SET) and INT_CLR (INTerrupt CLeaR).

- The polarity setting (APR) conditionally inverts the interrupt input event.
- The activation type setting (ATR) selects between latched/edge or direct/level event.
- The resulting interrupt event is visible through a read-action in the RSR.
- The RSR is AND-ed with the MASK register and the result is visible in the PEND register.
- The wake-up (CGU) and interrupt (VIC) outputs are active if one of the events is pending.

## 2.1 Event router pin connections

The event router module in the LPC29xx is connected to the pins listed below. The pins are combined with other functions on the port pins of the LPC29xx. Table 8–70 shows the pins connected to the event router, and also the corresponding bit position in the event-router registers and the default polarity. Not all pin connections are available on all parts. See Table 8–69 for available pin connections depending on part number.

**Table 70. Event-router pin connections**

| Symbol | Direction | Bit position | Description | Default polarity |
|---|---|---|---|---|
| EXTINT0 | IN | 0 | external interrupt input 0 | 1 |
| EXTINT1 | IN | 1 | external interrupt input 1 | 1 |
| EXTINT2 | IN | 2 | external interrupt input 2 | 1 |
| EXTINT3 | IN | 3 | external interrupt input 3 | 1 |
| EXTINT4 | IN | 4 | external interrupt input 4 | 1 |
| EXTINT5 | IN | 5 | external interrupt input 5 | 1 |
| EXTINT6 | IN | 6 | external interrupt input 6 | 1 |
| EXTINT7 | IN | 7 | external interrupt input 7 | 1 |
| CAN0 RXDC | IN | 8 | CAN0 receive data input wake-up | 0 |
| CAN1 RXDC | IN | 9 | CAN1 receive data input wake-up | 0 |
| I2C0_SCL | IN | 10 | I2C0 SCL clock input | 0 |
| I2C1_SCL | IN | 11 | I2C1 SCL clock input | 0 |
| - | | 13-12 | reserved | - |
| LIN0 RXDL | IN | 14 | LIN0 receive data input wake-up | 0 |
| LIN1 RXDL | IN | 15 | LIN1 receive data input wake-up | 0 |
| SPI0 SDI | IN | 16 | SPI0 data in | 0 |
| SPI1 SDI | IN | 17 | SPI1 data in | 0 |
| SPI2 SDI | IN | 18 | SPI2 data in | 0 |
| UART0 RXD | IN | 19 | UART0 receive data input | 0 |

**Table 70. Event-router pin connections** *…continued*

| Symbol | Direction | Bit position | Description | Default polarity |
|---|---|---|---|---|
| UART1 RXD | IN | 20 | UART1 receive data input | 0 |
| USB_I2C_SCL | IN | 21 | USB I$^2$C serial clock | 0 |
| - | na | 22 | CAN interrupt (internal) | 1 |
| - | na | 23 | VIC FIQ (internal) | 1 |
| - | na | 24 | VIC IRQ (internal) | 1 |
| - | - | 26 to 25 | reserved | - |

## 3. Register overview

**Table 71. Register overview: event router (base address: 0xE000 2000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| PEND | R | 0xC00 | Event status register | 0x0000 0000 | see Table 8–72 |
| INT_CLR | W | 0xC20 | Event-status clear register | - | see Table 8–73 |
| INT_SET | W | 0xC40 | Event-status set register | - | see Table 8–74 |
| MASK | R | 0xC60 | Event-enable register | 0x07FF FFFF | see Table 8–75 |
| MASK_CLR | W | 0xC80 | Event-enable clear register | - | see Table 8–76 |
| MASK_SET | W | 0xCA0 | Event-enable set register | - | see Table 8–77 |
| APR | R/W | 0xCC0 | Activation polarity register | 0x01C0 00FF | see Table 8–78 |
| ATR | R/W | 0xCE0 | Activation type register | 0x07FF FFFF | see Table 8–79 |
| reserved | R | 0xD00 | Reserved; do not modify | - | - |
| RSR | R/W | 0xD20 | Raw-status register | 0x0000 0000 | see Table 8–80 |

### 3.1 Event status register

The event status register determines when the Event Router forwards an interrupt request to the Vectored Interrupt Controller, if the corresponding event enable has been set.

Table 8–72 shows the bit assignment of the PEND register.

**Table 72. PEND register bit description (address 0xE000 2C00)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | PEND[26] | R | 1 | An event has occurred on a corresponding pin, or logic 1 is written to bit 26 in the INT_SET register |
| | | | 0* | No event is pending or logic 1 has been written to bit 26 in the INT_CLR register |

**Table 72. PEND register bit description (address 0xE000 2C00)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| : | : | : | : | : |
| 0 | PEND[0] | R | 1 | An event has occurred on a corresponding pin or logic 1 is written to bit 0 in the INT_SET register |
| | | | 0* | No event is pending or logic 1 has been written to bit 0 in the INT_CLR register |

## 3.2 Event-status clear register

The event-status clear register clears the bits in the event status register.

Table 8–73 shows the bit assignment of the INT_CLR register.

**Table 73. INT_CLR register bit description (address 0xE000 2C20)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | INT_CLR[26] | W | 1 | Bit 26 in the event status register is cleared |
| | | | 0 | Bit 26 in the event status register is unchanged |
| : | : | : | : | : |
| 0 | INT_CLR[0] | W | 1 | Bit 0 in the event status register is cleared |
| | | | 0 | Bit 0 in the event status register is unchanged |

## 3.3 Event-status set register

The event-status set register sets the bits in the event status register.

Table 8–74 shows the bit assignment of the INT_SET register.

**Table 74. INT_SET register bit description (address 0xE000 2C40)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | INT_SET[26] | W | 1 | Bit 26 in the event status register is set |
| | | | 0 | Bit 26 in the event status register is unchanged |
| : | : | : | : | : |
| 0 | INT_SET[0] | W | 1 | Bit 0 in the event status register is set |
| | | | 0 | Bit 0 in the event status register is unchanged |

## 3.4 Event enable register

The event enable register determines when the Event Router sets the event status and forwards this to the VIC if the corresponding event-enable has been set.

Table 8–75 shows the bit assignment of the MASK register.

**Table 75. MASK register bit description (address 0xE000 2C60)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | MASK[26] | R | | Event enable<br>This bit is set by writing a logic 1 to bit 26 in the MASK_SET register<br>This bit is cleared by writing a logic 1 to bit 26 in the MASK_CLR register |
| | | | 1* | |
| : | : | : | : | : |
| 0 | MASK[0] | R | | Event enable<br>This bit is set by writing a logic 1 to bit 0 in the MASK_SET register<br>This bit is cleared by writing a logic 1 to bit 0 in the MASK_CLR register |
| | | | 1* | |

## 3.5 Event-enable clear register

The event-enable clear register clears the bits in the event enable register.

Table 8–76 shows the bit assignment of the MASK_CLR register.

**Table 76. MASK_CLR register bit description (address 0xE000 2C80)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | MASK_CLR[26] | W | 1 | Bit 26 in the event enable register is cleared |
| | | | 0 | Bit 26 in the event enable register is unchanged |
| : | : | : | : | : |
| 0 | MASK_CLR[0] | W | 1 | Bit 0 in the event enable register is cleared |
| | | | 0 | Bit 0 in the event enable register is unchanged |

## 3.6 Event-enable set register

The event-enable set register sets the bits in the event enable register.

Table 8–77 shows the bit assignment of the MASK_SET register.

**Table 77. MASK_SET register bit description (address 0xE000 2CA0)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | MASK_SET[26] | W | 1 | Bit 26 in the event-enable register is set |
| | | | 0 | Bit 26 in the event-enable register is unchanged |
| : | : | : | : | : |
| 0 | MASK_SET[0] | W | 1 | Bit 0 in the event enable register is set |
| | | | 0 | Bit 0 in the event enable register is unchanged |

## 3.7 Activation polarity register

The APR is used to configure which level is the active state for the event source.

Table 8–78 shows the bit assignment of the APR register.

Table 78. APR register bit description (address 0xE000 2CC0)

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | APR[26] | R/W | 1[1] | The corresponding event is HIGH sensitive (HIGH-level or rising edge) |
| | | | 0[1] | The corresponding event is LOW sensitive (LOW-level or falling edge) |
| : | : | : | : | : |
| 0 | APR[0] | R/W | 1[1] | The corresponding event is HIGH sensitive (HIGH-level or rising edge) |
| | | | 0[1] | The corresponding event is LOW sensitive (LOW-level or falling edge) |

[1] Reset value is logic 1 for APR[24:22] and APR[7:0]; reset value is logic 0 for APR[26:25] and APR[21:8].

## 3.8 Activation type register

The ATR is used to configure whether an event is used directly or is latched. If the event is latched the interrupt persists after its source has become inactive until it is cleared by an interrupt-clear write action. The Event Router includes an edge-detection circuit which prevents re-assertion of an event interrupt if the input remains at active level after the latch is cleared. Level-sensitive events are expected to be held and removed by the event source.

Table 8–79 shows the bit assignment of the ATR register.

Table 79. ATR register bit description (address 0xE000 2CE0)
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | ATR[24] | R/W | 1* | Corresponding event is latched (edge-sensitive) |
| | | | 0 | Corresponding event is directly forwarded (level- sensitive) |
| : | : | : | : | : |
| 0 | ATR[0] | R/W | 1* | Corresponding event is latched (edge-sensitive) |
| | | | 0 | Corresponding event is directly forwarded (level-sensitive) |

## 3.9 Raw status register

The RSR shows unmasked events including latched events. Level-sensitive events are removed by the event source: edge-sensitive events need to be cleared via the event-clear register.

Table 8–80 shows the bit assignment of the RSR register.

Table 80. RSR register bit description (address 0xE000 2D20)

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 27 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 26 | RSR[26] | R | 1 | Corresponding event has occurred |
| | | | 0* | Corresponding event has not occurred |
| : | : | : | : | : |
| 0 | RSR[0] | R | 1 | Corresponding event has occurred |
| | | | 0* | Corresponding event has not occurred |

# UM10316

# Chapter 9: LPC29xx Vectored Interrupt Controller (VIC)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. See Table 9–81 for interrupt requests that are configuration dependent. All other interrupt requests are available in all LPC29xx parts (see Table 9–88).

**Table 81. Available interrupt requests**

| Part | USB interrupts | ADC interrupts | Flash interrupts |
|---|---|---|---|
| LPC2921/23/25 | 46 to 48, 50 | 17, 18 (ADC1/2) | 11 |
| LPC2917/19/01 | 45 to 51 | 17, 18 (ADC1/2) | 11 |
| LPC2926/27/29 | 45 to 51 | 16, 17, 18 (ADC0/1/2) | 11 |
| LPC2930 | 45 to 51 | 16, 17, 18 (ADC0/1/2) | n/a |
| LPC2939 | 45 to 51 | 16, 17, 18 (ADC0/1/2) | 11 |

## 2. VIC functional description

The VIC is a very flexible and powerful block for interrupting the ARM processor on request. The VIC routes incoming interrupt requests from multiple source to the ARM processor core. Figure 9–17 shows the VIC connections. An interrupt target is configured for each interrupt request input of the controller, and the various device peripherals are connected to the interrupt request inputs. An extensive list of inputs can be found in Table 9–88.

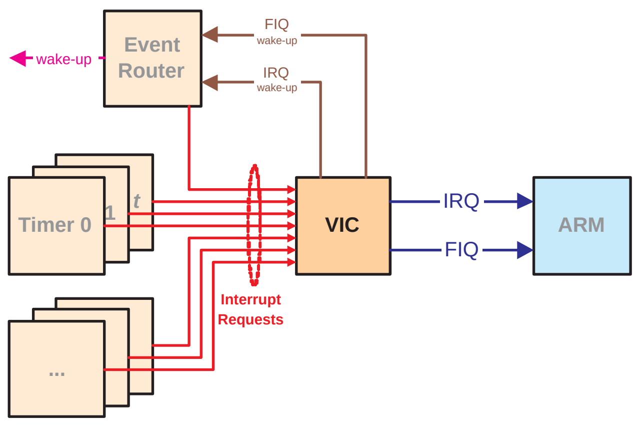


**Fig 17. Schematic representation of the VIC connections**

The ARM core has two possible interrupt targets: IRQ and FIQ.

- The FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register-saving in service routines. This minimizes the overhead of context switching. FIQ should not enable interrupt during execution: if needed an IRQ should be used for this purpose.

- The IRQ exception has a lower priority than FIQ and is masked out when an FIQ exception occurs. IRQ service routines should take care of saving and/or restoring the used registers themselves.

The VIC also provides IRQ and FIQ wake-up events to the Event Router. This enables the system to wake up upon an interrupt. See also Section 10–5 for interrupt and wake-up structure.

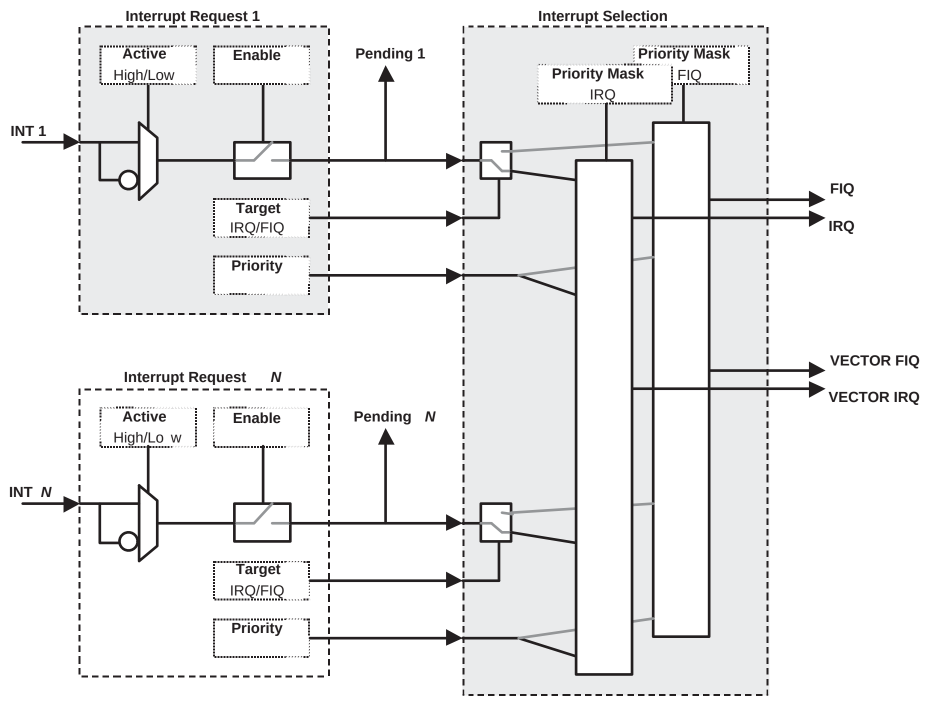


**Fig 18. Schematic representation of the VIC**

A representation of the VIC is shown in Figure 9–18. Each interrupt request has its own configuration:

- Polarity (active HIGH or LOW): The interrupt request inputs are level-sensitive. The activation level can be programmed according to the connected peripheral (see Table 8–70 for the recommended setting).
- Target (IRQ/FIQ): Two targets are possible within the ARM architecture:
  - IRQ, Interrupt request; This target is referred to as TARGET1
  - FIQ, Fast Interrupt request; This target is referred to as TARGET0
- Priority of the pending interrupt is compared with the priority mask of the selected target.
  - The interrupt is masked if the priority value of the pending interrupt is equal to or lower than the value in the priority mask.
  - For each interrupt target, pending interrupt requests with priority above the priority threshold are combined through a logical OR, and the result is then routed towards the interrupt target.

If the level-sensitive interrupt request line of the VIC is enabled (depending on the polarity setting), the request is forwarded to the interrupt selection. The interrupt selection part selects the interrupt request line with the highest priority, based on the target and priority of the interrupt request and priority masks.

The VIC introduces an interrupt latency (measured from assertion of an INT_N signal to an assertion of IRQ/FIQ) of less than two periods of the system clock.

The INT_VECTOR register can be used to identify the interrupt request line that needs to be served. It can be used as an interrupt vector to the interrupt service routine. In TABLE_ADDR the offset of the vector table can be programmed. Together with the INDEX this information forms a vector.

The IRQ or FIQ generates a corresponding exception on the ARM core. The exception handler should read the INT_VECTOR register to determine the highest-priority interrupt source. This functionality should be implemented in a dispatcher, usually in the assembler. This dispatcher performs the following steps:

### 2.1 Non-nested interrupt service routine

1. Put all registers that are used (according to the ARM-Procedure-Call Standard) on stack.
2. Determine the interrupt source by reading The INT_VECTOR register
3. Call the interrupt service routine
4. Get all (saved) registers back from the stack
5. End the interrupt service routine by restoring the Program Counter register (PC).

### 2.2 Nested interrupt service routine

1. Put all registers that are used (according to the ARM-Procedure-Call Standard) on stack.
2. Determine the interrupt source by reading The INT_VECTOR register
3. Raise the priority-masking threshold to the priority level of the interrupt request to be served
4. Re-enable interrupt in the processor
5. Call the interrupt service routine
6. Restore the saved priority mask
7. Get all (saved) registers back from the stack
8. End the interrupt service routine by restoring the program counter.

## 3. VIC programming example

The VIC driver provides an API to set up an interrupt source with all its parameters. All this information ends up in the INT_REQUEST register of the VIC.

In most cases interrupt handling is controlled by some kind of OS. Installation of interrupt vector tables depends on this.

## 4. Register overview

Table 82. Register overview: Vectored Interrupt Controller (base address: 0xFFFF F000)

| Name | Access | Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| INT_PRIORITYMASK_0 | R/W | 0x000 | Target 0 priority-mask register | - | see Table 9–83 |
| INT_PRIORITYMASK_1 | R/W | 0x004 | Target 1 priority-mask register | - | see Table 9–83 |
| INT_VECTOR_0 | R/W | 0x100 | Target 0 vector register | - | see Table 9–84 |
| INT_VECTOR_1 | R/W | 0x104 | Target 1 vector register | - | see Table 9–84 |
| INT_ PENDING_1_31 | R | 0x200 | Interrupt-pending status register | - | see Table 9–85 |
| INT_ PENDING_32_63 | R | 0x204 | Interrupt-pending status register | - | see Table 9–86 |
| INT_FEATURES | R | 0x300 | Interrupt controller features register | 0x0001 0F3F | see Table 9–87 |
| INT_REQUEST_1 | R/W | 0x404 | Interrupt Request 1 control register | - | see Table 9–89 |
| INT_REQUEST_2 | R/W | 0x408 | Interrupt Request 2 control register | - | see Table 9–89 |
| INT_REQUEST_3 | R/W | 0x40C | Interrupt Request 3 control register | - | see Table 9–89 |
| INT_REQUEST_4 | R/W | 0x410 | Interrupt Request 4 control register | - | see Table 9–89 |
| INT_REQUEST_5 | R/W | 0x414 | Interrupt Request 5 control register | - | see Table 9–89 |
| INT_REQUEST_6 | R/W | 0x418 | Interrupt Request 6 control register | - | see Table 9–89 |
| INT_REQUEST_7 | R/W | 0x41C | Interrupt Request 7 control register | - | see Table 9–89 |
| INT_REQUEST_8 | R/W | 0x420 | Interrupt Request 8 control register | - | see Table 9–89 |
| INT_REQUEST_9 | R/W | 0x424 | Interrupt Request 9 control register | - | see Table 9–89 |
| INT_REQUEST_10 | R/W | 0x428 | Interrupt Request 10 control register | - | see Table 9–89 |
| INT_REQUEST_11 | R/W | 0x42C | Interrupt Request 11 control register | - | see Table 9–89 |
| INT_REQUEST_12 | R/W | 0x430 | Interrupt Request 12 control register | - | see Table 9–89 |
| INT_REQUEST_13 | R/W | 0x434 | Interrupt Request 13 control register | - | see Table 9–89 |
| INT_REQUEST_14 | R/W | 0x438 | Interrupt Request 14 control register | - | see Table 9–89 |
| INT_REQUEST_15 | R/W | 0x43C | Interrupt Request 15 control register | - | see Table 9–89 |

**Table 82. Register overview: Vectored Interrupt Controller (base address: 0xFFFF F000)** *…continued*

| Name | Access | Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| INT_REQUEST_16 | R/W | 0x440 | Interrupt Request 16 control register | - | see Table 9–89 |
| INT_REQUEST_17 | R/W | 0x444 | Interrupt Request 17 control register | - | see Table 9–89 |
| INT_REQUEST_18 | R/W | 0x448 | Interrupt Request 18 control register | - | see Table 9–89 |
| INT_REQUEST_19 | R/W | 0x44C | Interrupt Request 19 control register | - | see Table 9–89 |
| INT_REQUEST_20 | R/W | 0x450 | Interrupt Request 20 control register | - | see Table 9–89 |
| INT_REQUEST_21 | R/W | 0x454 | Interrupt Request 21 control register | - | see Table 9–89 |
| INT_REQUEST_22 | R/W | 0x458 | Interrupt Request 22 control register | - | see Table 9–89 |
| INT_REQUEST_23 | R/W | 0x45C | Interrupt Request 23 control register | - | see Table 9–89 |
| INT_REQUEST_24 | R/W | 0x460 | Interrupt Request 24 control register | - | see Table 9–89 |
| INT_REQUEST_25 | R/W | 0x464 | Interrupt Request 25 control register | - | see Table 9–89 |
| INT_REQUEST_26 | R/W | 0x468 | Interrupt Request 26 control register | - | see Table 9–89 |
| INT_REQUEST_27 | R/W | 0x46C | Interrupt Request 27 control register | - | see Table 9–89 |
| INT_REQUEST_28 | R/W | 0x470 | Interrupt Request 28 control register | - | see Table 9–89 |
| INT_REQUEST_29 | R/W | 0x474 | Interrupt Request 29 control register | - | see Table 9–89 |
| INT_REQUEST_30 | R/W | 0x478 | Interrupt Request 30 control register | - | see Table 9–89 |
| INT_REQUEST_31 | R/W | 0x47C | Interrupt Request 31 control register | - | see Table 9–89 |
| INT_REQUEST_32 | R/W | 0x480 | Interrupt Request 32 control register | - | see Table 9–89 |
| INT_REQUEST_33 | R/W | 0x484 | Interrupt Request 33 control register | - | see Table 9–89 |
| INT_REQUEST_34 | R/W | 0x488 | Interrupt Request 34 control register | - | see Table 9–89 |
| INT_REQUEST_35 | R/W | 0x48C | Interrupt Request 35 control register | - | see Table 9–89 |
| INT_REQUEST_36 | R/W | 0x490 | Interrupt Request 36 control register | - | see Table 9–89 |
| INT_REQUEST_37 | R/W | 0x494 | Interrupt Request 37 control register | - | see Table 9–89 |
| INT_REQUEST_38 | R/W | 0x498 | Interrupt Request 38 control register | - | see Table 9–89 |

**Table 82. Register overview: Vectored Interrupt Controller (base address: 0xFFFF F000)** *…continued*

| Name | Access | Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| INT_REQUEST_39 | R/W | 0x49C | Interrupt Request 39 control register | - | see Table 9–89 |
| INT_REQUEST_40 | R/W | 0x4A0 | Interrupt Request 40 control register | - | see Table 9–89 |
| INT_REQUEST_41 | R/W | 0x4A4 | Interrupt Request 41 control register | - | see Table 9–89 |
| INT_REQUEST_42 | R/W | 0x4A8 | Interrupt Request 42 control register | - | see Table 9–89 |
| INT_REQUEST_43 | R/W | 0x4AC | Interrupt Request 43 control register | - | see Table 9–89 |
| INT_REQUEST_44 | R/W | 0x4B0 | Interrupt Request 44 control register | - | see Table 9–89 |
| INT_REQUEST_45 | R/W | 0x4B4 | Interrupt Request 45 control register | - | see Table 9–89 |
| INT_REQUEST_46 | R/W | 0x4B8 | Interrupt Request 46 control register | - | see Table 9–89 |
| INT_REQUEST_47 | R/W | 0x4BC | Interrupt Request 47 control register | - | see Table 9–89 |
| INT_REQUEST_48 | R/W | 0x4C0 | Interrupt Request 48 control register | - | see Table 9–89 |
| INT_REQUEST_49 | R/W | 0x4C4 | Interrupt Request 49 control register | - | see Table 9–89 |
| INT_REQUEST_50 | R/W | 0x4C8 | Interrupt Request 50 control register | - | see Table 9–89 |
| INT_REQUEST_51 | R/W | 0x4CC | Interrupt Request 51 control register | - | see Table 9–89 |
| INT_REQUEST_52 | R/W | 0x4D0 | Interrupt Request 52 control register | - | see Table 9–89 |
| INT_REQUEST_53 | R/W | 0x4D4 | Interrupt Request 53 control register | - | see Table 9–89 |
| INT_REQUEST_54 | R/W | 0x4D8 | Interrupt Request 54 control register | - | see Table 9–89 |
| INT_REQUEST_55 | R/W | 0x4DC | Interrupt Request 55 control register | - | see Table 9–89 |
| INT_REQUEST_56 | R/W | 0x4E0 | Interrupt Request 55 control register | - | see Table 9–89 |

## 4.1 Interrupt priority mask register

The interrupt priority-mask registers define the thresholds for priority-level masking. Each interrupt target has its own priority limiter which can be used to define the minimum priority level for nesting interrupts. Typically, the priority limiter is set to the priority level of the interrupt service routine that is currently being executed so that only interrupt requests at a higher priority level lead to a nested interrupt service. Nesting can be disabled by setting the priority level to Fh in the interrupt request register.

Table 9–83 shows the bit assignment of the INT_PRIORITYMASK_0 and INT_PRIORITYMASK_1 registers.

Table 83. INT_PRIORITYMASK_n registers bit description (INT_PRIORITYMASK_0/1, addresses 0xFFFF F000 and 0xFFFF F004)

| Bit | Symbol | Access | Reset value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 3 to 0 | PRIORITY_LIMITER[3:0] | R/W | - | Priority limiter. This sets a priority threshold that incoming interrupt requests must exceed to trigger interrupt requests towards the controller and power management controller |

## 4.2 Interrupt vector register

The interrupt vector registers identify for each interrupt target the highest-priority enabled pending interrupt request that is present at the time when the register is being read. The software interrupt service routine must always read the vector register that corresponds to the interrupt target. The interrupt vector content can be used as vector into a memory based table like that shown in Figure 9–19. This table has 32 entries. To be able to use the register content as a full 32-bit address pointer the table must be aligned to a 512-byte address boundary (or 2048 to be future-proof). If only the index variable is used as offset into the table then this address alignment is not required. Each table entry is 64 bits wide. It is recommended to pack for each table entry:

- The start address of a peripheral-specific interrupt service routine, plus
- The associated priority-limiter value (if nesting of interrupt service routines is performed)

A vector with index 0 indicates that no interrupt is pending with a priority above the priority threshold. For this special-case entry the vector table should implement a ‘no-interrupt’ handler.

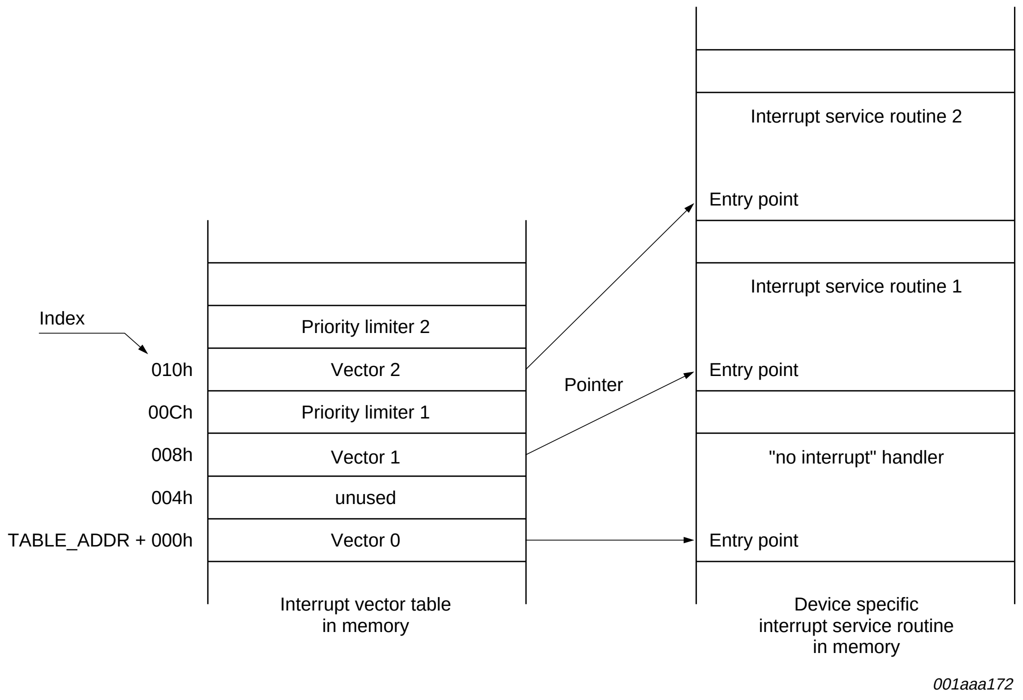


**Fig 19. Memory-based interrupt vector and priority table**

Table 9–84 shows the bit assignment of the INT_VECTOR registers.

**Table 84. INT_VECTORn register bit description (INT_VECTOR0/1, addresses 0xFFFF F100 and 0xFFFF F104)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 11 | TABLE_ADDR[20:0] | R/W | - | Table start address. This indicates the lower address boundary of a 512-byte aligned vector table in memory. To be compatible with future extension an address boundary of 2048 bytes is recommended |
| 10 to 9 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 8 to 3 | INDEX[5:0] | R/W[1] | | Index. This indicates the interrupt request line of the interrupt request to be served by the controller |
| | | | 00 0000 | No interrupt request to be serviced |
| | | | 00 0001 | Service interrupt request at input 1 |
| | | | : | : |
| | | | 01 1111 | Service interrupt request at input 31 |
| 2 to 0 | NULL[2:0] | R/W[1] | 0h | Always reflecting logic 0s |

[1] Write as 0.

## 4.3 Interrupt-pending register 1

The interrupt-pending register gathers the pending bits of interrupt requests 1 to 31. Software can make use of this feature to gain a faster overview of pending interrupts than it would get by reading the individual interrupt request registers.

The INT_PENDING_1_31 register is read-only.

Table 9–85 shows the bit assignment of the INT_PENDING_1_31 register.

**Table 85. INT_PENDING_1_31 register bit description (INT_PENDING_1_31, address 0xFFFF F200)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | PENDING[31] | R | 1 | Interrupt request 31 is pending |
| | | | 0 | There is no interrupt request 31 |
| : | : | | : | : |
| 1 | PENDING[1] | R | 1 | Interrupt request 1 is pending |
| | | | 0 | There is no interrupt request 1 |
| 0 | | R | 0 | Reserved; read as logic 0 |

## 4.4 Interrupt-pending register 2

The interrupt-pending register gathers the pending bits of all interrupt requests 32 to 63. Software can make use of this feature to gain a faster overview on pending interrupts than it would get by reading the individual interrupt request registers.

The INT_PENDING_32_63 register is read only.

Table 9–86 shows the bit assignment of the INT_PENDING_32_63 register.

**Table 86. INT_PENDING_32_63 register bit description (INT_PENDING_32_63, address 0xFFFF F204)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 25 | reserved | R | - | Reserved; read as don't care |
| 24 | PENDING[56] | R | 1 | Interrupt request 56 is pending |
| | | | 0 | There is no interrupt request 56 |
| : | : | | : | : |
| 0 | PENDING[32] | R | 1 | Interrupt request 32 is pending |
| | | | 0 | There is no interrupt request 32 |

## 4.5 Interrupt controller features register

The interrupt controller features register indicates the VIC configuration which an ISR can use for implementing interrupt controller configuration-specific behavior.

The INT_FEATURES register is read-only

Table 9–87 shows the bit assignment of the INT_FEATURES register.

**Table 87. INT_FEATURES register bit description (INT_FEATURES, address 0xFFFF F300)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; read as don't care |
| 21 to 16 | T | R | | Number of targets (minus one) |
| | | | 01h* | |
| 15 to 8 | P | R | | Number of priorities (minus one) |

**Table 87. INT_FEATURES register bit description (INT_FEATURES, address 0xFFFF F300)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| | | | 0Fh* | |
| 7 to 0 | N | R | | Number of interrupt requests |
| | | | 3Fh* | |

## 4.6 Interrupt request register

The reference between the interrupt source and interrupt request line is reflected in Table 9–88.

**Table 88. Interrupt source and request reference**

| Interrupt request | Interrupt source | Description |
|---|---|---|
| 1 | Watchdog | Interrupt from Watchdog timer |
| 2 | timer 0 | Capture or match interrupt from timer 0 |
| 3 | timer 1 | Capture or match interrupt from timer 1 |
| 4 | timer 2 | Capture or match interrupt from timer 2 |
| 5 | timer 3 | Capture or match interrupt from timer 3 |
| 6 | UART 0 | General interrupt from UART 0 |
| 7 | UART 1 | General interrupt from UART 1 |
| 8 | SPI 0 | General interrupt from SPI 0 |
| 9 | SPI 1 | General interrupt from SPI 1 |
| 10 | SPI 2 | General interrupt from SPI 2 |
| 11 | flash | Signature, burn or erase finished interrupt from flash |
| 12 | embedded RT-ICE | Comms Rx for ARM debug mode |
| 13 | embedded RT-ICE | Comms Tx for ARM debug mode |
| 14 | MSCSS timer 0 | Capture or match interrupt from MSCSS timer 0 |
| 15 | MSCSS timer 1 | Capture or match interrupt from MSCSS timer 1 |
| 16 | ADC int_req 0 | ADC interrupt from ADC 0 |
| 17 | ADC int_req 1 | ADC interrupt from ADC 1 |
| 18 | ADC int_req 2 | ADC interrupt from ADC 2 |
| 19 | PWM 0 | PWM interrupt from PWM 0 |
| 20 | PWM capt match 0 | PWM capture/match interrupt from PWM 0 |
| 21 | PWM 1 | PWM interrupt from PWM 1 |
| 22 | PWM capt match 1 | PWM capture/match interrupt from PWM 1 |
| 23 | PWM 2 | PWM interrupt from PWM 2 |
| 24 | PWM capt match 2 | PWM capture/match interrupt from PWM 2 |
| 25 | PWM 3 | PWM interrupt from PWM 3 |
| 26 | PWM capt match 3 | PWM capture/match interrupt from PWM 3 |
| 27 | Event Router | Event, wake up tick interrupt from Event Router |
| 28 | LIN master controller 0 | General interrupt from LIN master controller 0 |
| 29 | LIN master controller 1 | General interrupt from LIN master controller 1 |
| 30 | I2C0 | I$^2$C interrupt from I$^2$C0 (SI state change) |

Table 88. Interrupt source and request reference *…continued*

| Interrupt request | Interrupt source | Description |
|---|---|---|
| 31 | I2C1 | I$^2$C interrupt from I$^2$C1 (SI state change) |
| 32 | GPDMA | DMA |
| 33 | GPDMA | DMA err |
| 34 | GPDMA | DMA tc |
| 35 | all CAN controllers | FullCAN |
| 36 | all CAN controllers | Combined general interrupt of all CAN controllers and the CAN look-up table[1] |
| 37 | CAN controller 0 | Message-received interrupt from CAN controller 0[2] |
| 38 | CAN controller 1 | Message-received interrupt from CAN controller 1[2] |
| 39 - 42 | - | reserved |
| 43 | CAN controller 0 | Message-transmitted interrupt from CAN controller 0 |
| 44 | CAN controller 1 | Message-transmitted interrupt from CAN controller 1 |
| 45 | USB I2C | |
| 46 | USB device, high-priority | |
| 47 | USB device, low-priority | |
| 48 | USB device DMA | |
| 49 | USB host interrupt | |
| 50 | USB ATX | |
| 51 | USB OTG timer | |
| 52 | QEI | quadrature encoder interrupt |
| 53 - 54 | - | reserved |
| 55 | CGU0 | |
| 56 | CGU1 | |
| 57 - 63 | - | reserved |

[1] Combined general interrupt of all CAN controllers and the CAN look-up table; The following interrupts are combined here: error-warning interrupt (EWI), data-overrun interrupt (DOI), error-passive interrupt (EPI), arbitration-lost Interrupt (ALI), bus-error Interrupt (BEI) and look-up table error interrupt (CALUTE); see Section 21–9.4and Section 21–10.8 for details.

[2] Message-received interrupt from a CAN controller. The receive interrupt (RI) and the ID ready interrupt (IDI) are combined here; see Section 21–9.14 for details.

The interrupt request registers hold the configuration information related to interrupt request inputs of the interrupt controller and allow it to issue software interrupt requests. Each interrupt line has its own interrupt request register.

Table 9–89 shows the bit assignment of the INT_REQUEST register.

**Table 89. INT_REQUESTn register bit description (INT_REQUEST1 to 56, addresses 0xFFFF F404 to 0xFFFF F4E0).**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | PENDING | R | | Pending interrupt request. This reflects the state of the interrupt source channel. The pending status is also visible in the interrupt-pending register |
| | | | 1 | An interrupt request is pending |
| | | | 0 | There is no interrupt request |
| 30 | SET_SWINT | W | | Set software-interrupt request |
| | | | 1 | Sets the local software-interrupt request state |
| | | | 0* | No effect on the local software-interrupt request state. This bit is always read as logic 0 |
| 29 | CLR_SWINT | W | | Clear software-interrupt request |
| | | | 1 | clears the local software-interrupt request state |
| | | | 0* | no effect on the local software-interrupt request state. This bit is always read as logic 0 |
| 28 | WE_PRIORITY_LEVEL | W | | Write-enable priority level |
| | | | 1 | Enables the bit-state change during the same register access |
| | | | 0 | Does not change the bit state. This bit is always read as logic 0 |
| 27 | WE_TARGET | W | - | Write-enable target |
| | | | 1 | Enables the bit-state change during the same register access. For changing the bit state software must first disable the interrupt request (bit ENABLE = 0), then change this bit and finally re-enable the interrupt request (bit ENABLE = 1) |
| | | | 0 | Does not change this bit state. This bit is always read as logic 0 |
| 26 | WE_ENABLE | W | | Write enable |
| | | | 1 | Enables this bit-state change during the same register access |
| | | | 0 | Does not change this bit state. This bit is always read as logic 0 |
| 25 | WE_ACTIVE_LOW | W | | Write-enable active LOW |
| | | | 1 | Enables the bit-state change during the same register access |
| | | | 0 | Does not change the bit state. This bit is always read as logic 0 |
| 24 to 18 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 89. INT_REQUESTn register bit description (INT_REQUEST1 to 56, addresses 0xFFFF F404 to 0xFFFF F4E0).**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 17 | ACTIVE_LOW | R/W | | Active-LOW interrupt line. This selects the polarity of the interrupt request line. State changing is only possible if the corresponding write-enable bit has been set |
| | | | 1 | The interrupt request is active LOW |
| | | | 0* | The interrupt request is active HIGH |
| 16 | ENABLE | R/W | | Enable interrupt request. This controls interrupt-request processing by the interrupt controller. State changing is only possible if the corresponding write-enable bit has been set |
| | | | 1 | The interrupt request may cause an ARM processor interrupt request if further conditions become true |
| | | | 0* | The interrupt request is discarded and will not cause an ARM processor interrupt |
| 15 to 9 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 8 | TARGET | R/W | | Interrupt target. This defines the target of an interrupt request. State changing is only possible if the corresponding write-enable bit has been set |
| | | | 1 | The target is the IRQ |
| | | | 0* | The target is the FIQ |
| 7 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 3 to 0 | PRIORITY_LEVEL[3:0] | R/W | - | Interrupt priority level. This determines the priority level of the interrupt request. State changing is only possible if the corresponding write-enable bit has been set. Priority level 0 masks the interrupt request, so it is ignored. Priority level 1 has the lowest priority and level 15 the highest |

# UM10316

# Chapter 10: LPC29xx general system control

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

This chapter contains power control, interrupt, and wake-up features that pertain to various functions and peripherals on the LPC29xx.

## 3. Power modes

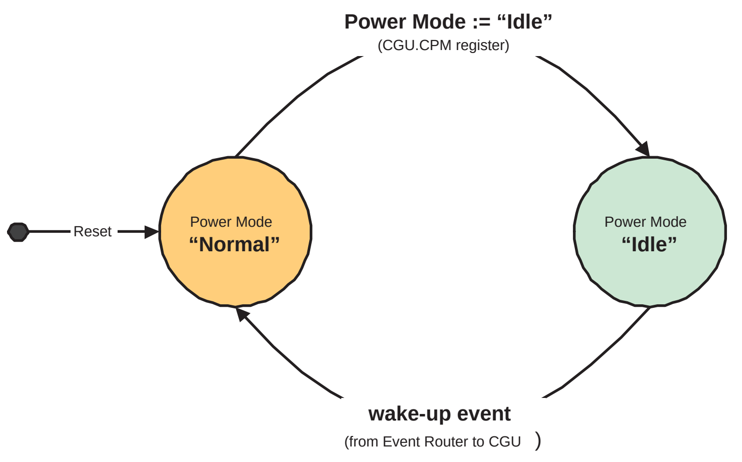


**Fig 20. Power modes**

The device operates in normal-power mode after reset. In this mode the device is fully functional, i.e. all clock domains are available[1]. The system can be put into idle-power mode either partially or fully. In this mode selected clock domains are switched off, and this might also suspend execution of the software. The clock domains are enabled again upon a wake-up event. This wake-up event is provided by the Event Router.

The clock domains that can be switched off during idle-power mode depend on the selected wake-up events. For an external interrupt (e.g. EXTINT0) no active clock is required, i.e. all clock domains can be switched off. However, for wake-up on a timer interrupt the clock domain of the timer should stay enabled during low-power mode. In general, each subsystem that might cause a wake-up upon an interrupt must be excluded from the low power mode, i.e. the clock domain of the subsystem should stay enabled.[2]

1. Although all clock domains are available, not all the domains are enabled. E.g. the ADC clock domain is switched off by default after reset.
2. The CAN and LIN controllers can issue a wake-up event via activity on the CAN or LIN bus. This feature does not require an active clock for their subsystem; but the first message can be lost.

Setting the power mode and configuring the clock domains is handled by the CGU, see Section 3–3. Configuration of wake-up events is handled by the Event Router, see Section 8–2.

## 4. Reset and power-up behavior

The LPC29xx contains external reset input and internal power-up reset circuits. This ensures that a reset is internally extended internally until the oscillators and flash have reached a stable state. Table 10–90 shows the reset pin.

**Table 90. Reset pin**

| Symbol | Direction | Description |
|---|---|---|
| RSTN | in | external reset input, active LOW; pulled up internally |

At activation of the RSTN pin the JTAGSEL pin is sensed as logic LOW. If this is the case the LPC29xx is assumed to be connected to debug hardware, and internal circuits re-program the source for the BASE_SYS_CLK to be the crystal oscillator instead of the Low-Power Ring Oscillator (LP_OSC). This is required because the clock rate when running at LP_OSC speed is too low for the external debugging environment.

## 5. Functional description of the interrupt and wake-up structure

An overview of the interrupt and wake-up structure is given in Figure 10–21. The main functions are:

- Events and interrupt requests causing an interrupt (IRQ or FIQ) on the ARM processor.
- Events and interrupt requests causing a wake-up. During low-power mode selected clock domains are switched off, and they are turned on by this wake-up.

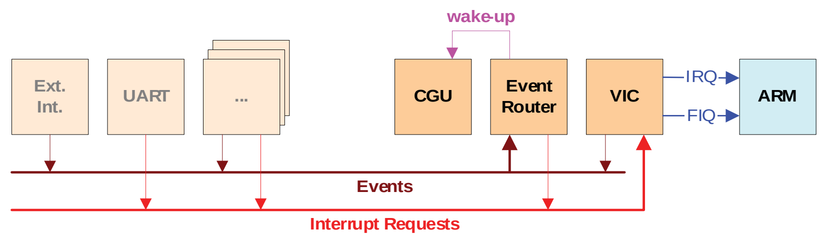


**Fig 21. Interrupt and wake-up structure**

In this case the VIC (Vectored Interrupt Controller) is configured to send an interrupt (IRQ or FIQ) towards the ARM processor. Examples are interrupts to indicate the reception of data via a serial interface, or timer interrupts. The Event Router serves as a multiplexer for internal and external events and indicates the occurrence of such an event towards the VIC (Event-Router interrupt). The Event Router is also able to latch the occurrence of these events (level or edge-triggered).

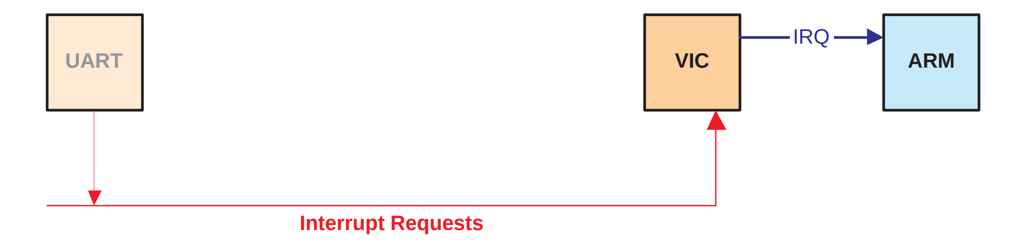


Fig 22. Interrupt (UART) causing an IRQ

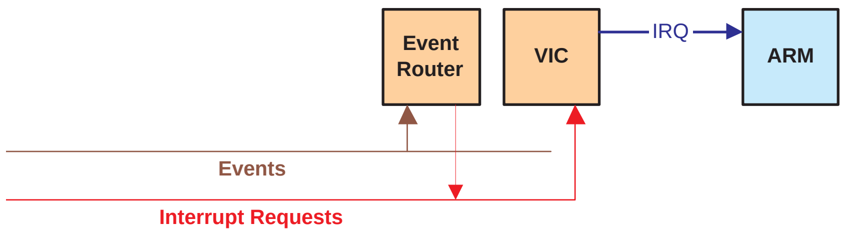


Fig 23. Event causing an IRQ

## 6. Interrupt device architecture

In the LPC29xx a general approach is taken to generate interrupt requests towards the CPU. A vectored Interrupt Controller (VIC) receives and collects the interrupt requests as generated by the several modules in the device.

Figure 10–24 shows the logic used to gate the event signal originating from the function with the parameters provided by the user software.

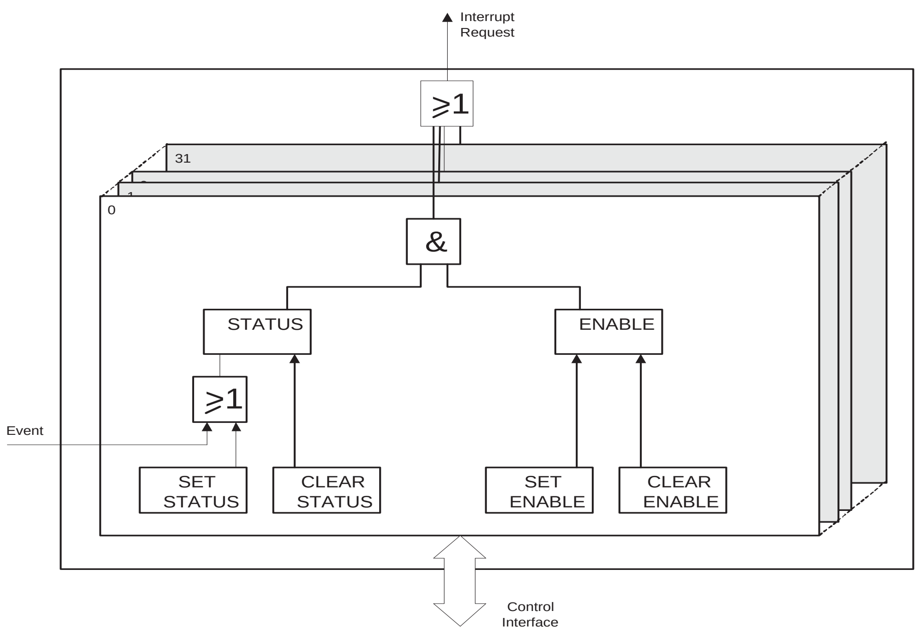


**Fig 24. Interrupt device architecture**

A set of software-accessible variables is provided for each interrupt source to control and observe interrupt request generation. In general, a pair of read-only registers is used for each event that leads to an interrupt request:

- STATUS captures the event. The variable is typically set by a hardware event and cleared by the software ISR, but for test purposes it can also be set by software
- ENABLE enables the assertion of an interrupt-request output signal for the captured event

In conjunction with the STATUS/ENABLE variables, commands are provided to set and clear the variable state through a software write-action to write-only registers. These commands are SET_STATUS, CLR_STATUS, SET_ENABLE and CLR_ENABLE.

The event signal is logically OR-ed with its associated SET_STATUS register bit, so both events writing to the SET_STATUS register sets the STATUS register.

Typically, the result of multiple STATUS/ENABLE pairs is logically OR-ed per functional group, forming an interrupt request signal towards the Vectored Interrupt Controller.

## 6.1 Interrupt registers

A list is provided for each function in the detailed block-description part of this document, containing the interrupt sources for that function. A table is also provided to indicate the bit positions per interrupt source. These positions are identical for all the six registers INT_STATUS, INT_ENABLE, INT_SET_STATUS, INT_CLEAR_STATUS, INT_SET_ENABLE and INT_CLEAR_ENABLE.

Up to 32 interrupt bits are available for each register.

### 6.1.1 Interrupt clear-enable register

Write ‘1’ actions to this register set one or more ENABLE variables in the INT_ENABLE register. INT_SET_ENABLE is write-only. Writing a 0 has no effect.

Table 91. INT_CLR_ENABLE register bit description

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | CLR_ENABLE[i] | W | 1 | Clears the ENABLE[i] variable in corresponding INT_ENABLE register (set to 0) |

### 6.1.2 Interrupt set-enable register

Write ‘1’ actions to this register set one or more ENABLE variables in the INT_ENABLE register. INT_SET_ENABLE is write-only. Writing a 0 has no effect.

Table 92. INT_SET_ENABLE register bit description

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | SET_ENABLE[i] | W | 1 | Sets the ENABLE[i] variable in corresponding INT_ENABLE register to 1 |

### 6.1.3 Interrupt status register

The interrupt status register reflects the status of the corresponding interrupt event that leads to an interrupt request. INT_STATUS is a read-only register. Its content is either changed by a hardware event (from logic 0 to 1 in the case of an event), or by software writing a 1 to the INT_CLR_STATUS or INT_SET_STATUS register.

Table 93. INT_STATUS register bit description
** = reset value*

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | STATUS[i] | R | 1 | Event captured; request for interrupt service on the corresponding interrupt request signal if ENABLE[i] = 1 interrupt for end of scan |
| | | | 0* | |

### 6.1.4 Interrupt enable register

This register enables or disables generation of interrupt requests on associated interrupt-request output signals. INT_ENABLE is a read-only register. Its content is changed by software writing to the INT_CLR_ENABLE or INT_SET_ENABLE registers.

Table 94. INT_ENABLE register bit description
** = reset value*

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | ENABLE[i] | R | 1 | Enables interrupt request generation. The corresponding interrupt request output signal is asserted when STATUS[i] =1 |
| | | | 0* | |

### 6.1.5 Interrupt clear-status register

Write ‘1’ actions to this register clear one or more status variables in the INT_STATUS register. Writing a ‘0’ has no effect.

Table 95. INT_CLR_STATUS register bit description

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | CLR_STATUS[i] | W | 1 | Clears STATUS[i] variable in INT_STATUS register (set to 0) |

### 6.1.6 Interrupt set-status register

Write ‘1’ actions to this register set one or more STATUS variables in the INT_STATUS register. This register is write-only and is intended for debug purposes. Writing a ‘0’ has no effect.

Table 96. INT_SET_STATUS register bit description

| Bit | Variable Name | Access | Value | Description |
|---|---|---|---|---|
| i | SET_STATUS[i] | W | 1 | Sets STATUS[i] variable in INT_STATUS register to 1 |

## 7. ISR functional description

The LPC29xx includes several peripherals, some of these influence each other during normal operation: for example the behaviors of the VIC and the Event Router. In most cases interrupt handling is controlled by some kind of OS, so the VIC and event-router functionality is divided into two components, ISR and ESR (Section 10–8). The ISR component can be used in situations where no OS is present or the OS does not support this functionality.

The ISR component also makes possible recursive calls to tmISR_EnableInterrupts and tmISR_DisableInterrupts. In this way atomic actions can be created, and can call other functions that contain atomic actions. Enabling or disabling the interrupts is dealt with automatically. A general rule is to keep atomic actions as small as possible.

## 8. Event-service routine (ESR) - Event handling

### 8.1 ESR functional description

This driver converts generated events to interrupt signals that are asserted in the VIC. It does not cover wake-up and power functions since these are handled by the CGU.

External interrupts are routed via the Event Router. When one of these signals is asserted the Event Router generates an interrupt on the VIC. The VIC then asserts the ARM core.

Handling of the VIC is done by the OS or by the ISR driver (see Section 10–7). Before the ESR driver is used the interrupt-handling software must be initialized. This is done by the OS or by the ISR driver.

The Event Router reacts to certain events when they are enabled. If an enabled event is asserted, the Event Router signals the VIC. This leads to execution of a special interrupt function: tmESR_EventDispatcher. This function checks the event-router status and executes the ESR of the active event source.

Usage of the ESR driver consists of several steps:

- Initialization of the driver:

  - Initialization of the interrupt functionality (outside the scope of this driver)
  - Installation of the event-dispatcher interrupt function
  - Initialization of the ESR driver
- Installation of the ESR:
  - Configure the signal specifications for external interrupts

    With this API the edge/level sensitivity can be programmed
  - Install the ESR handler.

    This function installs the ESR handler in the ESR vector table.
  - Enable the ESR handler.
  - Enable the specified event.

## 9. Wake-up

In low-power mode, selected idle clock domains are switched off. The wake-up signal towards the CGU enables the clock of these domains. A typical application is to configure all clock domains to switch off. Since the clock of the ARM processor is also switched off, execution of software is suspended and resumed on wake-up.

In this case the Event Router is configured to send a wake-up signal towards the CGU (Clock Generation Unit). Examples are events to indicate the reception of data (e.g. on the CAN receiver) or external interrupts.

The VIC can be used (IRQ wake-up event or FIQ wake-up event of the Event Router) to generate a wake-up event on an interrupt occurrence. This is only possible if the clock domain of the interrupt source is excluded from low-power mode. The VIC does not need a clock to generate these wake-up events.

Examples of use are to configure a timer to wake up the system after a defined time, or to wake up on receiving data via the UART.

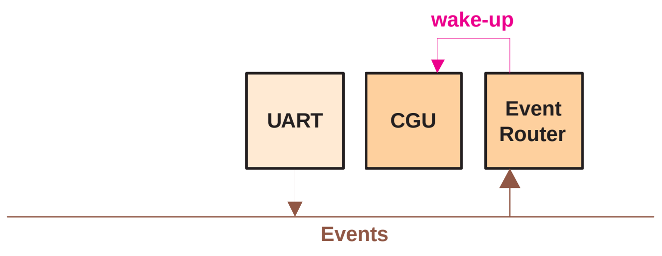


**Fig 25. Interrupt (UART) causing a wake-up**

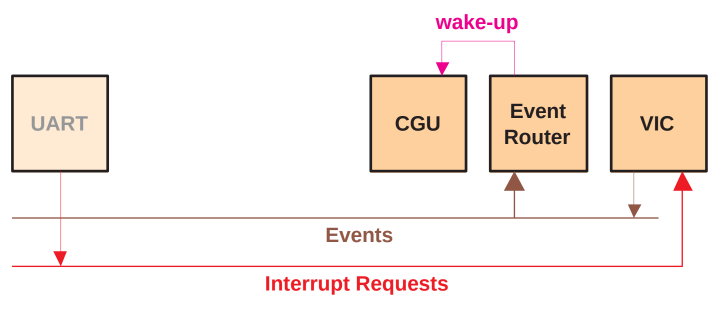


**Fig 26. Event causing a wake-up**

# UM10316

# Chapter 11: LPC29xx pin configuration

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

See Table 11–97 for pin configurations of all LPC29xx parts.

**Table 97. Feature overview**

| Part | Pins | Pin configuration | Pin assignment |
|---|---|---|---|
| LPC2917/19/01 | 144 | Figure 11–27 | Table 11–98 |
| LPC2921/23/25 | 100 | Figure 11–28 | Table 11–99 |
| LPC2926/27/29 | 144 | Figure 11–29 | Table 11–100 |
| LPC2930 | 208 | Figure 11–30 | Table 11–101 |
| LPC2939 | 208 | Figure 11–30 | Table 11–101 |

## 2. LPC2917/19/01 pinning information

The LPC29xx have up to four ports: two of 32 pins each, one of 28 pins and one of 16 pins. The pin to which each function is assigned is controlled by the SFSP registers in the SCU. The functions combined on each port pin are shown in the pin description tables in this section.

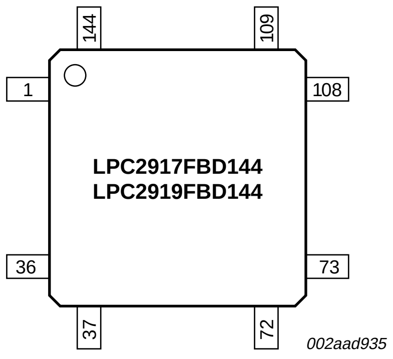


**Fig 27. Pin configuration for SOT486-1 (LQFP144)**

**Table 98. LPC2917/19/01 LQFP144 pin assignment**

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| TDO | 1[1] | IEEE 1149.1 test data out | | | |
| P2[21]SDI2/ PCAP2[1]/D19 | 2[1] | GPIO 2, pin 21 | SPI2 SDI | PWM2 CAP1 | EXTBUS D19 |
| P0[24]/TXD1/ TXDC1/SCS2[0] | 3[1] | GPIO 0, pin 24 | UART1 TXD | CAN1 TXD | SPI2 SCS0 |
| P0[25]/RXD1/ RXDC1/SDO2 | 4[1] | GPIO 0, pin 25 | UART1 RXD | CAN1 RXD | SPI2 SDO |

**Table 98. LPC2917/19/01 LQFP144 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| P0[26]/TXD1/ SDI2 | 5[1] | GPIO 0, pin 26 | - | UART1 TXD | SPI2 SDI |
| P0[27]/RXD1/ SCK2 | 6[1] | GPIO 0, pin 27 | - | UART1 RXD | SPI2 SCK |
| P0[28]/CAP0[0]/ MAT0[0] | 7[1] | GPIO 0, pin 28 | - | TIMER0 CAP0 | TIMER0 MAT0 |
| P0[29]/CAP0[1]/ MAT0[1] | 8[1] | GPIO 0, pin 29 | - | TIMER0 CAP1 | TIMER0 MAT1 |
| $V_{DD(IO)}$ | 9 | 3.3 V power supply for I/O | | | |
| P2[22]/SCK2/ PCAP2[2]/D20 | 10[1] | GPIO 2, pin 22 | SPI2 SCK | PWM2 CAP2 | EXTBUS D20 |
| P2[23]/SCS1[0]/ PCAP3[0]/D21 | 11[1] | GPIO 2, pin 23 | SPI1 SCS0 | PWM3 CAP0 | EXTBUS D21 |
| P3[6]/SCS0[3]/ PMAT1[0]/ TXDL1 | 12[1] | GPIO 3, pin 6 | SPI0 SCS3 | PWM1 MAT0 | LIN1/UART TXD |
| P3[7]/SCS2[1]/ PMAT1[1]/ RXDL1 | 13[1] | GPIO 3, pin 7 | SPI2 SCS1 | PWM1 MAT1 | LIN1/UART RXD |
| P0[30]/CAP0[2]/ MAT0[2] | 14[1] | GPIO 0, pin 30 | - | TIMER0 CAP2 | TIMER0 MAT2 |
| P0[31]/CAP0[3]/ MAT0[3] | 15[1] | GPIO 0, pin 31 | - | TIMER0 CAP3 | TIMER0 MAT3 |
| P2[24]/SCS1[1]/ PCAP3[1]/D22 | 16[1] | GPIO 2, pin 24 | SPI1 SCS1 | PWM3 CAP1 | EXTBUS D22 |
| P2[25]/SCS1[2]/ PCAP3[2]/D23 | 17[1] | GPIO 2, pin 25 | SPI1 SCS2 | PWM3 CAP2 | EXTBUS D23 |
| $V_{DD(CORE)}$ | 18 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 19 | ground for digital core | | | |
| P1[31]/CAP0[1]/ MAT0[1]/EI5 | 20[1] | GPIO 1, pin 31 | TIMER0 CAP1 | TIMER0 MAT1 | EXTINT5 |
| $V_{SS(IO)}$ | 21 | ground for I/O | | | |
| P1[30]/CAP0[0]/ MAT0[0]/EI4 | 22[1] | GPIO 1, pin 30 | TIMER0 CAP0 | TIMER0 MAT0 | EXTINT4 |
| P3[8]/SCS2[0]/ PMAT1[2] | 23[1] | GPIO 3, pin 8 | SPI2 SCS0 | PWM1 MAT2 | - |
| P3[9]/SDO2/PM AT1[3] | 24[1] | GPIO 3, pin 9 | SPI2 SDO | PWM1 MAT3 | - |
| P1[29]/CAP1[0]/ TRAP0/ PMAT3[5] | 25[1] | GPIO 1, pin 29 | TIMER1 CAP0 | PWM TRAP0 | PWM3 MAT5 |
| P1[28]/CAP1[1]/ TRAP1/ PMAT3[4] | 26[1] | GPIO 1, pin 28 | TIMER1 CAP1, ADC1 EXT START | PWM TRAP1 | PWM3 MAT4 |
| P2[26]/CAP0[2]/ MAT0[2]/EI6 | 27[1] | GPIO 2, pin 26 | TIMER0 CAP2 | TIMER0 MAT2 | EXTINT6 |

**Table 98. LPC2917/19/01 LQFP144 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| P2[27]/CAP0[3]/ MAT0[3]/EI7 | 28[1] | GPIO 2, pin 27 | TIMER0 CAP3 | TIMER0 MAT3 | EXTINT7 |
| P1[27]/CAP1[2]/ TRAP2/ PMAT3[3] | 29[1] | GPIO 1, pin 27 | TIMER1 CAP2, ADC2 EXT START | PWM TRAP2 | PWM3 MAT3 |
| P1[26]/ PMAT2[0]/ TRAP3/ PMAT3[2] | 30[1] | GPIO 1, pin 26 | PWM2 MAT0 | PWM TRAP3 | PWM3 MAT2 |
| $V_{DD(IO)}$ | 31 | 3.3 V power supply for I/O | | | |
| P1[25]/ PMAT1[0]/ PMAT3[1] | 32[1] | GPIO 1, pin 25 | PWM1 MAT0 | - | PWM3 MAT1 |
| P1[24]/ PMAT0[0]/ PMAT3[0] | 33[1] | GPIO 1, pin 24 | PWM0 MAT0 | - | PWM3 MAT0 |
| P1[23]/ RXD0/CS5 | 34[1] | GPIO 1, pin 23 | UART0 RXD | - | EXTBUS CS5 |
| P1[22]/TXD0/ CS4 | 35[1] | GPIO 1, pin 22 | UART0 TXD | - | EXTBUS CS4 |
| TMS | 36[1] | IEEE 1149.1 test mode select, pulled up internally | | | |
| TCK | 37[1] | IEEE 1149.1 test clock | | | |
| P1[21]/CAP3[3]/ CAP1[3]/D7 | 38[1] | GPIO 1, pin 21 | TIMER3 CAP3 | TIMER1 CAP3, MSCSS PAUSE | EXTBUS D7 |
| P1[20]/CAP3[2]/ SCS0[1]/D6 | 39[1] | GPIO 1, pin 20 | TIMER3 CAP2 | SPI0 SCS1 | EXTBUS D6 |
| P1[19]/CAP3[1]/ SCS0[2]/D5 | 40[1] | GPIO 1, pin 19 | TIMER3 CAP1 | SPI0 SCS2 | EXTBUS D5 |
| P1[18]/CAP3[0]/ SDO0/D4 | 41[1] | GPIO 1, pin 18 | TIMER3 CAP0 | SPI0 SDO | EXTBUS D4 |
| P1[17]/CAP2[3]/ SDI0/D3 | 42[1] | GPIO 1, pin 17 | TIMER2 CAP3 | SPI0 SDI | EXTBUS D3 |
| $V_{SS(IO)}$ | 43 | ground for I/O | | | |
| P1[16]/CAP2[2]/ SCK0/D2 | 44[1] | GPIO 1, pin 16 | TIMER2 CAP2 | SPI0 SCK | EXTBUS D2 |
| P2[0]/MAT2[0]/ TRAP3/D8 | 45[1] | GPIO 2, pin 0 | TIMER2 MAT0 | PWM TRAP3 | EXTBUS D8 |
| P2[1]/MAT2[1]/ TRAP2/D9 | 46[1] | GPIO 2, pin 1 | TIMER2 MAT1 | PWM TRAP2 | EXTBUS D9 |
| P3[10]/SDI2/ PMAT1[4] | 47[1] | GPIO 3, pin 10 | SPI2 SDI | PWM1 MAT4 | - |
| P3[11]/SCK2/ PMAT1[5] | 48[1] | GPIO 3, pin 11 | SPI2 SCK | PWM1 MAT5 | - |
| P1[15]/CAP2[1]/ SCS0[0]/D1 | 49[1] | GPIO 1, pin 15 | TIMER2 CAP1 | SPI0 SCS0 | EXTBUS D1 |

**Table 98. LPC2917/19/01 LQFP144 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| P1[14]/CAP2[0]/ SCS0[3]/D0 | 50[1] | GPIO 1, pin 14 | TIMER2 CAP0 | SPI0 SCS3 | EXTBUS D0 |
| P1[13]/SCL1/ EI3/WE_N | 51[1] | GPIO 1, pin 13 | EXTINT3 | I2C1 SCL | EXTBUS WE_N |
| P1[12]/SDA1/ EI2/OE_N | 52[1] | GPIO 1, pin 12 | EXTINT2 | I2C1 SDA | EXTBUS OE_N |
| $V_{DD(IO)}$ | 53 | 3.3 V power supply for I/O | | | |
| P2[2]/MAT2[2]/ TRAP1/D10 | 54[1] | GPIO 2, pin 2 | TIMER2 MAT2 | PWM TRAP1 | EXTBUS D10 |
| P2[3]/MAT2[3]/ TRAP0/D11 | 55[1] | GPIO 2, pin 3 | TIMER2 MAT3 | PWM TRAP0 | EXTBUS D11 |
| P1[11]/SCK1/ SCL0/CS3 | 56[1] | GPIO 1, pin 11 | SPI1 SCK | I2C0 SCL | EXTBUS CS3 |
| P1[10]/SDI1/ SDA0/CS2 | 57[1] | GPIO 1, pin 10 | SPI1 SDI | I2C0 SDA | EXTBUS CS2 |
| P3[12]/SCS1[0]/ EI4 | 58[1] | GPIO 3, pin 12 | SPI1 SCS0 | EXTINT4 | - |
| $V_{SS(CORE)}$ | 59 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 60 | 1.8 V power supply for digital core | | | |
| P3[13]/SDO1/ EI5/IDX0 | 61[1] | GPIO 3, pin 13 | SPI1 SDO | EXTINT5 | QEI0 IDX |
| P2[4]/MAT1[0]/ EI0/D12 | 62[1] | GPIO 2, pin 4 | TIMER1 MAT0 | EXTINT0 | EXTBUS D12 |
| P2[5]/MAT1[1]/ EI1/D13 | 63[1] | GPIO 2, pin 5 | TIMER1 MAT1 | EXTINT1 | EXTBUS D13 |
| P1[9]/SDO1/ RXDL1/CS1 | 64[1] | GPIO 1, pin 9 | SPI1 SDO | LIN1/UART RXD | EXTBUS CS1 |
| $V_{SS(IO)}$ | 65 | ground for I/O | | | |
| P1[8]/SCS1[0]/ TXDL1/CS0 | 66[1] | GPIO 1, pin 8 | SPI1 SCS0 | LIN1/UART TXD | EXTBUS CS0 |
| P1[7]/SCS1[3]/ RXD1/A7 | 67[1] | GPIO 1, pin 7 | SPI1 SCS3 | UART1 RXD | EXTBUS A7 |
| P1[6]/SCS1[2]/ TXD1/A6 | 68[1] | GPIO 1, pin 6 | SPI1 SCS2 | UART1 TXD | EXTBUS A6 |
| P2[6]/MAT1[2]/ EI2/D14 | 69[1][4] | GPIO 2, pin 6 | TIMER1 MAT2 | EXTINT2 | EXTBUS D14 |
| P1[5]/SCS1[1]/ PMAT3[5]/A5 | 70[1] | GPIO 1, pin 5 | SPI1 SCS1 | PWM3 MAT5 | EXTBUS A5 |
| P1[4]/SCS2[2]/ PMAT3[4]/A4 | 71[1] | GPIO 1, pin 4 | SPI2 SCS2 | PWM3 MAT4 | EXTBUS A4 |
| TRST_N | 72[1] | IEEE 1149.1 test reset NOT; active LOW; pulled up internally | | | |
| RST_N | 73[1] | asynchronous device reset; active LOW; pulled up internally | | | |
| $V_{SS(OSC)}$ | 74 | ground for oscillator | | | |
| XOUT_OSC | 75[3] | crystal out for oscillator | | | |
| XIN_OSC | 76[3] | crystal in for oscillator | | | |

**Table 98. LPC2917/19/01 LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| $V_{DD(OSC)}$ | 77 | 1.8 V supply for oscillator | | | |
| $V_{SS(PLL)}$ | 78 | ground for PLL | | | |
| P2[7]/MAT1[3]/ EI3/D15 | 79[1][4] | GPIO 2, pin 7 | TIMER1 MAT3 | EXTINT3 | EXTBUS D15 |
| P3[14]/SDI1/ EI6/TXDC0 | 80[1] | GPIO 3, pin 14 | SPI1 SDI | EXTINT6 | CAN0 TXD |
| P3[15]/SCK1/ EI7/RXDC0 | 81[1] | GPIO 3, pin 15 | SPI1 SCK | EXTINT7 | CAN0 RXD |
| $V_{DD(IO)}$ | 82 | 3.3 V power supply for I/O | | | |
| P2[8]/ CLK_OUT/ PMAT0[0]/ SCS0[2] | 83[1] | GPIO 2, pin 8 | CLK_OUT | PWM0 MAT0 | SPI0 SCS2 |
| P2[9]/PMAT0[1]/ SCS0[1] | 84[1] | GPIO 2, pin 9 | - | PWM0 MAT1 | SPI0 SCS1 |
| P1[3]/SCS2[1]/ PMAT3[3]/A3 | 85[1] | GPIO 1, pin 3 | SPI2 SCS1 | PWM3 MAT3 | EXTBUS A3 |
| P1[2]/SCS2[3]/ PMAT3[2]/A2 | 86[1] | GPIO 1, pin 2 | SPI2 SCS3 | PWM3 MAT2 | EXTBUS A2 |
| P1[1]/EI1/ PMAT3[1]/A1 | 87[1] | GPIO 1, pin 1 | EXTINT1 | PWM3 MAT1 | EXTBUS A1 |
| $V_{SS(CORE)}$ | 88 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 89 | 1.8 V power supply for digital core | | | |
| P1[0]/EI0/ PMAT3[0]/A0 | 90[1] | GPIO 1, pin 0 | EXTINT0 | PWM3 MAT0 | EXTBUS A0 |
| P2[10]/ PMAT0[2]/ SCS0[0] | 91[1] | GPIO 2, pin 10 | - | PWM0 MAT2 | SPI0 SCS0 |
| P2[11]/ PMAT0[3]/SCK0 | 92[1] | GPIO 2, pin 11 | - | PWM0 MAT3 | SPI0 SCK |
| P0[0]/PHB0/ TXDC0/D24 | 93[1] | GPIO 0, pin 0 | QEI0 PHB | CAN0 TXD | EXTBUS D24 |
| $V_{SS(IO)}$ | 94 | ground for I/O | | | |
| P0[1]/PHA0/ RXDC0/D25 | 95[1] | GPIO 0, pin 1 | QEI 0 PHA | CAN0 RXD | EXTBUS D25 |
| P0[2]/ CLK_OUT/ PMAT0[0]/D26 | 96[1] | GPIO 0, pin 2 | CLK_OUT | PWM0 MAT0 | EXTBUS D26 |
| P0[3]/PMAT0[1]/ D27 | 97[1] | GPIO 0, pin 3 | - | PWM0 MAT1 | EXTBUS D27 |
| P3[0]/PMAT2[0]/ CS6 | 98[1] | GPIO 3, pin 0 | - | PWM2 MAT0 | EXTBUS CS6 |
| P3[1]/PMAT2[1]/ CS7 | 99[1] | GPIO 3, pin 1 | - | PWM2 MAT1 | EXTBUS CS7 |
| P2[12]/ PMAT0[4]/SDI0 | 100[1] | GPIO 2, pin 12 | - | PWM0 MAT4 | SPI0 SDI |

**Table 98. LPC2917/19/01 LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| P2[13]/ PMAT0[5]/ SDO0 | 101[1] | GPIO 2, pin 13 | - | PWM0 MAT5 | SPI0 SDO |
| P0[4]/PMAT0[2]/ D28 | 102[1] | GPIO 0, pin 4 | - | PWM0 MAT2 | EXTBUS D28 |
| P0[5]/PMAT0[3]/ D29 | 103[1] | GPIO 0, pin 5 | - | PWM0 MAT3 | EXTBUS D29 |
| $V_{DD(IO)}$ | 104 | 3.3 V power supply for I/O | | | |
| P0[6]/ PMAT0[4]/D30 | 105[1] | GPIO 0, pin 6 | - | PWM0 MAT4 | EXTBUS D30 |
| P0[7]/ PMAT0[5]/D31 | 106[1] | GPIO 0, pin 7 | - | PWM0 MAT5 | EXTBUS D31 |
| $V_{DDA(ADC3V3)}$ | 107 | 3.3 V power supply for ADC | | | |
| JTAGSEL | 108[1] | TAP controller select input; LOW-level selects the ARM debug mode; HIGH-level selects boundary scan; pulled up internally. | | | |
| n.c. | 109 | not connected to a function, must be tied to 3.3 V power supply for ADC $V_{DDA(ADC3V3)}$. | | | |
| VREFP | 110[3] | HIGH reference for ADC | | | |
| VREFN | 111[3] | LOW reference for ADC | | | |
| P0[8]/IN1[0]/TX DL0/A20 | 112[4] | GPIO 0, pin 8 | ADC1 IN0 | LIN0/UART TXD | EXTBUS A20 |
| P0[9]/IN1[1]/ RXDL0/A21 | 113[4] | GPIO 0, pin 9 | ADC1 IN1 | LIN0/UART RXD | EXTBUS A21 |
| P0[10]/IN1[2]/ PMAT1[0]/A8 | 114[4] | GPIO 0, pin 10 | ADC1 IN2 | PWM1 MAT0 | EXTBUS A8 |
| P0[11]/IN1[3]/ PMAT1[1]/A9 | 115[4] | GPIO 0, pin 11 | ADC1 IN3 | PWM1 MAT1 | EXTBUS A9 |
| P2[14]/SDA1/ PCAP0[0]/BLS0 | 116[1] | GPIO 2, pin 14 | I2C1 SDA | PWM0 CAP0 | EXTBUS BLS0 |
| P2[15]/SCL1/ PCAP0[1]/BLS1 | 117[1] | GPIO 2, pin 15 | I2C1 SCL | PWM0 CAP1 | EXTBUS BLS1 |
| P3[2]/MAT3[0]/ PMAT2[2] | 118[1] | GPIO 3, pin 2 | TIMER3 MAT0 | PWM2 MAT2 | - |
| $V_{SS(IO)}$ | 119 | ground for I/O | | | |
| P3[3]/MAT3[1]/ PMAT2[3] | 120[1] | GPIO 3, pin 3 | TIMER3 MAT1 | PWM2 MAT3 | - |
| P0[12]/IN1[4]/ PMAT1[2]/A10 | 121[4] | GPIO 0, pin 12 | ADC1 IN4 | PWM1 MAT2 | EXTBUS A10 |
| P0[13]/IN1[5]/ PMAT1[3]/A11 | 122[4] | GPIO 0, pin 13 | ADC1 IN5 | PWM1 MAT3 | EXTBUS A11 |
| P0[14]/IN1[6]/ PMAT1[4]/A12 | 123[4] | GPIO 0, pin 14 | ADC1 IN6 | PWM1 MAT4 | EXTBUS A12 |
| P0[15]/IN1[7]/ PMAT1[5]/A13 | 124[4] | GPIO 0, pin 15 | ADC1 IN7 | PWM1 MAT5 | EXTBUS A13 |
| P0[16]IN2[0]/ TXD0/A22 | 125[4] | GPIO 0, pin 16 | ADC2 IN0 | UART0 TXD | EXTBUS A22 |

Table 98. LPC2917/19/01 LQFP144 pin assignment *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Default function | Function 1 | Function 2 | Function 3 |
| P0[17]/IN2[1]/ RXD0/A23 | 126[4] | GPIO 0, pin 17 | ADC2 IN1 | UART0 RXD | EXTBUS A23 |
| $V_{DD(CORE)}$ | 127 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 128 | ground for digital core | | | |
| P2[16]/TXD1/ PCAP0[2]/BLS2 | 129[1] | GPIO 2, pin 16 | UART1 TXD | PWM0 CAP2 | EXTBUS BLS2 |
| P2[17]/RXD1/ PCAP1[0]/BLS3 | 130[1] | GPIO 2, pin 17 | UART1 RXD | PWM1 CAP0 | EXTBUS BLS3 |
| $V_{DD(IO)}$ | 131 | 3.3 V power supply for I/O | | | |
| P0[18]/IN2[2]/ PMAT2[0]/A14 | 132[4] | GPIO 0, pin 18 | ADC2 IN2 | PWM2 MAT0 | EXTBUS A14 |
| P0[19]/IN2[3]/ PMAT2[1]/A15 | 133[4] | GPIO 0, pin 19 | ADC2 IN3 | PWM2 MAT1 | EXTBUS A15 |
| P3[4]/MAT3[2]/ PMAT2[4]/ TXDC1 | 134[1] | GPIO 3, pin 4 | TIMER3 MAT2 | PWM2 MAT4 | CAN1 TXD |
| P3[5]/MAT3[3]/ PMAT2[5]/ RXDC1 | 135[1] | GPIO 3, pin 5 | TIMER3 MAT3 | PWM2 MAT5 | CAN1 RXD |
| P2[18]/SCS2[1]/ PCAP1[1]/D16 | 136[1] | GPIO 2, pin 18 | SPI2 SCS1 | PWM1 CAP1 | EXTBUS D16 |
| P2[19]/SCS2[0]/ PCAP1[2]/D17 | 137[1] | GPIO 2, pin 19 | SPI2 SCS0 | PWM1 CAP2 | EXTBUS D17 |
| P0[20]/IN2[4]/ PMAT2[2]/A16 | 138[4] | GPIO 0, pin 20 | ADC2 IN4 | PWM2 MAT2 | EXTBUS A16 |
| P0[21]/IN2[5]/ PMAT2[3]/A17 | 139[4] | GPIO 0, pin 21 | ADC2 IN5 | PWM2 MAT3 | EXTBUS A17 |
| P0[22]/IN2[6]/ PMAT2[4]/A18 | 140[4] | GPIO 0, pin 22 | ADC2 IN6 | PWM2 MAT4 | EXTBUS A18 |
| $V_{SS(IO)}$ | 141 | ground for I/O | | | |
| P0[23]/IN2[7]/ PMAT2[5]/A19 | 142[4] | GPIO 0, pin 23 | ADC2 IN7 | PWM2 MAT5 | EXTBUS A19 |
| P2[20]/ PCAP2[0]/D18 | 143[1] | GPIO 2, pin 20 | SPI2 SDO | PWM2 CAP0 | EXTBUS D18 |
| TDI | 144[1] | IEEE 1149.1 data in, pulled up internally | | | |

[1] Bidirectional Pad; Analog Port; Plain Input; 3state Output; Slew Rate Control; 5V Tolerant; TTL with Hysteresis; Programmable Pull Up / Pull Down / Repeater.

[2] Analog Pad; Analog Input Output.

[3] Analog pad, <tbd>.

## 3. LPC2921/23/25 pin configuration

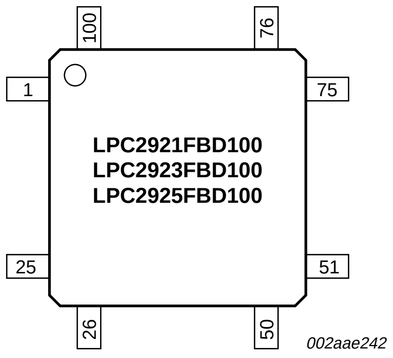


Fig 28. Pin configuration for SOT407-1 (LQFP100)

The LPC2921/23/25 uses three ports: port 1 with 32 pins, port 1 with 28 pins, and port 5 with 2 pins. Ports 4/3/2 are not used. The pin to which each function is assigned is controlled by the SFSP registers in the SCU. The functions combined on each port pin are shown in the pin description tables in this section.

Table 99. LPC2921/23/25 LQFP100 pin assignment

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| TDO | 1[1] | IEEE 1149.1 test data out | | | |
| P0[24]/TXD1/ TXDC1/SCS2[0] | 2[1] | GPIO 0, pin 24 | UART1 TXD | CAN1 TXD | SPI2 SCS0 |
| P0[25]/RXD1/ RXDC1/SDO2 | 3[1] | GPIO 0, pin 25 | UART1 RXD | CAN1 RXD | SPI2 SDO |
| P0[26]/TXD1/SDI2 | 4[1] | GPIO 0, pin 26 | - | UART1 TXD | SPI2 SDI |
| P0[27]/RXD1/SCK2 | 5[1] | GPIO 0, pin 27 | - | UART1 RXD | SPI2 SCK |
| P0[28]/CAP0[0]/ MAT0[0] | 6[1] | GPIO 0, pin 28 | - | TIMER0 CAP0 | TIMER0 MAT0 |
| P0[29]/CAP0[1]/ MAT0[1] | 7[1] | GPIO 0, pin 29 | - | TIMER0 CAP1 | TIMER0 MAT1 |
| $V_{DD(IO)}$ | 8 | 3.3 V power supply for I/O | | | |
| P0[30]/CAP0[2]/ MAT0[2] | 9[1] | GPIO 0, pin 30 | - | TIMER0 CAP2 | TIMER0 MAT2 |
| P0[31]/CAP0[3]/ MAT0[3] | 10[1] | GPIO 0, pin 31 | - | TIMER0 CAP3 | TIMER0 MAT3 |
| $V_{SS(IO)}$ | 11 | ground for I/O | | | |
| P5[19]/USB_D+ | 12[2] | GPIO 5, pin 19 | USB_D+ | - | - |
| P5[18]/USB_D– | 13[2] | GPIO 5, pin 18 | USB_D– | - | - |
| $V_{DD(IO)}$ | 14 | 3.3 V power supply for I/O | | | |
| $V_{DD(CORE)}$ | 15 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 16 | ground for core | | | |
| $V_{SS(IO)}$ | 17 | ground for I/O | | | |

**Table 99. LPC2921/23/25 LQFP100 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P1[27]/CAP1[2]/ TRAP2/PMAT3[3] | 18[1] | GPIO 1, pin 27 | TIMER1 CAP2, ADC2 EXT START | PWM TRAP2 | PWM3 MAT3 |
| P1[26]/PMAT2[0]/ TRAP3/PMAT3[2] | 19[1] | GPIO 1, pin 26 | PWM2 MAT0 | PWM TRAP3 | PWM3 MAT2 |
| $V_{DD(IO)}$ | 20 | 3.3 V power supply for I/O | | | |
| P1[25]/PMAT1[0]/ USB_VBUS/ PMAT3[1] | 21[1] | GPIO 1, pin 25 | PWM1 MAT0 | USB_VBUS | PWM3 MAT1 |
| P1[24]/PMAT0[0]/ $\overline{\text{USB\_CONNECT}}$/ PMAT3[0] | 22[1] | GPIO 1, pin 24 | PWM0 MAT0 | $\overline{\text{USB\_CONNECT}}$ | PWM3 MAT0 |
| P1[23]/RXD0 | 23[1] | GPIO 1, pin 23 | UART0 RXD | - | - |
| P1[22]/TXD0/ $\overline{\text{USB\_UP\_LED}}$ | 24[1] | GPIO 1, pin 22 | UART0 TXD | $\overline{\text{USB\_UP\_LED}}$ | - |
| TMS | 25[1] | IEEE 1149.1 test mode select, pulled up internally | | | |
| TCK | 26[1] | IEEE 1149.1 test clock | | | |
| P1[21]/CAP3[3]/ CAP1[3] | 27[1] | GPIO 1, pin 21 | TIMER3 CAP3 | TIMER1 CAP3, MSCSS PAUSE | - |
| P1[20]/CAP3[2]/ SCS0[1] | 28[1] | GPIO 1, pin 20 | TIMER3 CAP2 | SPI0 SCS1 | - |
| P1[19]/CAP3[1]/ SCS0[2] | 29[1] | GPIO 1, pin 19 | TIMER3 CAP1 | SPI0 SCS2 | - |
| P1[18]/CAP3[0]/ SDO0 | 30[1] | GPIO 1, pin 18 | TIMER3 CAP0 | SPI0 SDO | - |
| P1[17]/CAP2[3]/ SDI0 | 31[1] | GPIO 1, pin 17 | TIMER2 CAP3 | SPI0 SDI | - |
| $V_{SS(IO)}$ | 32 | ground for I/O | | | |
| P1[16]/CAP2[2]/ SCK0 | 33[1] | GPIO 1, pin 16 | TIMER2 CAP2 | SPI0 SCK | - |
| P1[15]/CAP2[1]/ SCS0[0] | 34[1] | GPIO 1, pin 15 | TIMER2 CAP1 | SPI0 SCS0 | - |
| P1[14]/CAP2[0]/ SCS0[3] | 35[1] | GPIO 1, pin 14 | TIMER2 CAP0 | SPI0 SCS3 | - |
| P1[13]/EI3/SCL1 | 36[1] | GPIO 1, pin 13 | EXTINT3 | I2C1 SCL | - |
| P1[12]/EI2/SDA1 | 37[1] | GPIO 1, pin 12 | EXTINT2 | I2C1 SDA | - |
| $V_{DD(IO)}$ | 38 | 3.3 V power supply for I/O | | | |
| P1[11]/SCK1/SCL0 | 39[1] | GPIO 1, pin 11 | SPI1 SCK | I2C0 SCL | - |
| P1[10]/SDI1/SDA0 | 40[1] | GPIO 1, pin 10 | SPI1 SDI | I2C0 SDA | - |
| $V_{SS(CORE)}$ | 41 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 42 | 1.8 V power supply for digital core | | | |
| P1[9]/SDO1 | 43[1] | GPIO 1, pin 9 | SPI1 SDO | - | - |
| $V_{SS(IO)}$ | 44 | ground for I/O | | | |

**Table 99. LPC2921/23/25 LQFP100 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P1[8]/SCS1[0]/ TXDL1/CS0 | 45[1] | GPIO 1, pin 8 | SPI1 SCS0 | - | - |
| P1[7]/SCS1[3]/RXD1 | 46[1] | GPIO 1, pin 7 | SPI1 SCS3 | UART1 RXD | - |
| P1[6]/SCS1[2]/TXD1 | 47[1] | GPIO 1, pin 6 | SPI1 SCS2 | UART1 TXD | - |
| P1[5]/SCS1[1]/ PMAT3[5] | 48[1] | GPIO 1, pin 5 | SPI1 SCS1 | PWM3 MAT5 | - |
| P1[4]/SCS2[2]/ PMAT3[4] | 49[1] | GPIO 1, pin 4 | SPI2 SCS2 | PWM3 MAT4 | - |
| TRST_N | 50[1] | IEEE 1149.1 test reset NOT; active LOW; pulled up internally | | | |
| RST_N | 51[1] | asynchronous device reset; active LOW; pulled up internally | | | |
| $V_{SS(OSC)}$ | 52 | ground for oscillator | | | |
| XOUT_OSC | 53[3] | crystal out for oscillator | | | |
| XIN_OSC | 54[3] | crystal in for oscillator | | | |
| $V_{DD(OSC)}$ | 55 | 1.8 V supply for oscillator. | | | |
| $V_{SS(PLL)}$ | 56 | ground for PLL | | | |
| $V_{DD(IO)}$ | 57 | 3.3 V power supply for I/O | | | |
| P1[3]/SCS2[1]/ PMAT3[3] | 58[1] | GPIO 1, pin 3 | SPI2 SCS1 | PWM3 MAT3 | - |
| P1[2]/SCS2[3]/ PMAT3[2] | 59[1] | GPIO 1, pin 2 | SPI2 SCS3 | PWM3 MAT2 | - |
| P1[1]/EI1/PMAT3[1] | 60[1] | GPIO 1, pin 1 | EXTINT1 | PWM3 MAT1 | - |
| $V_{SS(CORE)}$ | 61 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 62 | 1.8 V power supply for digital core | | | |
| P1[0]/EI0/PMAT3[0] | 63[1] | GPIO 1, pin 0 | EXTINT0 | PWM3 MAT0 | - |
| P0[0]/PHB0/ TXDC0/D24 | 64[1] | GPIO 0, pin 0 | QEI0 PHB | CAN0 TXD | - |
| $V_{SS(IO)}$ | 65 | ground for I/O | | | |
| P0[1]/PHA0/RXDC0 | 66[1] | GPIO 0, pin 1 | QEI0 PHA | CAN0 RXD | - |
| P0[2]/CLK_OUT/ PMAT0[0] | 67[1] | GPIO 0, pin 2 | CLK_OUT | PWM0 MAT0 | - |
| P0[3]/$\overline{\text{USB\_UP\_LED}}$/ PMAT0[1] | 68[1] | GPIO 0, pin 3 | $\overline{\text{USB\_UP\_LED}}$ | PWM0 MAT1 | - |
| P0[4]/PMAT0[2] | 69[1] | GPIO 0, pin 4 | - | PWM0 MAT2 | - |
| P0[5]/PMAT0[3] | 70[1] | GPIO 0, pin 5 | - | PWM0 MAT3 | - |
| $V_{DD(IO)}$ | 71 | 3.3 V power supply for I/O | | | |
| P0[6]/PMAT0[4] | 72[1] | GPIO 0, pin 6 | - | PWM0 MAT4 | - |
| P0[7]/PMAT0[5] | 73[1] | GPIO 0, pin 7 | - | PWM0 MAT5 | - |
| $V_{DDA(ADC3V3)}$ | 74 | 3.3 V power supply for ADC | | | |
| JTAGSEL | 75[1] | TAP controller select input; LOW-level selects the ARM debug mode; HIGH-level selects boundary scan; pulled up internally. | | | |
| n.c. | 76 | not connected to a function, must be tied to 3.3 V power supply for ADC $V_{DDA(ADC3V3)}$. | | | |
| VREFP | 77[3] | HIGH reference for ADC | | | |

**Table 99. LPC2921/23/25 LQFP100 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| VREFN | 78[3] | LOW reference for ADC | | | |
| P0[8]/IN1[0] | 79[4] | GPIO 0, pin 8 | ADC1 IN0 | - | - |
| P0[9]/IN1[1] | 80[4] | GPIO 0, pin 9 | ADC1 IN1 | - | - |
| P0[10]/IN1[2]/ PMAT1[0] | 81[4] | GPIO 0, pin 10 | ADC1 IN2 | PWM1 MAT0 | - |
| P0[11]/IN1[3]/ PMAT1[1] | 82[4] | GPIO 0, pin 11 | ADC1 IN3 | PWM1 MAT1 | - |
| $V_{SS(IO)}$ | 83 | ground for I/O | | | |
| P0[12]/IN1[4]/ PMAT1[2] | 84[4] | GPIO 0, pin 12 | ADC1 IN4 | PWM1 MAT2 | - |
| P0[13]/IN1[5]/ PMAT1[3] | 85[4] | GPIO 0, pin 13 | ADC1 IN5 | PWM1 MAT3 | - |
| P0[14]/IN1[6]/ PMAT1[4] | 86[4] | GPIO 0, pin 14 | ADC1 IN6 | PWM1 MAT4 | - |
| P0[15]/IN1[7]/ PMAT1[5] | 87[4] | GPIO 0, pin 15 | ADC1 IN7 | PWM1 MAT5 | - |
| P0[16]IN2[0]/TXD0 | 88[4] | GPIO 0, pin 16 | ADC2 IN0 | UART0 TXD | - |
| P0[17]/IN2[1]/ RXD0/A23 | 89[4] | GPIO 0, pin 17 | ADC2 IN1 | UART0 RXD | - |
| $V_{DD(CORE)}$ | 90 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 91 | ground for digital core | | | |
| $V_{DD(IO)}$ | 92 | 3.3 V power supply for I/O | | | |
| P0[18]/IN2[2]/ PMAT2[0] | 93[4] | GPIO 0, pin 18 | ADC2 IN2 | PWM2 MAT0 | - |
| P0[19]/IN2[3]/ PMAT2[1] | 94[4] | GPIO 0, pin 19 | ADC2 IN3 | PWM2 MAT1 | - |
| P0[20]/IN2[4]/ PMAT2[2] | 95[4] | GPIO 0, pin 20 | ADC2 IN4 | PWM2 MAT2 | - |
| P0[21]/IN2[5]/ PMAT2[3] | 96[4] | GPIO 0, pin 21 | ADC2 IN5 | PWM2 MAT3 | - |
| P0[22]/IN2[6]/ PMAT2[4]/A18 | 97[4] | GPIO 0, pin 22 | ADC2 IN6 | PWM2 MAT4 | - |
| $V_{SS(IO)}$ | 98 | ground for I/O | | | |

**Table 99. LPC2921/23/25 LQFP100 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P0[23]/IN2[7]/ PMAT2[5]/A19 | 99[4] | GPIO 0, pin 23 | ADC2 IN7 | PWM2 MAT5 | - |
| TDI | 100[1] | IEEE 1149.1 data in, pulled up internally | | | |

[1] Bidirectional Pad; Analog Port; Plain Input; 3state Output; Slew Rate Control; 5V Tolerant; TTL with Hysteresis; Programmable Pull Up / Pull Down / Repeater.

[2] USB pad, <tbd>.

[3] Analog Pad; Analog Input Output.

[4] Analog I/O pad, <tbd>.

## 4. LPC2927/29 pin configuration

The LPC2927/29 uses five ports: port 0 with 32 pins, ports 1 and 2 with 26 pins each, port 3 with 16 pins, and port 5 with 2 pins. Port 4 is not used. The pin to which each function is assigned is controlled by the SFSP registers in the SCU. The functions combined on each port pin are shown in the pin description tables in this section.

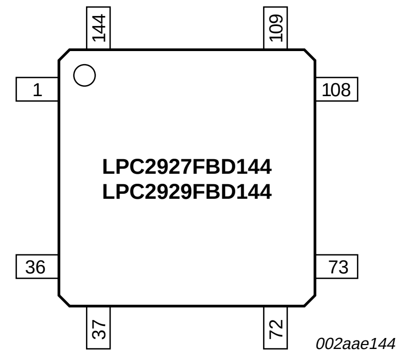


**Fig 29. Pin configuration for SOT486-1 (LQFP144)**

**Table 100. LQFP144 pin assignment**

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| TDO | 1[1] | IEEE 1149.1 test data out | | | |
| P2[21]/SDI2/ PCAP2[1]/D19 | 2[1] | GPIO 2, pin 21 | SPI2 SDI | PWM2 CAP1 | EXTBUS D19 |
| P0[24]/TXD1/ TXDC1/SCS2[0] | 3[1] | GPIO 0, pin 24 | UART1 TXD | CAN1 TXD | SPI2 SCS0 |
| P0[25]/RXD1/ RXDC1/SDO2 | 4[1] | GPIO 0, pin 25 | UART1 RXD | CAN1 RXD | SPI2 SDO |
| P0[26]/TXD1/SDI2 | 5[1] | GPIO 0, pin 26 | - | UART1 TXD | SPI2 SDI |
| P0[27]/RXD1/SCK2 | 6[1] | GPIO 0, pin 27 | - | UART1 RXD | SPI2 SCK |
| P0[28]/CAP0[0]/ MAT0[0] | 7[1] | GPIO 0, pin 28 | - | TIMER0 CAP0 | TIMER0 MAT0 |

**Table 100. LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P0[29]/CAP0[1]/ MAT0[1] | 8[1] | GPIO 0, pin 29 | - | TIMER0 CAP1 | TIMER0 MAT1 |
| $V_{DD(IO)}$ | 9 | 3.3 V power supply for I/O | | | |
| P2[22]/SCK2/ PCAP2[2]/D20 | 10[1] | GPIO 2, pin 22 | SPI2 SCK | PWM2 CAP2 | EXTBUS D20 |
| P2[23]/SCS1[0]/ PCAP3[0]/D21 | 11[1] | GPIO 2, pin 23 | SPI1 SCS0 | PWM3 CAP0 | EXTBUS D21 |
| P3[6]/SCS0[3]/ PMAT1[0]/TXDL1 | 12[1] | GPIO 3, pin 6 | SPI0 SCS3 | PWM1 MAT0 | LIN1/UART TXD |
| P3[7]/SCS2[1]/ PMAT1[1]/RXDL1 | 13[1] | GPIO 3, pin 7 | SPI2 SCS1 | PWM1 MAT1 | LIN1/UART RXD |
| P0[30]/CAP0[2]/ MAT0[2] | 14[1] | GPIO 0, pin 30 | - | TIMER0 CAP2 | TIMER0 MAT2 |
| P0[31]/CAP0[3]/ MAT0[3] | 15[1] | GPIO 0, pin 31 | - | TIMER0 CAP3 | TIMER0 MAT3 |
| P2[24]/SCS1[1]/ PCAP3[1]/D22 | 16[1] | GPIO 2, pin 24 | SPI1 SCS1 | PWM3 CAP1 | EXTBUS D22 |
| P2[25]/SCS1[2]/ PCAP3[2]/D23 | 17[1] | GPIO 2, pin 25 | SPI1 SCS2 | PWM3 CAP2 | EXTBUS D23 |
| $V_{SS(IO)}$ | 18 | ground for I/O | | | |
| P5[19]/USB_D+ | 19[2] | GPIO 5, pin 19 | USB_D+ | - | - |
| P5[18]/USB_D– | 20[2] | GPIO 5, pin 18 | USB_D– | - | - |
| $V_{DD(IO)}$ | 21 | 3.3 V power supply for I/O | | | |
| $V_{DD(CORE)}$ | 22 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 23 | ground for core | | | |
| $V_{SS(IO)}$ | 24 | ground for I/O | | | |
| P3[8]/SCS2[0]/ PMAT1[2] | 25[1] | GPIO 3, pin 8 | SPI2 SCS0 | PWM1 MAT2 | - |
| P3[9]/SDO2/ PMAT1[3] | 26[1] | GPIO 3, pin 9 | SPI2 SDO | PWM1 MAT3 | - |
| P2[26]/CAP0[2]/ MAT0[2]/EI6 | 27[1] | GPIO 2, pin 26 | TIMER0 CAP2 | TIMER0 MAT2 | EXTINT6 |
| P2[27]/CAP0[3]/ MAT0[3]/EI7 | 28[1] | GPIO 2, pin 27 | TIMER0 CAP3 | TIMER0 MAT3 | EXTINT7 |
| P1[27]/CAP1[2]/ TRAP2/PMAT3[3] | 29[1] | GPIO 1, pin 27 | TIMER1 CAP2, ADC2 EXT START | PWM TRAP2 | PWM3 MAT3 |
| P1[26]/PMAT2[0]/ TRAP3/PMAT3[2] | 30[1] | GPIO 1, pin 26 | PWM2 MAT0 | PWM TRAP3 | PWM3 MAT2 |
| $V_{DD(IO)}$ | 31 | 3.3 V power supply for I/O | | | |
| P1[25]/PMAT1[0]/ USB_VBUS/ PMAT3[1] | 32[1] | GPIO 1, pin 25 | PWM1 MAT0 | USB_VBUS | PWM3 MAT1 |

**Table 100. LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P1[24]/PMAT0[0]/ $\overline{\text{USB\_CONNECT}}$/ PMAT3[0] | 33[1] | GPIO 1, pin 24 | PWM0 MAT0 | $\overline{\text{USB\_CONNECT}}$ | PWM3 MAT0 |
| P1[23]/RXD0/ USB_SSPND/$\overline{\text{CS5}}$ | 34[1] | GPIO 1, pin 23 | UART0 RXD | USB_SSPND | EXTBUS $\overline{\text{CS5}}$ |
| P1[22]/TXD0/ $\overline{\text{USB\_UP\_LED}}$/ $\overline{\text{CS4}}$ | 35[1] | GPIO 1, pin 22 | UART0 TXD | $\overline{\text{USB\_UP\_LED}}$ | EXTBUS $\overline{\text{CS4}}$ |
| TMS | 36[1] | IEEE 1149.1 test mode select, pulled up internally | | | |
| TCK | 37[1] | IEEE 1149.1 test clock | | | |
| P1[21]/CAP3[3]/ CAP1[3]/D7 | 38[1] | GPIO 1, pin 21 | TIMER3 CAP3 | TIMER1 CAP3, MSCSS PAUSE | EXTBUS D7 |
| P1[20]/CAP3[2]/ SCS0[1]/D6 | 39[1] | GPIO 1, pin 20 | TIMER3 CAP2 | SPI0 SCS1 | EXTBUS D6 |
| P1[19]/CAP3[1]/ SCS0[2]/D5 | 40[1] | GPIO 1, pin 19 | TIMER3 CAP1 | SPI0 SCS2 | EXTBUS D5 |
| P1[18]/CAP3[0]/ SDO0/D4 | 41[1] | GPIO 1, pin 18 | TIMER3 CAP0 | SPI0 SDO | EXTBUS D4 |
| P1[17]/CAP2[3]/ SDI0/D3 | 42[1] | GPIO 1, pin 17 | TIMER2 CAP3 | SPI0 SDI | EXTBUS D3 |
| $V_{SS(IO)}$ | 43 | ground for I/O | | | |
| P1[16]/CAP2[2]/ SCK0/D2 | 44[1] | GPIO 1, pin 16 | TIMER2 CAP2 | SPI0 SCK | EXTBUS D2 |
| P2[0]/MAT2[0]/ TRAP3/D8 | 45[1] | GPIO 2, pin 0 | TIMER2 MAT0 | PWM TRAP3 | EXTBUS D8 |
| P2[1]/MAT2[1]/ TRAP2/D9 | 46[1] | GPIO 2, pin 1 | TIMER2 MAT1 | PWM TRAP2 | EXTBUS D9 |
| P3[10]/SDI2/ PMAT1[4] | 47[1] | GPIO 3, pin 10 | SPI2 SDI | PWM1 MAT4 | - |
| P3[11]/SCK2/ PMAT1[5]/USB_LS | 48[1] | GPIO 3, pin 11 | SPI2 SCK | PWM1 MAT5 | USB_LS |
| P1[15]/CAP2[1]/ SCS0[0]/D1 | 49[1] | GPIO 1, pin 15 | TIMER2 CAP1 | SPI0 SCS0 | EXTBUS D1 |
| P1[14]/CAP2[0]/ SCS0[3]/D0 | 50[1] | GPIO 1, pin 14 | TIMER2 CAP0 | SPI0 SCS3 | EXTBUS D0 |
| P1[13]/SCL1/ EI3/$\overline{\text{WE}}$ | 51[1] | GPIO 1, pin 13 | EXTINT3 | I2C1 SCL | EXTBUS $\overline{\text{WE}}$ |
| P1[12]/SDA1/ EI2/$\overline{\text{OE}}$ | 52[1] | GPIO 1, pin 12 | EXTINT2 | I2C1 SDA | EXTBUS $\overline{\text{OE}}$ |
| $V_{DD(IO)}$ | 53 | 3.3 V power supply for I/O | | | |
| P2[2]/MAT2[2]/ TRAP1/D10 | 54[1] | GPIO 2, pin 2 | TIMER2 MAT2 | PWM TRAP1 | EXTBUS D10 |
| P2[3]/MAT2[3]/ TRAP0/D11 | 55[1] | GPIO 2, pin 3 | TIMER2 MAT3 | PWM TRAP0 | EXTBUS D11 |

**Table 100. LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P1[11]/SCK1/ SCL0/$\overline{\text{CS3}}$ | 56[1] | GPIO 1, pin 11 | SPI1 SCK | I2C0 SCL | EXTBUS $\overline{\text{CS3}}$ |
| P1[10]/SDI1/ SDA0/$\overline{\text{CS2}}$ | 57[1] | GPIO 1, pin 10 | SPI1 SDI | I2C0 SDA | EXTBUS $\overline{\text{CS2}}$ |
| P3[12]/SCS1[0]/EI4/ USB_SSPND | 58[1] | GPIO 3, pin 12 | SPI1 SCS0 | EXTINT4 | USB_SSPND |
| $V_{SS(CORE)}$ | 59 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 60 | 1.8 V power supply for digital core | | | |
| P3[13]/SDO1/ EI5/IDX0 | 61[1] | GPIO 3, pin 13 | SPI1 SDO | EXTINT5 | QEI0 IDX |
| P2[4]/MAT1[0]/ EI0/D12 | 62[1] | GPIO 2, pin 4 | TIMER1 MAT0 | EXTINT0 | EXTBUS D12 |
| P2[5]/MAT1[1]/ EI1/D13 | 63[1] | GPIO 2, pin 5 | TIMER1 MAT1 | EXTINT1 | EXTBUS D13 |
| P1[9]/SDO1/ RXDL1/$\overline{\text{CS1}}$ | 64[1] | GPIO 1, pin 9 | SPI1 SDO | LIN1 RXD/UART RXD | EXTBUS $\overline{\text{CS1}}$ |
| $V_{SS(IO)}$ | 65 | ground for I/O | | | |
| P1[8]/SCS1[0]/ TXDL1/$\overline{\text{CS0}}$ | 66[1] | GPIO 1, pin 8 | SPI1 SCS0 | LIN1 TXD/ UART TXD | EXTBUS $\overline{\text{CS0}}$ |
| P1[7]/SCS1[3]/RXD1/ A7 | 67[1] | GPIO 1, pin 7 | SPI1 SCS3 | UART1 RXD | EXTBUS A7 |
| P1[6]/SCS1[2]/ TXD1/A6 | 68[1] | GPIO 1, pin 6 | SPI1 SCS2 | UART1 TXD | EXTBUS A6 |
| P2[6]/MAT1[2]/ EI2/D14 | 69[1] | GPIO 2, pin 6 | TIMER1 MAT2 | EXTINT2 | EXTBUS D14 |
| P1[5]/SCS1[1]/PMAT 3[5]/A5 | 70[1] | GPIO 1, pin 5 | SPI1 SCS1 | PWM3 MAT5 | EXTBUS A5 |
| P1[4]/SCS2[2]/PMAT 3[4]/A4 | 71[1] | GPIO 1, pin 4 | SPI2 SCS2 | PWM3 MAT4 | EXTBUS A4 |
| $\overline{\text{TRST}}$ | 72[1] | IEEE 1149.1 test reset NOT; active LOW; pulled up internally | | | |
| $\overline{\text{RST}}$ | 73[1] | asynchronous device reset; active LOW; pulled up internally | | | |
| $V_{SS(OSC)}$ | 74 | ground for oscillator | | | |
| XOUT_OSC | 75[3] | crystal out for oscillator | | | |
| XIN_OSC | 76[3] | crystal in for oscillator | | | |
| $V_{DD(OSC\_PLL)}$ | 77 | 1.8 V supply for oscillator and PLL | | | |
| $V_{SS(PLL)}$ | 78 | ground for PLL | | | |
| P2[7]/MAT1[3]/ EI3/D15 | 79[1] | GPIO 2, pin 7 | TIMER1 MAT3 | EXTINT3 | EXTBUS D15 |
| P3[14]/SDI1/ EI6/TXDC0 | 80[1] | GPIO 3, pin 14 | SPI1 SDI | EXTINT6 | CAN0 TXD |
| P3[15]/SCK1/ EI7/RXDC0 | 81[1] | GPIO 3, pin 15 | SPI1 SCK | EXTINT7 | CAN0 RXD |
| $V_{DD(IO)}$ | 82 | 3.3 V power supply for I/O | | | |

**Table 100. LQFP144 pin assignment** *...continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P2[8]/CLK_OUT/ PMAT0[0]/SCS0[2] | 83[1] | GPIO 2, pin 8 | CLK_OUT | PWM0 MAT0 | SPI0 SCS2 |
| P2[9]/ $\overline{\text{USB\_UP\_LED}}$/ PMAT0[1]/ SCS0[1] | 84[1] | GPIO 2, pin 9 | $\overline{\text{USB\_UP\_LED}}$ | PWM0 MAT1 | SPI0 SCS1 |
| P1[3]/SCS2[1]/ PMAT3[3]/A3 | 85[1] | GPIO 1, pin 3 | SPI2 SCS1 | PWM3 MAT3 | EXTBUS A3 |
| P1[2]/SCS2[3]/ PMAT3[2]/A2 | 86[1] | GPIO 1, pin 2 | SPI2 SCS3 | PWM3 MAT2 | EXTBUS A2 |
| P1[1]/EI1/ PMAT3[1]/A1 | 87[1] | GPIO 1, pin 1 | EXTINT1 | PWM3 MAT1 | EXTBUS A1 |
| $V_{SS(CORE)}$ | 88 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 89 | 1.8 V power supply for digital core | | | |
| P1[0]/EI0/ PMAT3[0]/A0 | 90[1] | GPIO 1, pin 0 | EXTINT0 | PWM3 MAT0 | EXTBUS A0 |
| P2[10]/ PMAT0[2]/ SCS0[0] | 91[1] | GPIO 2, pin 10 | $\overline{\text{USB\_INT}}$ | PWM0 MAT2 | SPI0 SCS0 |
| P2[11]/ PMAT0[3]/SCK0 | 92[1] | GPIO 2, pin 11 | $\overline{\text{USB\_RST}}$ | PWM0 MAT3 | SPI0 SCK |
| P0[0]/PHB0/ TXDC0/D24 | 93[1] | GPIO 0, pin 0 | QEI0 PHB | CAN0 TXD | EXTBUS D24 |
| $V_{SS(IO)}$ | 94 | ground for I/O | | | |
| P0[1]/PHA0/ RXDC0/D25 | 95[1] | GPIO 0, pin 1 | QEI 0 PHA | CAN0 RXD | EXTBUS D25 |
| P0[2]/CLK_OUT/ PMAT0[0]/D26 | 96[1] | GPIO 0, pin 2 | CLK_OUT | PWM0 MAT0 | EXTBUS D26 |
| P0[3]/$\overline{\text{USB\_UP\_LED}}$/ PMAT0[1]/D27 | 97[1] | GPIO 0, pin 3 | $\overline{\text{USB\_UP\_LED}}$ | PWM0 MAT1 | EXTBUS D27 |
| P3[0]/IN0[6]/ PMAT2[0]/$\overline{\text{CS6}}$ | 98[1] | GPIO 3, pin 0 | ADC0 IN6 | PWM2 MAT0 | EXTBUS $\overline{\text{CS6}}$ |
| P3[1]/IN0[7/ PMAT2[1]/$\overline{\text{CS7}}$ | 99[1] | GPIO 3, pin 1 | ADC0 IN7 | PWM2 MAT1 | EXTBUS $\overline{\text{CS7}}$ |
| P2[12]/IN0[4] PMAT0[4]/SDI0 | 100[1] | GPIO 2, pin 12 | ADC0 IN4 | PWM0 MAT4 | SPI0 SDI |
| P2[13]/IN0[5] PMAT0[5]/SDO0 | 101[1] | GPIO 2, pin 13 | ADC0 IN5 | PWM0 MAT5 | SPI0 SDO |
| P0[4]/IN0[0]/ PMAT0[2]/D28 | 102[1] | GPIO 0, pin 4 | ADC0 IN0 | PWM0 MAT2 | EXTBUS D28 |
| P0[5]/IN0[1]/ PMAT0[3]/D29 | 103[1] | GPIO 0, pin 5 | ADC0 IN1 | PWM0 MAT3 | EXTBUS D29 |
| $V_{DD(IO)}$ | 104 | 3.3 V power supply for I/O | | | |
| P0[6]/IN0[2]/ PMAT0[4]/D30 | 105[1] | GPIO 0, pin 6 | ADC0 IN2 | PWM0 MAT4 | EXTBUS D30 |

**Table 100. LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P0[7]/IN0[3]/ PMAT0[5]/D31 | 106[1] | GPIO 0, pin 7 | ADC0 IN3 | PWM0 MAT5 | EXTBUS D31 |
| $V_{DDA(ADC3V3)}$ | 107 | 3.3 V power supply for ADC | | | |
| JTAGSEL | 108[1] | TAP controller select input; LOW-level selects the ARM debug mode; HIGH-level selects boundary scan; pulled up internally. | | | |
| $V_{DDA(ADC5V0)}$ | 109 | 5 V supply voltage for ADC0 and 5 V reference for ADC0. | | | |
| VREFP | 110[3] | HIGH reference for ADC | | | |
| VREFN | 111[3] | LOW reference for ADC | | | |
| P0[8]/IN1[0]/TXDL0/ A20 | 112[4] | GPIO 0, pin 8 | ADC1 IN0 | LIN0 TXD/ UART TXD | EXTBUS A20 |
| P0[9]/IN1[1]/ RXDL0/A21 | 113[4] | GPIO 0, pin 9 | ADC1 IN1 | LIN0 RXD/ UART TXD | EXTBUS A21 |
| P0[10]/IN1[2]/ PMAT1[0]/A8 | 114[4] | GPIO 0, pin 10 | ADC1 IN2 | PWM1 MAT0 | EXTBUS A8 |
| P0[11]/IN1[3]/ PMAT1[1]/A9 | 115[4] | GPIO 0, pin 11 | ADC1 IN3 | PWM1 MAT1 | EXTBUS A9 |
| P2[14]/SDA1/ PCAP0[0]/$\overline{\text{BLS0}}$ | 116[1] | GPIO 2, pin 14 | I2C1 SDA | PWM0 CAP0 | EXTBUS $\overline{\text{BLS0}}$ |
| P2[15]/SCL1/ PCAP0[1]/$\overline{\text{BLS1}}$ | 117[1] | GPIO 2, pin 15 | I2C1 SCL | PWM0 CAP1 | EXTBUS $\overline{\text{BLS1}}$ |
| P3[2]/MAT3[0]/ PMAT2[2]/ USB_SDA | 118[1] | GPIO 3, pin 2 | TIMER3 MAT0 | PWM2 MAT2 | USB_SDA |
| $V_{SS(IO)}$ | 119 | ground for I/O | | | |
| P3[3]/MAT3[1]/ PMAT2[3]/ USB_SCL | 120[1] | GPIO 3, pin 3 | TIMER3 MAT1 | PWM2 MAT3 | USB_SCL |
| P0[12]/IN1[4]/ PMAT1[2]/A10 | 121[4] | GPIO 0, pin 12 | ADC1 IN4 | PWM1 MAT2 | EXTBUS A10 |
| P0[13]/IN1[5]/ PMAT1[3]/A11 | 122[4] | GPIO 0, pin 13 | ADC1 IN5 | PWM1 MAT3 | EXTBUS A11 |
| P0[14]/IN1[6]/ PMAT1[4]/A12 | 123[4] | GPIO 0, pin 14 | ADC1 IN6 | PWM1 MAT4 | EXTBUS A12 |
| P0[15]/IN1[7]/ PMAT1[5]/A13 | 124[4] | GPIO 0, pin 15 | ADC1 IN7 | PWM1 MAT5 | EXTBUS A13 |
| P0[16]IN2[0]/ TXD0/A22 | 125[4] | GPIO 0, pin 16 | ADC2 IN0 | UART0 TXD | EXTBUS A22 |
| P0[17]/IN2[1]/ RXD0/A23 | 126[4] | GPIO 0, pin 17 | ADC2 IN1 | UART0 RXD | EXTBUS A23 |
| $V_{DD(CORE)}$ | 127 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 128 | ground for digital core | | | |
| P2[16]/TXD1/ PCAP0[2]/$\overline{\text{BLS2}}$ | 129[1] | GPIO 2, pin 16 | UART1 TXD | PWM0 CAP2 | EXTBUS $\overline{\text{BLS2}}$ |

**Table 100. LQFP144 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P2[17]/RXD1/ PCAP1[0]/$\overline{\text{BLS3}}$ | 130[1] | GPIO 2, pin 17 | UART1 RXD | PWM1 CAP0 | EXTBUS $\overline{\text{BLS3}}$ |
| $V_{DD(IO)}$ | 131 | 3.3 V power supply for I/O | | | |
| P0[18]/IN2[2]/ PMAT2[0]/A14 | 132[4] | GPIO 0, pin 18 | ADC2 IN2 | PWM2 MAT0 | EXTBUS A14 |
| P0[19]/IN2[3]/ PMAT2[1]/A15 | 133[4] | GPIO 0, pin 19 | ADC2 IN3 | PWM2 MAT1 | EXTBUS A15 |
| P3[4]/MAT3[2]/ PMAT2[4]/TXDC1 | 134[1] | GPIO 3, pin 4 | TIMER3 MAT2 | PWM2 MAT4 | CAN1 TXD |
| P3[5]/MAT3[3]/ PMAT2[5]/RXDC1 | 135[1] | GPIO 3, pin 5 | TIMER3 MAT3 | PWM2 MAT5 | CAN1 RXD |
| P2[18]/SCS2[1]/ PCAP1[1]/D16 | 136[1] | GPIO 2, pin 18 | SPI2 SCS1 | PWM1 CAP1 | EXTBUS D16 |
| P2[19]/SCS2[0]/ PCAP1[2]/D17 | 137[1] | GPIO 2, pin 19 | SPI2 SCS0 | PWM1 CAP2 | EXTBUS D17 |
| P0[20]/IN2[4]/ PMAT2[2]/A16 | 138[4] | GPIO 0, pin 20 | ADC2 IN4 | PWM2 MAT2 | EXTBUS A16 |
| P0[21]/IN2[5]/ PMAT2[3]/A17 | 139[4] | GPIO 0, pin 21 | ADC2 IN5 | PWM2 MAT3 | EXTBUS A17 |
| P0[22]/IN2[6]/ PMAT2[4]/A18 | 140[4] | GPIO 0, pin 22 | ADC2 IN6 | PWM2 MAT4 | EXTBUS A18 |
| $V_{SS(IO)}$ | 141 | ground for I/O | | | |
| P0[23]/IN2[7]/ PMAT2[5]/A19 | 142[4] | GPIO 0, pin 23 | ADC2 IN7 | PWM2 MAT5 | EXTBUS A19 |
| P2[20]/ PCAP2[0]/D18 | 143[1] | GPIO 2, pin 20 | SPI2 SDO | PWM2 CAP0 | EXTBUS D18 |
| TDI | 144[1] | IEEE 1149.1 data in, pulled up internally | | | |

[1] Bidirectional pad; analog port; plain input; 3-state output; slew rate control; 5 V tolerant; TTL with hysteresis; programmable pull-up / pull-down / repeater.

[2] USB pad.

[3] Analog pad; analog I/O.

[4] Analog I/O pad.

## 5. LPC2930/30 pin configuration

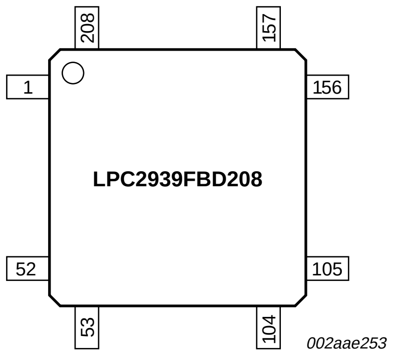


**Fig 30. Pin configuration for LQFP208 package**

The LPC2930/29 uses five ports: port 0 and port 1 with 32 pins, ports 2 with 28 pins each, port 3 with 16 pins, port 4 with 24 pins, and port 5 with 20 pins. The pin to which each function is assigned is controlled by the SFSP registers in the SCU. The functions combined on each port pin are shown in the pin description tables in this section.

**Table 101. LPC2930/39 LQFP208 pin assignment**

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| TDO | 1[1] | IEEE 1149.1 test data out | | | |
| P2[21]/SDI2/ PCAP2[1]/D19 | 2[1] | GPIO 2, pin 21 | SPI2 SDI | PWM2 CAP1 | EXTBUS D19 |
| P0[24]/TXD1/ TXDC1/SCS2[0] | 3[1] | GPIO 0, pin 24 | UART1 TXD | CAN1 TXD | SPI2 SCS0 |
| P0[25]/RXD1/ RXDC1/SDO2 | 4[1] | GPIO 0, pin 25 | UART1 RXD | CAN1 RXD | SPI2 SDO |
| P0[26]/TXD1/SDI2 | 5[1] | GPIO 0, pin 26 | - | UART1 TXD | SPI2 SDI |
| P0[27]/RXD1/SCK2 | 6[1] | GPIO 0, pin 27 | - | UART1 RXD | SPI2 SCK |
| P0[28]/CAP0[0]/ MAT0[0] | 7[1] | GPIO 0, pin 28 | - | TIMER0 CAP0 | TIMER0 MAT0 |
| P0[29]/CAP0[1]/ MAT0[1] | 8[1] | GPIO 0, pin 29 | - | TIMER0 CAP1 | TIMER0 MAT1 |
| $V_{DD(IO)}$ | 9 | 3.3 V power supply for I/O | | | |
| P2[22]/SCK2/ PCAP2[2]/D20 | 10[1] | GPIO 2, pin 22 | SPI2 SCK | PWM2 CAP2 | EXTBUS D20 |
| P2[23]/SCS1[0]/ PCAP3[0]/D21 | 11[1] | GPIO 2, pin 23 | SPI1 SCS0 | PWM3 CAP0 | EXTBUS D21 |
| P3[6]/SCS0[3]/ PMAT1[0]/TXDL1 | 12[1] | GPIO 3, pin 6 | SPI0 SCS3 | PWM1 MAT0 | LIN1/UART TXD |
| P3[7]/SCS2[1]/ PMAT1[1]/RXDL1 | 13[1] | GPIO 3, pin 7 | SPI2 SCS1 | PWM1 MAT1 | LIN1/UART RXD |
| P0[30]/CAP0[2]/ MAT0[2] | 14[1] | GPIO 0, pin 30 | - | TIMER0 CAP2 | TIMER0 MAT2 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P0[31]/CAP0[3]/ MAT0[3] | 15[1] | GPIO 0, pin 31 | - | TIMER0 CAP3 | TIMER0 MAT3 |
| P2[24]/SCS1[1]/ PCAP3[1]/D22 | 16[1] | GPIO 2, pin 24 | SPI1 SCS1 | PWM3 CAP1 | EXTBUS D22 |
| P2[25]/SCS1[2]/ PCAP3[2]/D23 | 17[1] | GPIO 2, pin 25 | SPI1 SCS2 | PWM3 CAP2 | EXTBUS D23 |
| $V_{SS(IO)}$ | 18 | ground for I/O | | | |
| P5[19]/USB_D+1 | 19[2] | GPIO 5, pin 19 | USB_D+1 | - | - |
| P5[18]/USB_D–1 | 20[2] | GPIO 5, pin 18 | USB_D–1 | - | - |
| P5[17]/USB_D+2 | 21[2] | GPIO 5, pin 17 | USB_D+2 | - | - |
| P5[16]/USB_D–2 | 22[2] | GPIO 5, pin 16 | USB_D–2 | - | - |
| $V_{DD(IO)}$ | 23 | 3.3 V power supply for I/O | | | |
| $V_{DD(CORE)}$ | 24 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 25 | ground for core | | | |
| P1[31]/CAP0[1]/ MAT0[1]/EI5 | 26[1] | GPIO 1, pin 31 | TIMER0 CAP1 | TIMER0 MAT1 | EXTINT5 |
| $V_{SS(IO)}$ | 27 | ground for I/O | | | |
| P4[0]/A8 | 28[1] | GPIO 4, pin 0 | EXTBUS A8 | - | - |
| P1[30]/CAP0[0]/ MAT0[0]/EI4 | 29[1] | GPIO 1, pin 30 | TIMER0 CAP0 | TIMER0 MAT0 | EXTINT4 |
| P5[0]/D8 | 30[1] | GPIO 5, pin 0 | EXTBUS D8 | - | - |
| P3[8]/SCS2[0]/ PMAT1[2]/ $\overline{\text{USB\_OVRCR1}}$ | 31[1] | GPIO 3, pin 8 | SPI2 SCS0 | PWM1 MAT2 | $\overline{\text{USB\_OVRCR1}}$ |
| P3[9]/SDO2/ PMAT1[3]/ $\overline{\text{USB\_PPWR1}}$ | 32[1] | GPIO 3, pin 9 | SPI2 SDO | PWM1 MAT3 | $\overline{\text{USB\_PPWR1}}$ |
| P1[29]/CAP1[0]/ TRAP0/ PMAT3[5] | 33[1] | GPIO 1, pin 29 | TIMER1 CAP0/ ADC0 EXTSTART | PWM TRAP0 | PWM3 MAT5 |
| $V_{DD(IO)}$ | 34 | 3.3 V power supply for I/O | | | |
| P4[16]/$\overline{\text{CS6}}$/ U1OUT1 | 35[1] | GPIO 4, pin 16 | EXTBUS $\overline{\text{CS6}}$ | UART1 OUT1 | - |
| P1[28]/CAP1[1]/ TRAP1/PMAT3[4] | 36[1] | GPIO 1, pin 28 | TIMER1 CAP1/ ADC1 EXTSTART | PWM TRAP1 | PWM3 MAT4 |
| P2[26]/CAP0[2]/ MAT0[2]/EI6 | 37[1] | GPIO 2, pin 26 | TIMER0 CAP2 | TIMER0 MAT2 | EXTINT6 |
| P4[8]/A22/DSR1 | 38 | GPIO 4, pin 8 | EXTBUS A22 | UART1 DSR | - |
| $V_{SS(IO)}$ | 39 | ground for I/O | | | |
| P2[27]/CAP0[3]/ MAT0[3]/EI7 | 40[1] | GPIO 2, pin 27 | TIMER0 CAP3 | TIMER0 MAT3 | EXTINT7 |
| P5[8]/D20/U0OUT2 | 41[1] | GPIO 5, pin 8 | EXTBUS D20 | UART0 OUT2 | - |
| P1[27]/CAP1[2]/ TRAP2/PMAT3[3] | 42[1] | GPIO 1, pin 27 | TIMER1 CAP2, ADC2 EXT START | PWM TRAP2 | PWM3 MAT3 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P1[26]/PMAT2[0]/ TRAP3/PMAT3[2] | 43[1] | GPIO 1, pin 26 | PWM2 MAT0 | PWM TRAP3 | PWM3 MAT2 |
| P4[20]/ USB_VBUS2 | 44[1] | GPIO4, pin 20 | USB_VBUS2 | | |
| $V_{DD(IO)}$ | 45 | 3.3 V power supply for I/O | | | |
| P1[25]/PMAT1[0]/ USB_VBUS1/ PMAT3[1] | 46[1] | GPIO 1, pin 25 | PWM1 MAT0 | USB_VBUS1 | PWM3 MAT1 |
| $V_{SS(CORE)}$ | 47 | ground for core | | | |
| $V_{DD(CORE)}$ | 48 | 1.8 V power supply for digital core | | | |
| P1[24]/PMAT0[0]/ $\overline{\text{USB\_CONNECT1}}$/ PMAT3[0] | 49[1] | GPIO 1, pin 24 | PWM0 MAT0 | $\overline{\text{USB\_CONNECT1}}$ | PWM3 MAT0 |
| P1[23]/RXD0/ $\overline{\text{USB\_SSPND1}}$/ $\overline{\text{CS5}}$ | 50[1] | GPIO 1, pin 23 | UART0 RXD | USB_SSPND1 | EXTBUS $\overline{\text{CS5}}$ |
| P1[22]/TXD0/ $\overline{\text{USB\_UP\_LED1}}$/$\overline{\text{CS4}}$ | 51[1] | GPIO 1, pin 22 | UART0 TXD | $\overline{\text{USB\_UP\_LED1}}$ | EXTBUS $\overline{\text{CS4}}$ |
| TMS | 52[1] | IEEE 1149.1 test mode select, pulled up internally | | | |
| TCK | 53[1] | IEEE 1149.1 test clock | | | |
| P1[21]/CAP3[3]/ CAP1[3]/D7 | 54[1] | GPIO 1, pin 21 | TIMER3 CAP3 | TIMER1 CAP3, MSCSS PAUSE | EXTBUS D7 |
| P1[20]/CAP3[2]/ SCS0[1]/D6 | 55[1] | GPIO 1, pin 20 | TIMER3 CAP2 | SPI0 SCS1 | EXTBUS D6 |
| P1[19]/CAP3[1]/ SCS0[2]/D5 | 56[1] | GPIO 1, pin 19 | TIMER3 CAP1 | SPI0 SCS2 | EXTBUS D5 |
| P1[18]/CAP3[0]/ SDO0/D4 | 57[1] | GPIO 1, pin 18 | TIMER3 CAP0 | SPI0 SDO | EXTBUS D4 |
| P1[17]/CAP2[3]/ SDI0/D3 | 58[1] | GPIO 1, pin 17 | TIMER2 CAP3 | SPI0 SDI | EXTBUS D3 |
| $V_{SS(IO)}$ | 59 | ground for I/O | | | |
| P4[4]/A12 | 60[1] | GPIO 4, pin 4 | EXTBUS A12 | - | - |
| P1[16]/CAP2[2]/ SCK0/D2 | 61[1] | GPIO 1, pin 16 | TIMER2 CAP2 | SPI0 SCK | EXTBUS D2 |
| P5[4]/D16 | 62[1] | GPIO 5, pin 4 | EXTBUS D16 | - | - |
| P2[0]/MAT2[0]/ TRAP3/D8 | 63[1] | GPIO 2, pin 0 | TIMER2 MAT0 | PWM TRAP3 | EXTBUS D8 |
| P4[12]/$\overline{\text{BLS0}}$ | 64[1] | GPIO 4, pin 12 | EXTBUS $\overline{\text{BLS0}}$ | - | - |
| P2[1]/MAT2[1]/ TRAP2/D9 | 65[1] | GPIO 2, pin 1 | TIMER2 MAT1 | PWM TRAP2 | EXTBUS D9 |
| P5[12]/D24 | 66[1] | GPIO 5, pin 12 | EXTBUS D24 | - | - |
| $V_{DD(IO)}$ | 67 | 3.3 V power supply for I/O | | | |
| P4[1]/A9 | 68[1] | GPIO 4, pin 1 | EXTBUS A9 | - | - |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| P3[10]/SDI2/ PMAT1[4]/ USB_PWRD1 | 69[1] | GPIO 3, pin 10 | SPI2 SDI | PWM1 MAT4 | USB_PWRD1 |
| $V_{SS(CORE)}$ | 70 | ground for core | | | |
| $V_{DD(CORE)}$ | 71 | 1.8 V power supply for digital core | | | |
| P5[1]/D9 | 72[1] | GPIO 5, pin 1 | EXTBUS D9 | - | - |
| P3[11]/SCK2/ PMAT1[5]/USB_LS1 | 73[1] | GPIO 3, pin 11 | SPI2 SCK | PWM1 MAT5 | USB_LS1 |
| P4[17]/CS7/U1OUT2 | 74[1] | GPIO 4, pin 17 | EXTBUS CS7 | UART1 OUT2 | - |
| P1[15]/CAP2[1]/ SCS0[0]/D1 | 75[1] | GPIO 1, pin 15 | TIMER2 CAP1 | SPI0 SCS0 | EXTBUS D1 |
| P4[9]/A23/DCD1 | 76[1] | GPIO4, pin 9 | EXTBUS A23 | UART1 DCD | - |
| $V_{SS(IO)}$ | 77 | ground for I/O | | | |
| P5[9]/D21/DTR0 | 78[1] | GPIO 5, pin 9 | EXTBUS D21 | UART0 DTR | - |
| P1[14]/CAP2[0]/SCS 0[3]/D0 | 79[1] | GPIO 1, pin 14 | TIMER2 CAP0 | SPI0 SCS3 | EXTBUS D0 |
| P4[21]/ USB_OVRCR2 | 80[1] | GPIO 4, pin 21 | USB_OVRCR2 | - | - |
| P1[13]/EI3/SCL1/WE | 81[1] | GPIO 1, pin 13 | EXTINT3 | I$^2$C1 SCL | EXTBUS WE |
| P4[5]/A13 | 82[1] | GPIO 4, pin 5 | EXTBUS A13 | - | - |
| P1[12]/EI2/SDA1/OE | 83[1] | GPIO 1, pin 12 | EXTINT2 | I$^2$C1 SDA | EXTBUS OE |
| P5[5]/D17 | 84[1] | GPIO 5, pin 5 | EXTBUS D17 | - | - |
| $V_{DD(IO)}$ | 85 | 3.3 V power supply for I/O | | | |
| P2[2]/MAT2[2]/ TRAP1/D10 | 86[1] | GPIO 2, pin 2 | TIMER2 MAT2 | PWM TRAP1 | EXTBUS D10 |
| P2[3]/MAT2[3]/ TRAP0/D11 | 87[1] | GPIO 2, pin 3 | TIMER2 MAT3 | PWM TRAP0 | EXTBUS D11 |
| P1[11]/SCK1/ SCL0/CS3 | 88[1] | GPIO 1, pin 11 | SPI1 SCK | I$^2$C0 SCL | EXTBUS CS3 |
| P1[10]/SDI1/ SDA0/CS2 | 89[1] | GPIO 1, pin 10 | SPI1 SDI | I$^2$C0 SDA | EXTBUS CS2 |
| P3[12]/SCS1[0]/EI4/ USB_SSPND1 | 90[1] | GPIO 3, pin 12 | SPI1 SCS0 | EXTINT4 | USB_SSPND1 |
| $V_{SS(CORE)}$ | 91 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 92 | 1.8 V power supply for digital core | | | |
| P3[13]/SDO1/ EI5/IDX0 | 93[1] | GPIO 3, pin 13 | SPI1 SDO | EXTINT5 | QEI0 IDX |
| P2[4]/MAT1[0]/ EI0/D12 | 94[1] | GPIO 2, pin 4 | TIMER1 MAT0 | EXTINT0 | EXTBUS D12 |
| P2[5]/MAT1[1]/ EI1/D13 | 95[1] | GPIO 2, pin 5 | TIMER1 MAT1 | EXTINT1 | EXTBUS D13 |
| P1[9]/SDO1/ RXDL1/CS1 | 96[1] | GPIO 1, pin 9 | SPI1 SDO | LIN1 RXD/UART RXD | EXTBUS CS1 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| $V_{SS(IO)}$ | 97 | ground for I/O | | | |
| P1[8]/SCS1[0]/ TXDL1/$\overline{\text{CS0}}$ | 98[1] | GPIO 1, pin 8 | SPI1 SCS0 | LIN1 TXD/ UART TXD | EXTBUS $\overline{\text{CS0}}$ |
| P1[7]/SCS1[3]/RXD1/ A7 | 99[1] | GPIO 1, pin 7 | SPI1 SCS3 | UART1 RXD | EXTBUS A7 |
| P1[6]/SCS1[2]/ TXD1/A6 | 100[1] | GPIO 1, pin 6 | SPI1 SCS2 | UART1 TXD | EXTBUS A6 |
| P2[6]/MAT1[2]/ EI2/D14 | 101[1] | GPIO 2, pin 6 | TIMER1 MAT2 | EXTINT2 | EXTBUS D14 |
| P1[5]/SCS1[1]/ PMAT3[5]/A5 | 102[1] | GPIO 1, pin 5 | SPI1 SCS1 | PWM3 MAT5 | EXTBUS A5 |
| P1[4]/SCS2[2]/ PMAT3[4]/A4 | 103[1] | GPIO 1, pin 4 | SPI2 SCS2 | PWM3 MAT4 | EXTBUS A4 |
| $\overline{\text{TRST}}$ | 104[1] | IEEE 1149.1 test reset NOT; active LOW; pulled up internally | | | |
| $\overline{\text{RST}}$ | 105[1] | asynchronous device reset; active LOW; pulled up internally | | | |
| $V_{SS(OSC)}$ | 106 | ground for oscillator | | | |
| XOUT_OSC | 107[3] | crystal out for oscillator | | | |
| XIN_OSC | 108[3] | crystal in for oscillator | | | |
| $V_{DD(OSC\_PLL)}$ | 109 | 1.8 V supply for oscillator and PLL | | | |
| $V_{SS(PLL)}$ | 110 | ground for PLL | | | |
| P2[7]/MAT1[3]/ EI3/D15 | 111[1] | GPIO 2, pin 7 | TIMER1 MAT3 | EXTINT3 | EXTBUS D15 |
| P3[14]/SDI1/ EI6/TXDC0 | 112[1] | GPIO 3, pin 14 | SPI1 SDI | EXTINT6 | CAN0 TXD |
| P3[15]/SCK1/ EI7/RXDC0 | 113[1] | GPIO 3, pin 15 | SPI1 SCK | EXTINT7 | CAN0 RXD |
| $V_{DD(IO)}$ | 114 | 3.3 V power supply for I/O | | | |
| P2[8]/CLK_OUT/ PMAT0[0]/SCS0[2] | 115[1] | GPIO 2, pin 8 | CLK_OUT | PWM0 MAT0 | SPI0 SCS2 |
| P2[9]/ $\overline{\text{USB\_UP\_LED1}}$/ PMAT0[1]/SCS0[1] | 116[1] | GPIO 2, pin 9 | $\overline{\text{USB\_UP\_LED1}}$ | PWM0 MAT1 | SPI0 SCS1 |
| P1[3]/SCS2[1]/ PMAT3[3]/A3 | 117[1] | GPIO 1, pin 3 | SPI2 SCS1 | PWM3 MAT3 | EXTBUS A3 |
| P1[2]/SCS2[3]/ PMAT3[2]/A2 | 118[1] | GPIO 1, pin 2 | SPI2 SCS3 | PWM3 MAT2 | EXTBUS A2 |
| P1[1]/EI1/PMAT3[1]/ A1 | 119[1] | GPIO 1, pin 1 | EXTINT1 | PWM3 MAT1 | EXTBUS A1 |
| $V_{SS(CORE)}$ | 120 | ground for digital core | | | |
| $V_{DD(CORE)}$ | 121 | 1.8 V power supply for digital core | | | |
| P1[0]/EI0/ PMAT3[0]/A0 | 122[1] | GPIO 1, pin 0 | EXTINT0 | PWM3 MAT0 | EXTBUS A0 |
| P2[10]/$\overline{\text{USB\_INT1}}$/ PMAT0[2]/SCS0[0] | 123[1] | GPIO 2, pin 10 | $\overline{\text{USB\_INT1}}$ | PWM0 MAT2 | SPI0 SCS0 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P2[11]/$\overline{\text{USB\_RST1}}$/ PMAT0[3]/SCK0 | 124[1] | GPIO 2, pin 11 | $\overline{\text{USB\_RST1}}$ | PWM0 MAT3 | SPI0 SCK |
| P0[0]/PHB0/ TXDC0/D24 | 125[1] | GPIO 0, pin 0 | QEI0 PHB | CAN0 TXD | EXTBUS D24 |
| $V_{SS(IO)}$ | 126 | ground for I/O | | | |
| P4[13]/$\overline{\text{BLS1}}$ | 127[1] | GPIO 4, pin 13 | EXTBUS $\overline{\text{BLS1}}$ | - | - |
| P0[1]/PHA0/ RXDC0/D25 | 128[1] | GPIO 0, pin 1 | QEI 0 PHA | CAN0 RXD | EXTBUS D25 |
| P5[13]/D25 | 129[1] | GPIO 5, pin 13 | EXTBUS D25 | - | - |
| P0[2]/CLK_OUT/ PMAT0[0]/D26 | 130[1] | GPIO 0, pin 2 | CLK_OUT | PWM0 MAT0 | EXTBUS D26 |
| P4[2]/A10 | 131[1] | GPIO 4, pin 2 | EXTBUS A10 | - | - |
| $V_{DD(IO)}$ | 132 | 3.3 V power supply for I/O | | | |
| P5[2]/D10 | 133[1] | GPIO 5, pin 2 | EXTBUS D10 | - | - |
| P0[3]/ $\overline{\text{USB\_UP\_LED1}}$/ PMAT0[1]/D27 | 134[1] | GPIO 0, pin 3 | $\overline{\text{USB\_UP\_LED1}}$ | PWM0 MAT1 | EXTBUS D27 |
| P4[18]/ $\overline{\text{USB\_UP\_LED2}}$ | 135[1] | GPIO 4, pin 18 | $\overline{\text{USB\_UP\_LED2}}$ | - | - |
| P3[0]/IN0[6]/ PMAT2[0]/$\overline{\text{CS6}}$ | 136[1] | GPIO 3, pin 0 | ADC0 IN6 | PWM2 MAT0 | EXTBUS $\overline{\text{CS6}}$ |
| P4[10]/OE/CTS1 | 137[1] | GPIO 4, pin 10 | EXTBUS $\overline{\text{OE}}$ | UART1 CTS | - |
| P3[1]/IN0[7/ PMAT2[1]/$\overline{\text{CS7}}$ | 138[1] | GPIO 3, pin 1 | ADC0 IN7 | PWM2 MAT1 | EXTBUS $\overline{\text{CS7}}$ |
| P5[10]/D22/DSR0 | 139[1] | GPIO 5, pin 10 | EXTBUS D22 | UART0 DSR | - |
| P2[12]/IN0[4] PMAT0[4]/SDI0 | 140[1] | GPIO 2, pin 12 | ADC0 IN4 | PWM0 MAT4 | SPI0 SDI |
| $V_{DD(CORE)}$ | 141 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 142 | ground for digital core | | | |
| P4[22]/ $\overline{\text{USB\_PPWR2}}$ | 143[1] | GPIO 4, pin 22 | $\overline{\text{USB\_PPWR2}}$ | - | - |
| $V_{SS(IO)}$ | 144 | ground for I/O | | | |
| P2[13]/IN0[5]/ PMAT0[5]/SDO0 | 145[1] | GPIO 2, pin 13 | ADC0 IN5 | PWM0 MAT5 | SPI0 SDO |
| P4[6]/A20/RI1 | 146[1] | GPIO 4, pin 6 | EXTBUS A20 | UART1 RI1 | - |
| P0[4]/IN0[0]/ PMAT0[2]/D28 | 147[1] | GPIO 0, pin 4 | ADC0 IN0 | PWM0 MAT2 | EXTBUS D28 |
| P5[6]/D18/RI0 | 148[1] | GPIO 5, pin 6 | EXTBUS D18 | UART0 RI | - |
| P4[14]/$\overline{\text{BLS2}}$ | 149[1] | GPIO 4, pin 14 | EXTBUS $\overline{\text{BLS2}}$ | - | - |
| P0[5]/IN0[1]/ PMAT0[3]/D29 | 150[1] | GPIO 0, pin 5 | ADC0 IN1 | PWM0 MAT3 | EXTBUS D29 |
| P5[14]/ USB_SSPND1/RTS0 | 151[1] | GPIO 5, pin 14 | USB_SSPND1 | UART0 RTS | - |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| $V_{DD(IO)}$ | 152 | 3.3 V power supply for I/O | | | |
| P0[6]/IN0[2]/ PMAT0[4]/D30 | 153[1] | GPIO 0, pin 6 | ADC0 IN2 | PWM0 MAT4 | EXTBUS D30 |
| P0[7]/IN0[3]/ PMAT0[5]/D31 | 154[1] | GPIO 0, pin 7 | ADC0 IN3 | PWM0 MAT5 | EXTBUS D31 |
| $V_{DDA(ADC3V3)}$ | 155 | 3.3 V power supply for ADC | | | |
| JTAGSEL | 156[1] | TAP controller select input; LOW-level selects the ARM debug mode; HIGH-level selects boundary scan; pulled up internally. | | | |
| $V_{DDA(ADC5V0)}$ | 157 | 5 V supply voltage for ADC0 and 5 V reference for ADC0. | | | |
| VREFP | 158[3] | HIGH reference for ADC | | | |
| VREFN | 159[3] | LOW reference for ADC | | | |
| P0[8]/IN1[0]/TXDL0/ A20 | 160[4] | GPIO 0, pin 8 | ADC1 IN0 | LIN0 TXD/ UART TXD | EXTBUS A20 |
| P0[9]/IN1[1]/ RXDL0/A21 | 161[4] | GPIO 0, pin 9 | ADC1 IN1 | LIN0 RXD/ UART TXD | EXTBUS A21 |
| P0[10]/IN1[2]/ PMAT1[0]/A8 | 162[4] | GPIO 0, pin 10 | ADC1 IN2 | PWM1 MAT0 | EXTBUS A8 |
| P0[11]/IN1[3]/ PMAT1[1]/A9 | 163[4] | GPIO 0, pin 11 | ADC1 IN3 | PWM1 MAT1 | EXTBUS A9 |
| P2[14]/SDA1/ PCAP0[0]/$\overline{\text{BLS0}}$ | 164[1] | GPIO 2, pin 14 | I$^2$C1 SDA | PWM0 CAP0 | EXTBUS $\overline{\text{BLS0}}$ |
| P2[15]/SCL1/ PCAP0[1]/$\overline{\text{BLS1}}$ | 165[1] | GPIO 2, pin 15 | I$^2$C1 SCL | PWM0 CAP1 | EXTBUS $\overline{\text{BLS1}}$ |
| P3[2]/MAT3[0]/ PMAT2[2]/ USB_SDA1 | 166[1] | GPIO 3, pin 2 | TIMER3 MAT0 | PWM2 MAT2 | USB_SDA1 |
| $V_{DD(CORE)}$ | 167 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 168 | ground for digital core | | | |
| $V_{SS(IO)}$ | 169 | ground for I/O | | | |
| P4[3]/A11 | 170[1] | GPIO 4, pin 3 | EXTBUS A11 | - | - |
| P3[3]/MAT3[1]/ PMAT2[3]/ USB_SCL1 | 171[1] | GPIO 3, pin 3 | TIMER3 MAT1 | PWM2 MAT3 | USB_SCL1 |
| P5[3]/D11 | 172[1] | GPIO 5, pin 3 | EXTBUS D11 | - | - |
| P0[12]/IN1[4]/ PMAT1[2]/A10 | 173[4] | GPIO 0, pin 12 | ADC1 IN4 | PWM1 MAT2 | EXTBUS A10 |
| P4[19]/ $\overline{\text{USB\_CONNECT2}}$ | 174[1] | GPIO 4, pin 19 | $\overline{\text{USB\_CONNECT2}}$ | - | - |
| P0[13]/IN1[5]/ PMAT1[3]/A11 | 175[4] | GPIO 0, pin 13 | ADC1 IN5 | PWM1 MAT3 | EXTBUS A11 |
| $V_{DD(IO)}$ | 176 | 3.3 V power supply for I/O | | | |
| P4[11]/$\overline{\text{WE}}$/CTS0 | 177[1] | GPIO 4, pin 11 | EXTBUS $\overline{\text{WE}}$ | UART0 CTS | - |
| P0[14]/IN1[6]/ PMAT1[4]/A12 | 178[4] | GPIO 0, pin 14 | ADC1 IN6 | PWM1 MAT4 | EXTBUS A12 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | **Function 0 (default)** | **Function 1** | **Function 2** | **Function 3** |
| P5[11]/D23/DCD0 | 179[1] | GPIO 5, pin 11 | EXTBUS D23 | UART0 DCD | - |
| P0[15]/IN1[7]/ PMAT1[5]/A13 | 180[4] | GPIO 0, pin 15 | ADC1 IN7 | PWM1 MAT5 | EXTBUS A13 |
| P4[23]/ $\overline{\text{USB\_PWRD2}}$ | 181[1] | GPIO 4, pin 23 | $\overline{\text{USB\_PWRD2}}$ | - | - |
| P0[16]IN2[0]/ TXD0/A22 | 182[4] | GPIO 0, pin 16 | ADC2 IN0 | UART0 TXD | EXTBUS A22 |
| P4[7]/A21/DTR1 | 183[1] | GPIO 4, pin 7 | EXTBUS A21 | UART1 DTR | - |
| $V_{SS(IO)}$ | 184 | ground for I/O | | | |
| P5[7]/D19/ U0OUT1 | 185[1] | GPIO 5, pin 7 | EXTBUS D19 | UART0 OUT1 | - |
| P0[17]/IN2[1]/ RXD0/A23 | 186[4] | GPIO 0, pin 17 | ADC2 IN1 | UART0 RXD | EXTBUS A23 |
| P4[15]/$\overline{\text{BLS3}}$ | 187[1] | GPIO 4, pin 15 | EXTBUS $\overline{\text{BLS3}}$ | - | - |
| P5[15]/ $\overline{\text{USB\_UP\_LED1}}$/ RTS1 | 188[1] | GPIO 5, pin 15 | $\overline{\text{USB\_UP\_LED1}}$ | UART1 RTS | - |
| $V_{DD(CORE)}$ | 189 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 190 | ground for digital core | | | |
| P2[16]/TXD1/ PCAP0[2]/$\overline{\text{BLS2}}$ | 191[1] | GPIO 2, pin 16 | UART1 TXD | PWM0 CAP2 | EXTBUS $\overline{\text{BLS2}}$ |
| P2[17]/RXD1/ PCAP1[0]/$\overline{\text{BLS3}}$ | 192[1] | GPIO 2, pin 17 | UART1 RXD | PWM1 CAP0 | EXTBUS $\overline{\text{BLS3}}$ |
| $V_{DD(IO)}$ | 193 | 3.3 V power supply for I/O | | | |
| P0[18]/IN2[2]/ PMAT2[0]/A14 | 194[4] | GPIO 0, pin 18 | ADC2 IN2 | PWM2 MAT0 | EXTBUS A14 |
| P0[19]/IN2[3]/ PMAT2[1]/A15 | 195[4] | GPIO 0, pin 19 | ADC2 IN3 | PWM2 MAT1 | EXTBUS A15 |
| P3[4]/MAT3[2]/ PMAT2[4]/ TXDC1 | 196[1] | GPIO 3, pin 4 | TIMER3 MAT2 | PWM2 MAT4 | CAN1 TXD |
| P3[5]/MAT3[3]/ PMAT2[5]/ RXDC1 | 197[1] | GPIO 3, pin 5 | TIMER3 MAT3 | PWM2 MAT5 | CAN1 RXD |
| P2[18]/SCS2[1]/ PCAP1[1]/D16 | 198[1] | GPIO 2, pin 18 | SPI2 SCS1 | PWM1 CAP1 | EXTBUS D16 |
| P2[19]/SCS2[0]/ PCAP1[2]/D17 | 199[1] | GPIO 2, pin 19 | SPI2 SCS0 | PWM1 CAP2 | EXTBUS D17 |
| P0[20]/IN2[4]/ PMAT2[2]/A16 | 200[4] | GPIO 0, pin 20 | ADC2 IN4 | PWM2 MAT2 | EXTBUS A16 |
| P0[21]/IN2[5]/ PMAT2[3]/A17 | 201[4] | GPIO 0, pin 21 | ADC2 IN5 | PWM2 MAT3 | EXTBUS A17 |
| P0[22]/IN2[6]/ PMAT2[4]/A18 | 202[4] | GPIO 0, pin 22 | ADC2 IN6 | PWM2 MAT4 | EXTBUS A18 |

**Table 101. LPC2930/39 LQFP208 pin assignment** *…continued*

| Pin name | Pin | Description | | | |
|---|---|---|---|---|---|
| | | Function 0 (default) | Function 1 | Function 2 | Function 3 |
| $V_{SS(IO)}$ | 203 | ground for I/O | | | |
| P0[23]/IN2[7]/ PMAT2[5]/A19 | 204[4] | GPIO 0, pin 23 | ADC2 IN7 | PWM2 MAT5 | EXTBUS A19 |
| P2[20]/ PCAP2[0]/D18 | 205[1] | GPIO 2, pin 20 | SPI2 SDO | PWM2 CAP0 | EXTBUS D18 |
| $V_{DD(CORE)}$ | 206 | 1.8 V power supply for digital core | | | |
| $V_{SS(CORE)}$ | 207 | ground for digital core | | | |
| TDI | 208[1] | IEEE 1149.1 data in, pulled up internally | | | |

[1] Bidirectional pad; analog port; plain input; 3-state output; slew rate control; 5V tolerant; TTL with hysteresis; programmable pull-up / pull-down / repeater.

[2] USB pad.

[3] Analog pad; Analog I/O.

[4] Boot control pin (LPC2930 only).

[5] Analog I/O pad.

## 6. LPC2930 boot control

The flashless LPC2930 uses pins P2[7]/D15 and P2[6]/D14 to configure the external memory bus during the boot process. These pins are sampled during POR. See Table 11–102 for possible memory configurations.

**Table 102. Boot control pins P2[7]/D15 and P2[6]/D14**

| P2[7]/D15 (BOOT1) | P2[6]/D14 (BOOT0) | Description |
|---|---|---|
| 0 | 0 | Boot from 8-bit external memory on $\overline{CS7}$. |
| 0 | 1 | Reserved. |
| 1 | 0 | Boot from 32-bit external memory on $\overline{CS7}$. |
| 1 | 1 | Boot from 16-bit external memory on $\overline{CS7}$. |

The start-up code residing in the external memory must be linked to execute from address location 0x5C00 0000 ($\overline{CS7}$) if the TCM is enabled (see Section 2–3.1). If TCM is **not** enabled, executing from address 0x0000 0000 is also possible.

**Remark:** During the boot process all address lines A23 to A0 are configured as digital output. Do not drive any of the address lines as input even if they are not used.

See Section 12–3 for connecting 8-bit (Figure 12–37), 16-bit (Figure 12–36), or 32-bit (Figure 12–34) external memory devices to the external memory controller. In all cases connect $\overline{CS7}$ to the CE input of the memory device.

For booting from external memory, connect the pins as shown in Table 11–103 to the external memory device.

**Table 103. LPC2930 boot configuration**

| LPC2930 connections | | | External memory connections | | |
|---|---|---|---|---|---|
| **Pin** | **Port** | **Function** | **8-bit** | **16-bit** | **32- bit** |
| 137 | P4[10] | $\overline{\text{OE}}$ | $\overline{\text{OE}}$ | $\overline{\text{OE}}$ | $\overline{\text{OE}}$ |
| 177 | P4[11] | $\overline{\text{WE}}$ | - | $\overline{\text{WE}}$ | $\overline{\text{WE}}$ |
| 74 | P4[17] | $\overline{\text{CS7}}$ | $\overline{\text{CS}}$ | $\overline{\text{CS}}$ | $\overline{\text{CS}}$ |
| 64 | P4[12] | $\overline{\text{BLS0}}$ | $\overline{\text{WE}}$ | $\overline{\text{BLS0}}$ | $\overline{\text{BLS0}}$ |
| 127 | P4[13] | $\overline{\text{BLS1}}$ | - | $\overline{\text{BLS1}}$ | $\overline{\text{BLS1}}$ |
| 149 | P4[14] | $\overline{\text{BLS2}}$ | - | - | $\overline{\text{BLS2}}$ |
| 187 | P4[15] | $\overline{\text{BLS3}}$ | - | - | $\overline{\text{BLS3}}$ |
| 122 | P1[0] | A0 | A0 | - | - |
| 119 | P1[1] | A1 | A1 | A0 | - |
| 118 | P1[2] | A2 | A2 | A1 | A0 |
| 117 | P1[3] | A3 | A3 | A2 | A1 |
| 103 | P1[4] | A4 | A4 | A3 | A2 |
| 102 | P1[5] | A5 | A5 | A4 | A3 |
| 100 | P1[6] | A6 | A6 | A5 | A4 |
| 99 | P1[7] | A7 | A7 | A6 | A5 |
| 28 | P4[0] | A8 | A8 | A7 | A6 |
| 68 | P4[1] | A9 | A9 | A8 | A7 |
| 131 | P4[2] | A10 | A10 | A9 | A8 |
| 170 | P4[3] | A11 | A11 | A10 | A9 |
| 60 | P4[4] | A12 | A12 | A11 | A10 |
| 82 | P4[5] | A13 | A13 | A12 | A11 |
| 194 | P0[18] | A14 | A14 | A13 | A12 |
| 195 | P0[19] | A15 | A15 | A14 | A13 |
| 200 | P0[20] | A16 | A16 | A15 | A14 |
| 201 | P0[21] | A17 | A17 | A16 | A15 |
| 202 | P0[22] | A18 | A18 | A17 | A16 |
| 204 | P0[23] | A19 | A19 | A18 | A17 |
| 146 | P4[6] | A20 | A20 | A19 | A18 |
| 183 | P4[7] | A21 | A21 | A20 | A19 |
| 38 | P4[8] | A22 | A22 | A21 | A20 |
| 76 | P4[9] | A23 | A23 | A22 | A21 |
| 79 | P1[14] | D0 | D0 | D0 | D0 |
| 75 | P1[15] | D1 | D1 | D1 | D1 |
| 61 | P1[16] | D2 | D2 | D2 | D2 |
| 58 | P1[17] | D3 | D3 | D3 | D3 |
| 57 | P1[18] | D4 | D4 | D4 | D4 |
| 56 | P1[19] | D5 | D5 | D5 | D5 |
| 55 | P1[20] | D6 | D6 | D6 | D6 |

Table 103. LPC2930 boot configuration

| LPC2930 connections | | | External memory connections | | |
|---|---|---|---|---|---|
| Pin | Port | Function | 8-bit | 16-bit | 32- bit |
| 54 | P1[21] | D7 | D7 | D7 | D7 |
| 30 | P5[0] | D8 | D8 | D8 | D8 |
| 72 | P5[1] | D9 | - | D9 | D9 |
| 133 | P5[2] | D10 | - | D10 | D10 |
| 172 | P5[3] | D11 | - | D11 | D11 |
| 94 | P2[4] | D12 | - | D12 | D12 |
| 95 | P2[5] | D13 | - | D13 | D13 |
| 101 | P2[6] | D14[1] | - | D14 | D14 |
| 111 | P2[7] | D15[1] | - | D15 | D15 |
| 62 | P5[4] | D16 | - | - | D16 |
| 84 | P5[5] | D17 | - | - | D17 |
| 148 | P5[6 | D18 | - | - | D18 |
| 185 | P5[7] | D19 | - | - | D19 |
| 41 | P5[8] | D20 | - | - | D20 |
| 78 | P5[9] | D21 | - | - | D21 |
| 139 | P5[10] | D22 | - | - | D22 |
| 179 | P5[11] | D23 | - | - | D23 |
| 66 | P5[12] | D24 | - | - | D24 |
| 129 | P5[13] | D25 | - | - | D25 |
| 130 | P0[2] | D26 | - | - | D26 |
| 134 | P0[3] | D27 | - | - | D27 |
| 147 | P0[4] | D28 | - | - | D28 |
| 150 | P0[5] | D29 | - | - | D29 |
| 153 | P0[6] | D30 | - | - | D30 |
| 154 | P0[7] | D31 | - | - | D31 |

[1] Boot pins control pins. See Table 11–102 for configuration of the boot control pins.

# UM10316

# Chapter 12: LPC29xx external Static Memory Controller (SMC)

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The contents of this chapter apply to parts LPC2917/19/01, LPC2926/27/29, LPC2930, and LPC2939. The LPC2921/23/25 do not have an external static memory controller.

The flashless LPC2930 uses the external SMC for booting (see Section 11–6).

## 2. SMC functional description

External memory can be connected to the device. The Static Memory Controller (SMC) controls timing and configuration of this external memory.

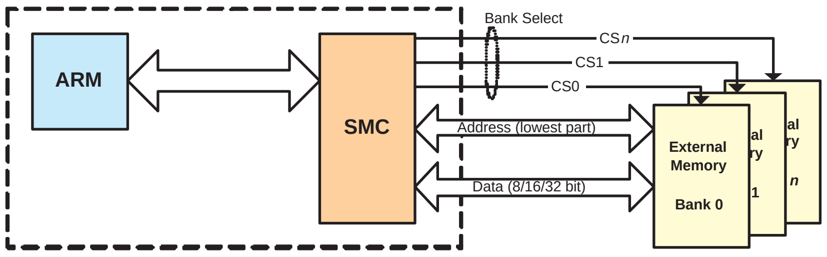


Fig 31. Schematic representation of the SMC

The SMC provides an interface between a system bus and external (off-chip) memory devices. It provides support for up to eight independently configurable memory banks simultaneously. Each memory bank is capable of supporting SRAM, ROM, Flash EPROM, Burst ROM memory or external I/O devices (memory-mapped).

Each memory bank may be 8, 16, or 32 bits wide.

Table 104. Static-memory bank address range

| Bank | Address Range | |
|---|---|---|
| 0 | 0x4000 0000 | 0x43FF FFFF |
| 1 | 0x4400 0000 | 0x47FF FFFF |
| 2 | 0x4800 0000 | 0x4BFF FFFF |
| 3 | 0x4C00 0000 | 0x4FFF FFFF |
| 4 | 0x5000 0000 | 0x53FF FFFF |
| 5 | 0x5400 0000 | 0x57FF FFFF |
| 6 | 0x5800 0000 | 0x5BFF FFFF |
| 7 | 0x5C00 0000 | 0x5FFF FFFF |

Memory banks can be set to write-protect state. In this case the memory controller blocks write access for the specified bank. When an illegal write occurs the WRITEPROTERR bit in the SMBSR register is set.

## 3. External memory interface

The external memory interface depends on the bank width: 32, 16 or 8 bits selected via MW bits in the corresponding SMBCR register. Choice of memory chips requires an adequate set-up of the RBLE bit in the same register. RBLE = 0 for 8-bit based external memories, while memory chips capable of accepting 16- or 32-bit wide data will work with RBLE = 1. If a memory bank is configured to be 32 bits wide, address lines A0 and A1 can be used as non-address lines. Memory banks configured to 16 bits wide do not require A0, while 8-bit wide memory banks require address lines down to A0.

Configuring A1 and/or A0 line(s) to provide address or non-address function is accomplished by setting up the SCU. Symbol A[x] refers to the highest-order address line of the memory chip used in the external-memory interface. CS refers to the eight bank-select lines, and BLS refers to the four byte-lane select lines. WE_N is the write output enable and OE_N is the output enable. Address pins on the device are shared with other functions. When connecting external memories, check that the I/O pin is programmed to the correct function. Control of these settings is handled by the SCU (see Section 6–2).

Figure 12–32 shows configuration of a 32-bit wide memory bank using 8-bit devices. Figure 12–33 and Figure 12–34 show a 32-bit wide memory using 16- and 32-bit devices. Figure 12–35 shows configuration of a 16-bit wide memory bank using 8-bit devices. Figure 12–36 shows configuration of a 16-bit wide memory bank using 16-bit devices. Figure 12–37 shows an 8-bit wide memory bank. This memory width requires 8-bit devices.

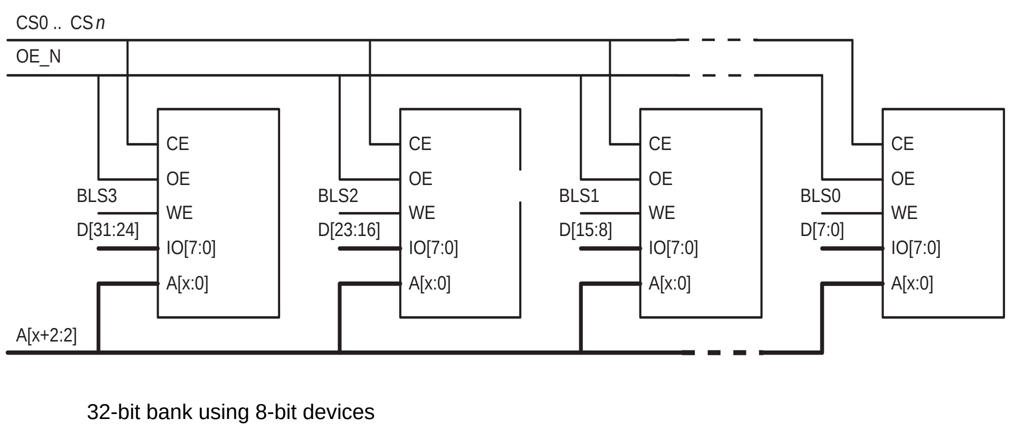
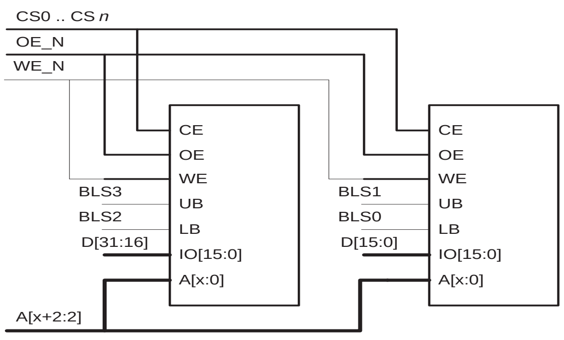
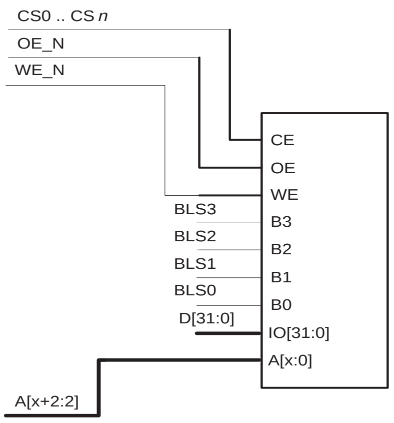


**Fig 32. External memory interface: 32-bit banks with 8-bit devices**



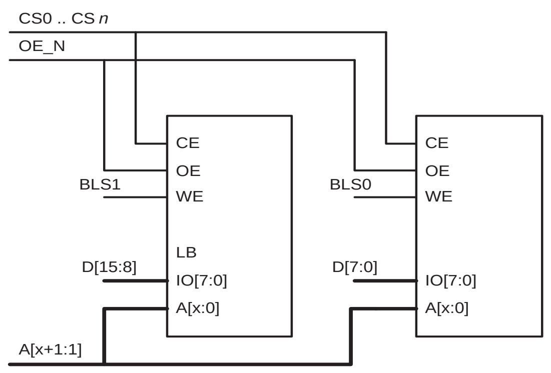
32-bit bank using 16-bit devices

**Fig 33. External memory interface: 32-bit banks with 16-bit devices**



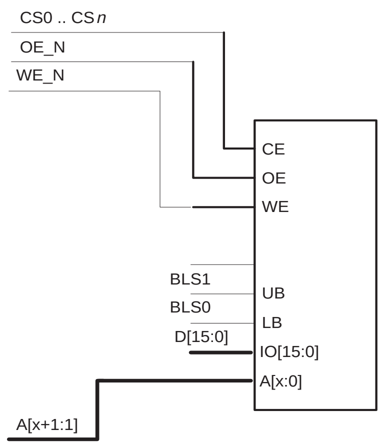
32-bit bank using 32-bit device

**Fig 34. External memory interface: 32-bit banks with 32-bit devices**



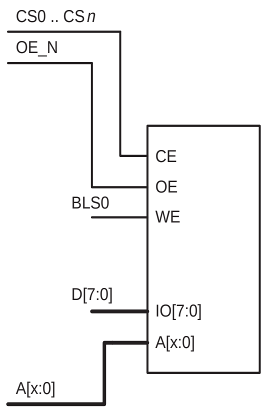
16-bit bank using 8-bit devices

**Fig 35. External memory interface: 16-bit banks with 8-bit devices**



16-bit bank using 16-bit device

**Fig 36. External memory interface: 16-bit banks with 16-bit devices**

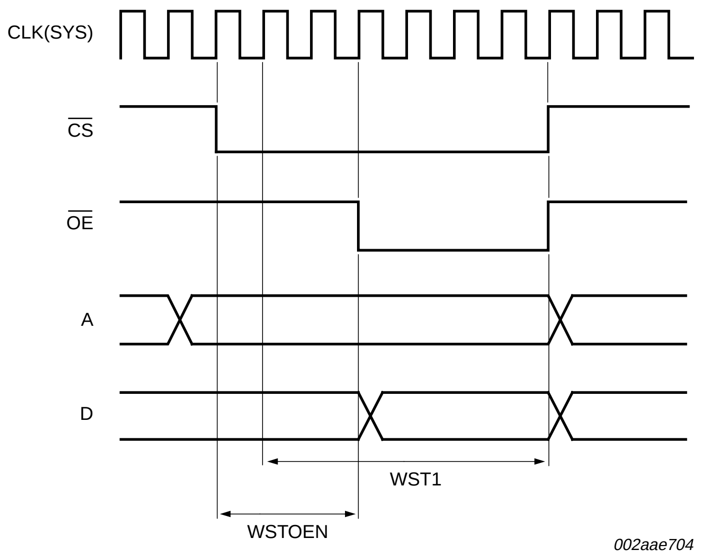


8-bit bank using 8-bit device

**Fig 37. External memory interface: 8-bit banks with 8-bit devices**

Memory is available in various speeds, so the numbers of wait-states for both read and write access must be set up. These settings should be reconsidered when the ARM processor-core clock changes.

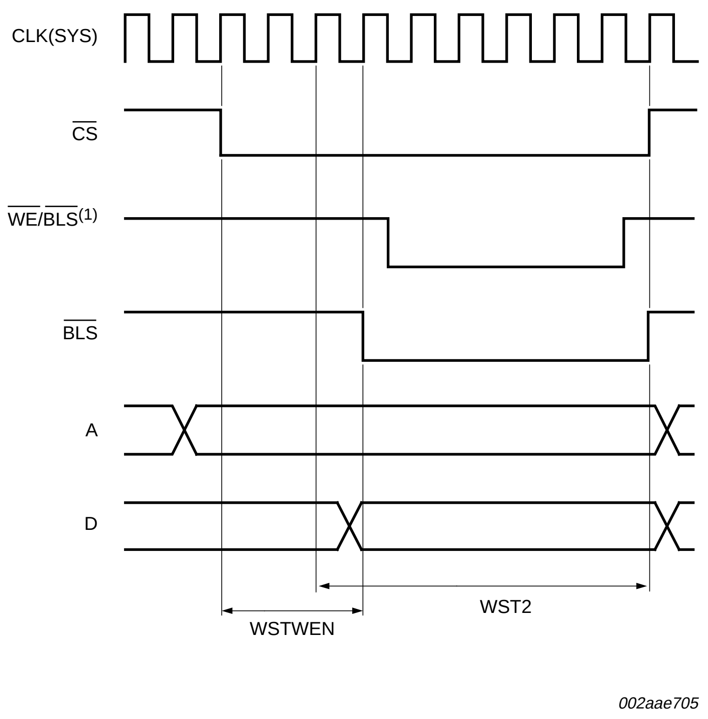
In Figure 12–38 a timing diagram for reading external memory is shown. The relationship between the wait-state settings is indicated with arrows.



WSTOEN=3, WST1=6

**Fig 38. Reading from external memory**

In Figure 12–39 a timing diagram for writing external memory is shown. The relationship between wait-state settings is indicated with arrows.

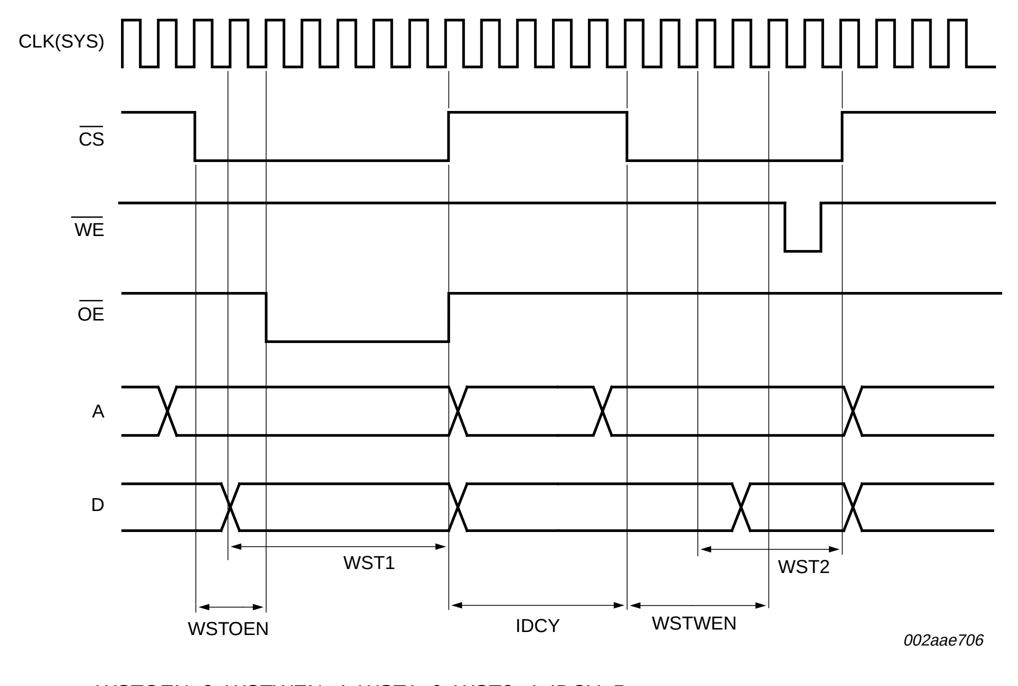


WSTWEN=3, WST2=7

(1) $\overline{BLS}$ has the same timing as $\overline{WE}$ in configurations that use the byte lane enable signals to connect to write enable (8 bit devices).

**Fig 39. Writing to external memory**

In Figure 12–40 usage of the idle/turn-around time (IDCY) is demonstrated. Extra wait-states are added between a read and a write cycle in the same external memory device.



WSTOEN=2, WSTWEN=4, WST1=6, WST2=4, IDCY=5

**Fig 40. Reading/writing external memory**

Address pins on the device are shared with other functions. When connecting external memories, check that the I/O pin is programmed to the correct function. Control of these settings is handled by the SCU.

## 4. Register overview

**Table 105. Register overview: external SMC (base address 0x6000 0000)**

| Symbol | Access | Offset Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| **Bank 0** | | | | | |
| SMBIDCYR0 | R/W | 0x000 | Idle-cycle control register for memory bank 0 | 0xF | Table 12–106 |
| SMBWST1R0 | R/W | 0x004 | Wait-state 1 control register for memory bank 0 | 0x1F | Table 12–107 |
| SMBWST2R0 | R/W | 0x008 | Wait-state 2 control register for memory bank 0 | 0x1F | Table 12–108 |
| SMBWSTOENR0 | R/W | 0x00C | Output-enable assertion delay control register for memory bank 0 | 0x0 | Table 12–109 |
| SMBWSTWENR0 | R/W | 0x010 | Write-enable assertion delay control register for memory bank 0 | 0x1 | Table 12–110 |
| SMBCR0 | R/W | 0x014 | Configuration register for memory bank 0 | 0x80 | Table 12–111 |
| SMBSR0 | R/W | 0x018 | Status register for memory bank 0 | 0x0 | Table 12–112 |
| **Bank 1** | | | | | |
| SMBIDCYR1 | R/W | 0x01C | Idle-cycle control register for memory bank 1 | 0xF | Table 12–106 |
| SMBWST1R1 | R/W | 0x020 | Wait-state 1 control register for memory bank 1 | 0x1F | Table 12–107 |
| SMBWST2R1 | R/W | 0x024 | Wait-state 2 control register for memory bank 1 | 0x1F | Table 12–108 |

**Table 105. Register overview: external SMC** *…continued*(base address 0x6000 0000)

| Symbol | Access | Offset Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SMBWSTOENR1 | R/W | 0x028 | Output-enable assertion delay control register for memory bank 1 | 0x0 | Table 12–109 |
| SMBWSTWENR1 | R/W | 0x02C | Write-enable assertion delay control register for memory bank 1 | 0x1 | Table 12–110 |
| SMBCR1 | R/W | 0x030 | Configuration register for memory bank 1 | 0x00 | Table 12–111 |
| SMBSR1 | R/W | 0x034 | Status register for memory bank 1 | 0x0 | Table 12–112 |
| **Bank 2** | | | | | |
| SMBIDCYR2 | R/W | 0x038 | Idle-cycle control register for memory bank 2 | 0xF | Table 12–106 |
| SMBWST1R2 | R/W | 0x03C | Wait-state 1 control register for memory bank 2 | 0x1F | Table 12–107 |
| SMBWST2R2 | R/W | 0x040 | Wait-state 2 control register for memory bank 2 | 0x1F | Table 12–108 |
| SMBWSTOENR2 | R/W | 0x044 | Output-enable assertion delay control register for memory bank 2 | 0x0 | Table 12–109 |
| SMBWSTWENR2 | R/W | 0x048 | Write-enable assertion delay control register for memory bank 2 | 0x1 | Table 12–110 |
| SMBCR2 | R/W | 0x04C | Configuration register for memory bank 2 | 0x40 | Table 12–111 |
| SMBSR2 | R/W | 0x050 | Status register for memory bank 2 | 0x0 | Table 12–112 |
| **Bank 3** | | | | | |
| SMBIDCYR3 | R/W | 0x054 | Idle-cycle control register for memory bank 3 | 0xF | Table 12–106 |
| SMBWST1R3 | R/W | 0x058 | Wait-state 1 control register for memory bank 3 | 0x1F | Table 12–107 |
| SMBWST2R3 | R/W | 0x05C | Wait-state 2 control register for memory bank 3 | 0x1F | Table 12–108 |
| SMBWSTOENR3 | R/W | 0x060 | Output-enable assertion delay control register for memory bank 3 | 0x0 | Table 12–109 |
| SMBWSTWENR3 | R/W | 0x064 | Write-enable assertion delay control register for memory bank 3 | 0x1 | Table 12–110 |
| SMBCR3 | R/W | 0x068 | Configuration register for memory bank 3 | 0x00 | Table 12–111 |
| SMBSR3 | R/W | 0x06C | Status register for memory bank 3 | 0x0 | Table 12–112 |
| **Bank 4** | | | | | |
| SMBIDCYR4 | R/W | 0x070 | Idle-cycle control register for memory bank 4 | 0xF | Table 12–106 |
| SMBWST1R4 | R/W | 0x074 | Wait-state 1 control register for memory bank 4 | 0x1F | Table 12–107 |
| SMBWST2R4 | R/W | 0x078 | Wait-state 2 control register for memory bank 4 | 0x1F | Table 12–108 |
| SMBWSTOENR4 | R/W | 0x07C | Output-enable assertion delay control register for memory bank 4 | 0x0 | Table 12–109 |
| SMBWSTWENR4 | R/W | 0x080 | Write-enable assertion delay control register for memory bank 4 | 0x1 | Table 12–110 |
| SMBCR4 | R/W | 0x084 | Configuration register for memory bank 4 | 0x80 | Table 12–111 |
| SMBSR4 | R/W | 0x088 | Status register for memory bank 4 | 0x0 | Table 12–112 |
| **Bank 5** | | | | | |
| SMBIDCYR5 | R/W | 0x08C | Idle-cycle control register for memory bank 5 | 0xF | Table 12–106 |
| SMBWST1R5 | R/W | 0x090 | Wait-state 1 control register for memory bank 5 | 0x1F | Table 12–107 |
| SMBWST2R5 | R/W | 0x094 | Wait-state 2 control register for memory bank 5 | 0x1F | Table 12–108 |
| SMBWSTOENR5 | R/W | 0x098 | Output-enable assertion delay control register for memory bank 5 | 0x0 | Table 12–109 |

**Table 105. Register overview: external SMC** *…continued***(base address 0x6000 0000)**

| Symbol | Access | Offset Address | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SMBWSTWENR5 | R/W | 0x09C | Write-enable assertion delay control register for memory bank 5 | 0x1 | Table 12–110 |
| SMBCR5 | R/W | 0x0A0 | Configuration register for memory bank 5 | 0x80 | Table 12–111 |
| SMBSR5 | R/W | 0x0A4 | Status register for memory bank 5 | 0x0 | Table 12–112 |
| **Bank 6** | | | | | |
| SMBIDCYR6 | R/W | 0x0A8 | Idle-cycle control register for memory bank 6 | 0xF | Table 12–106 |
| SMBWST1R6 | R/W | 0x0AC | Wait-state 1 control register for memory bank 6 | 0x1F | Table 12–107 |
| SMBWST2R6 | R/W | 0x0B0 | Wait-state 2 control register for memory bank 6 | 1F | Table 12–108 |
| SMBWSTOENR6 | R/W | 0x0B4 | Output-enable assertion delay control register for memory bank 6 | 0x0 | Table 12–109 |
| SMBWSTWENR6 | R/W | 0x0B8 | Write-enable assertion delay control register for memory bank 6 | 0x1 | Table 12–110 |
| SMBCR6 | R/W | 0x0BC | Configuration register for memory bank 6 | 0x40 | Table 12–111 |
| SMBSR6 | R/W | 0x0C0 | Status register for memory bank 6 | 0x0 | Table 12–112 |
| **Bank 7** | | | | | |
| SMBIDCYR7 | R/W | 0x0C4 | Idle-cycle control register for memory bank 7 | 0xF | Table 12–106 |
| SMBWST1R7 | R/W | 0x0C8 | Wait-state 1 control register for memory bank 7 | 0x1F | Table 12–107 |
| SMBWST2R7 | R/W | 0x0CC | Wait-state 2 control register for memory bank 7 | 0x1F | Table 12–108 |
| SMBWSTOENR7 | R/W | 0x0D0 | Output enable assertion delay control register for memory bank 7 | 0x0 | Table 12–109 |
| SMBWSTWENR7 | R/W | 0x0D4 | Write-enable assertion delay control register for memory bank 7 | 0x1 | Table 12–110 |
| SMBCR7 | R/W | 0x0D8 | Configuration register for memory bank 7 | 0x00 | Table 12–111 |
| SMBSR7 | R/W | 0x0DC | Status register for memory bank 7 | 0x0 | Table 12–112 |

## 4.1 Bank idle-cycle control registers

The bank idle-cycle control register configures the external bus turnaround cycles between read and write memory accesses to avoid bus contention on the external-memory data bus. The bus turnaround wait-time is inserted between external bus transfers in the case of:

- Read-to-read, to different memory banks
- Read-to-write, to the same memory bank
- Read-to-write, to different memory banks

Table 12–106 shows the bit assignment of the SMBIDCYR0 to SMBIDCYR7 registers.

**Table 106. SMBIDCYRn register bit description (SMBIDCYR0 to 7, addresses 0x6000 0000, 0x6000 001C, 0x6000 0038, 0x6000 0054, 0x6000 0070, 0x6000 008C, 0x6000 00A8, 0x6000 00C4)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 3 to 0 | IDCY[3:0] | R/W | | Idle or turnaround cycles. This register contains the number of bus turnaround cycles added between read and write accesses. The turnaround time is the programmed number of cycles multiplied by the system clock period |
| | | | 0xF* | |

## 4.2 Bank wait-state 1 control registers

The bank wait-state 1 control register configures the external transfer wait-states in read accesses. The bank configuration register contains the enable and polarity setting for the external wait.

The minimum wait-states value WST1 can be calculated from the following formula:

$$WST1 = \frac{t_{a(R)int} + t_{emd(read)}}{t_{clk(sys)}} - 1$$

Where:

$t_{a(R)int}$ = internal read delay. For more information see Ref. 31–1 Dynamic characteristics.

$t_{emd(read)}$ = external-memory read delay in ns.

Table 12–107 shows the bit assignment of the SMBWST1R0 to SMBWST1R7 registers.

**Table 107. SMBWST1Rn register bit description (SMBWRST1R0 to SMBWRST1R7, addresses 0x6000 0004, 0x6000 0020, 0x6000 003C, 0x6000 0058, 0x6000 0074, 0x6000 0090, 0x6000 00AC, 0x6000 00C8)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 4 to 0 | WST1[4:0] | R/W | | Wait-state 1. This register contains the length of read accesses, except for burst ROM where it defines the length of the first read access only. The read-access time is the programmed number of wait-states multiplied by the system clock period.<br>WST1 must be equal or larger than WSTOEN. |
| | | | 0x1F* | |

## 4.3 Bank wait-state 2 control registers

The bank wait-state 2 control register configures the external transfer wait-states in write accesses or in burst-read accesses. The bank configuration register contains the enable and polarity settings for the external wait.

Sequential-access burst-reads from burst-flash devices of the same type as for burst ROM are supported. Due to sharing of the SMBWST2R register between write and burst-read transfers it is only possible to have one setting at a time for burst flash; either write delay or the burst-read delay. This means that for write transfer the SMBWST2R register must be programmed with the write-delay value, and for a burst-read transfer it must be programmed with the burst-access delay.

The minimum wait-states value WST2 can be calculated from the following formula:

$$WST2 = \frac{t_{a(W)int} + t_{emd(write)}}{t_{clk(sys)}} - 1$$

Where:

$t_{a(W)int}$ = internal write delay. For more information see Ref. 31–1 Dynamic characteristics.

$t_{emd(write)}$ = external-memory write delay in ns.

Table 12–108 shows the bit assignment of the SMBWST2R0 to SMBWST2R7 registers.

**Table 108. SMBWST2Rn register bit description (SMBWRST1R0 to SMBWRST1R7, addresses 0x6000 0008, 0x6000 0024, 0x6000 0040, 0x6000 005C, 0x6000 0078, 0x6000 0094, 0x6000 00B0, 0x6000 00CC)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 5 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 4 to 0 | WST2[4:0] | R/W | | Wait-state 2. This register contains the length of write accesses, except for burst ROM where it defines the length of the burst-read accesses. The write-access time c.q. the burst ROM read access time is the programmed number of wait-states multiplied by the system clock period. WST2 must be equal or larger than WSTWEN. |
| | | | 0x1F* | |

## 4.4 Bank output enable assertion-delay control register

The bank output-enable assertion-delay 1 control register configures the delay between the assertion of the chip-select and the output enable. This delay is used to reduce the power consumption for memories that are unable to provide valid data immediately after the chip-select is asserted. Since the access is timed by the wait-states, the programmed value must be equal to or less than the bank wait-state 1 programmed value. The output enable is always deasserted at the same time as the chip-select at the end of the transfer. The bank configuration register contains the enable for output assertion delay.

Table 12–109 shows the bit assignment of the SMBWSTOENR0 to SMBWSTOENR7 registers.

Table 109. SMBWSTOENRn register bit description (SMBWSTOENR0 to SMBWSTOENR7, addresses 0x6000 000C, 0x6000 0028, 0x6000 0044, 0x6000 0060, 0x6000 007C, 0x6000 0098, 0x6000 00B4, 0x6000 00D0)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0. |
| 3 to 0 | WSTOEN | R/W | | Output-enable assertion delay. This register contains the length of the output-enable delay after the chip-select assertion. The output-enable assertion-delay time is the programmed number of wait-states multiplied by the system clock period. WSTOEN must be equal or smaller than WST1. |
| | | | 0x0* | |

## 4.5 Bank write-enable assertion-delay control register

The bank write-enable assertion-delay 1 control register configures the delay between the assertion of the chip-select and the write enable. This delay is used to reduce power consumption for memories. Since the access is timed by the wait-states the programmed value must be equal to or less than the bank wait-state 2 programmed value. The write enable is asserted half a system-clock cycle after assertion of the chip-select for logic 0 wait-states. The write enable is deasserted half a system-clock cycle before the chip-select, at the end of the transfer. The byte-lane select outputs have the same timing as the write-enable output for writes to 8-bit devices that use the byte-lane selects instead of the write enables. The bank configuration register contains the enable for output assertion delay.

Table 12–110 shows the bit assignment of the SMBWSTWENR0 to SMBWSTWENR7 registers.

Table 110. SMBWSTWENRn register bit description (SMBWSTWENR0 toSMBWSTWENR7, addresses 0x6000 0010, 0x6000 002C, 0x6000 0048, 0x6000 0064, 0x6000 0080, 0x6000 009C, 0x6000 00B8, 0x6000 00D4)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 3 to 0 | WSTWEN | R/W | | Write-enable assertion delay. This register contains the length of the write enable delay after the chip-select assertion. The write-enable assertion-delay time is the programmed number of wait-states multiplied by the system clock period. WSTWEN must be equal or smaller than WST2. |
| | | | 0x1* | |

## 4.6 Bank configuration register

The bank configuration register defines bank access for the connected memory device.

A data transfer can be initiated to the external memory greater than the width of the external-memory data bus. In this case the external transfer is automatically split up into several separate transfers.

Table 12–111 shows the bit assignment of the SMBCR0 to SMBCR7 registers.

**Table 111. SMBCRn register bit description (SMBCR0 toSMBCR7, addresses 0x6000 0014, 0x6000 0030, 0x6000 004C, 0x6000 0068, 0x6000 0084, 0x6000 00A0, 0x6000 00BC, 0x6000 00D8)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 7 and 6 | MW[1:0] | R/W | | Memory-width configuration |
| | | | 00* | 8-bit; reset value for memory banks 1, 3 and 7 |
| | | | 01* | 16-bit; reset value for memory banks 2 and 6 |
| | | | 10* | 32-bit; reset value for memory banks 0, 4 and 5 |
| | | | 11 | Reserved |
| 5 | BM | R/W | | Burst mode |
| | | | 1 | Sequential access burst-reads to a maximum of four consecutive locations is supported to increase the bandwidth by using reduced access time. However, bursts crossing quad boundaries are split up so that the first transfer after the boundary uses the slow wait-state 1 read timing |
| | | | 0* | The memory bank is configured for non-burst memory |
| 4 | WP | R/W | | Write-protect; e.g. (burst) ROM, read-only flash or SRAM |
| | | | 1 | The connected device is write-protected |
| | | | 0* | No write-protection is required |
| 3 | CSPOL | R/W | | Chip-select polarity |
| | | | 1 | The chip-select input is active HIGH |
| | | | 0* | The chip-select input is active LOW |
| 2 and 1 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 0 | RBLE | R/W | | Read-byte lane enable |
| | | | 1 | The byte-lane select pins are held asserted (logic 0) during a read access. This is for 16-bit or 32-bit devices where the separate write-enable signal is used and the byte-lane selects must be held asserted during a read. The write-enable pin WEN is used as the write-enable in this configuration. |
| | | | 0* | The byte-lane select pins BLSn are all deasserted (logic 1) during a read access. This is for 8-bit devices if the byte-lane enable is connected to the write-enable pin, so must be deasserted during a read access (default at reset). The byte-lane select pins are used as write-enables in this configuration |

## 4.7 Bank status register

The bank status register reflects the status flags of each memory bank.

Table 12–112 shows the bit assignment of the SMBSR0 to SMBSR7 registers.

**Table 112. SMBSRn register bit description (SMBSR0 toSMBSR7, addresses 0x6000 0018, 0x6000 0034, 0x6000 0050, 0x6000 006C, 0x6000 0088, 0x6000 00A4, 0x6000 00C0, 0x6000 00DC)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0 |
| 1 | WRITEPROTERR | R/W | | Write-protect error |
| | | | 1 | A write access to a write-protected memory device was initiated. Writing logic 1 to this register clears the write-protect status flag |
| | | | 0* | Writing a logic 0 has no effect |
| 0 | reserved | R | - | reserved; do not modify. Read as logic 0, write as logic 0 |

# UM10316

# Chapter 13: LPC29xx USB device

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The USB device controller is used in parts LPC2921/23/25, LPC2926/27/29, LPC2930, and LPC2939. The LPC2917/19/01 do not have an USB controller (see also Table 1–4).

## 2. Introduction

The Universal Serial Bus (USB) is a four-wire bus that supports communication between a host and one or more (up to 127) peripherals. The host controller allocates the USB bandwidth to attached devices through a token-based protocol. The bus supports hot plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The host schedules transactions in 1 ms frames. Each frame contains a Start-Of-Frame (SOF) marker and transactions that transfer data to or from device endpoints. Each device can have a maximum of 16 logical or 32 physical endpoints. There are four types of transfers defined for the endpoints. Control transfers are used to configure the device. Interrupt transfers are used for periodic data transfer. Bulk transfers are used when the rate of transfer is not critical. Isochronous transfers have guaranteed delivery time but no error correction.

For more information on the Universal Serial Bus, see the USB Implementers Forum website.

The USB device controller on the LPC29xx enables full-speed (12 Mb/s) data exchange with a USB host controller.

**Table 113. USB related acronyms, abbreviations, and definitions used in this chapter**

| Acronym/abbreviation | Description |
|---|---|
| AHB | Advanced High-performance bus |
| ATLE | Auto Transfer Length Extraction |
| ATX | Analog Transceiver |
| DD | DMA Descriptor |
| DDP | DMA Description Pointer |
| DMA | Direct Memory Access |
| EOP | End-Of-Packet |
| EP | Endpoint |
| EP_RAM | Endpoint RAM |
| FS | Full Speed |
| LED | Light Emitting Diode |
| LS | Low Speed |
| MPS | Maximum Packet Size |
| NAK | Negative Acknowledge |
| PLL | Phase Locked Loop |

Table 113. USB related acronyms, abbreviations, and definitions used in this chapter

| Acronym/abbreviation | Description |
|---|---|
| RAM | Random Access Memory |
| SOF | Start-Of-Frame |
| SIE | Serial Interface Engine |
| SRAM | Synchronous RAM |
| UDCA | USB Device Communication Area |
| USB | Universal Serial Bus |

## 3. Features

- Fully compliant with the *USB 2.0 specification* (full speed).
- Supports 32 physical (16 logical) endpoints.
- Supports Control, Bulk, Interrupt and Isochronous endpoints.
- Scalable realization of endpoints at run time.
- Endpoint maximum packet size selection (up to USB maximum specification) by software at run time.
- Supports SoftConnect and GoodLink features.
- Supports DMA transfers on all non-control endpoints.
- Allows dynamic switching between CPU controlled and DMA modes.
- Double buffer implementation for Bulk and Isochronous endpoints.

## 4. Fixed endpoint configuration

Table 13–114 shows the supported endpoint configurations. Endpoints are realized and configured at run time using the Endpoint realization registers, documented in Section 13–9.4 "Endpoint realization registers".

Table 114. Fixed endpoint configuration

| Logical endpoint | Physical endpoint | Endpoint type | Direction | Packet size (bytes) | Double buffer |
|---|---|---|---|---|---|
| 0 | 0 | Control | Out | 8, 16, 32, 64 | No |
| 0 | 1 | Control | In | 8, 16, 32, 64 | No |
| 1 | 2 | Interrupt | Out | 1 to 64 | No |
| 1 | 3 | Interrupt | In | 1 to 64 | No |
| 2 | 4 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 2 | 5 | Bulk | In | 8, 16, 32, 64 | Yes |
| 3 | 6 | Isochronous | Out | 1 to 1023 | Yes |
| 3 | 7 | Isochronous | In | 1 to 1023 | Yes |
| 4 | 8 | Interrupt | Out | 1 to 64 | No |
| 4 | 9 | Interrupt | In | 1 to 64 | No |
| 5 | 10 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 5 | 11 | Bulk | In | 8, 16, 32, 64 | Yes |
| 6 | 12 | Isochronous | Out | 1 to 1023 | Yes |

Table 114. Fixed endpoint configuration

| Logical endpoint | Physical endpoint | Endpoint type | Direction | Packet size (bytes) | Double buffer |
|---|---|---|---|---|---|
| 6 | 13 | Isochronous | In | 1 to 1023 | Yes |
| 7 | 14 | Interrupt | Out | 1 to 64 | No |
| 7 | 15 | Interrupt | In | 1 to 64 | No |
| 8 | 16 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 8 | 17 | Bulk | In | 8, 16, 32, 64 | Yes |
| 9 | 18 | Isochronous | Out | 1 to 1023 | Yes |
| 9 | 19 | Isochronous | In | 1 to 1023 | Yes |
| 10 | 20 | Interrupt | Out | 1 to 64 | No |
| 10 | 21 | Interrupt | In | 1 to 64 | No |
| 11 | 22 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 11 | 23 | Bulk | In | 8, 16, 32, 64 | Yes |
| 12 | 24 | Isochronous | Out | 1 to 1023 | Yes |
| 12 | 25 | Isochronous | In | 1 to 1023 | Yes |
| 13 | 26 | Interrupt | Out | 1 to 64 | No |
| 13 | 27 | Interrupt | In | 1 to 64 | No |
| 14 | 28 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 14 | 29 | Bulk | In | 8, 16, 32, 64 | Yes |
| 15 | 30 | Bulk | Out | 8, 16, 32, 64 | Yes |
| 15 | 31 | Bulk | In | 8, 16, 32, 64 | Yes |

## 5. Functional description

The architecture of the USB device controller is shown below in Figure 13–41.

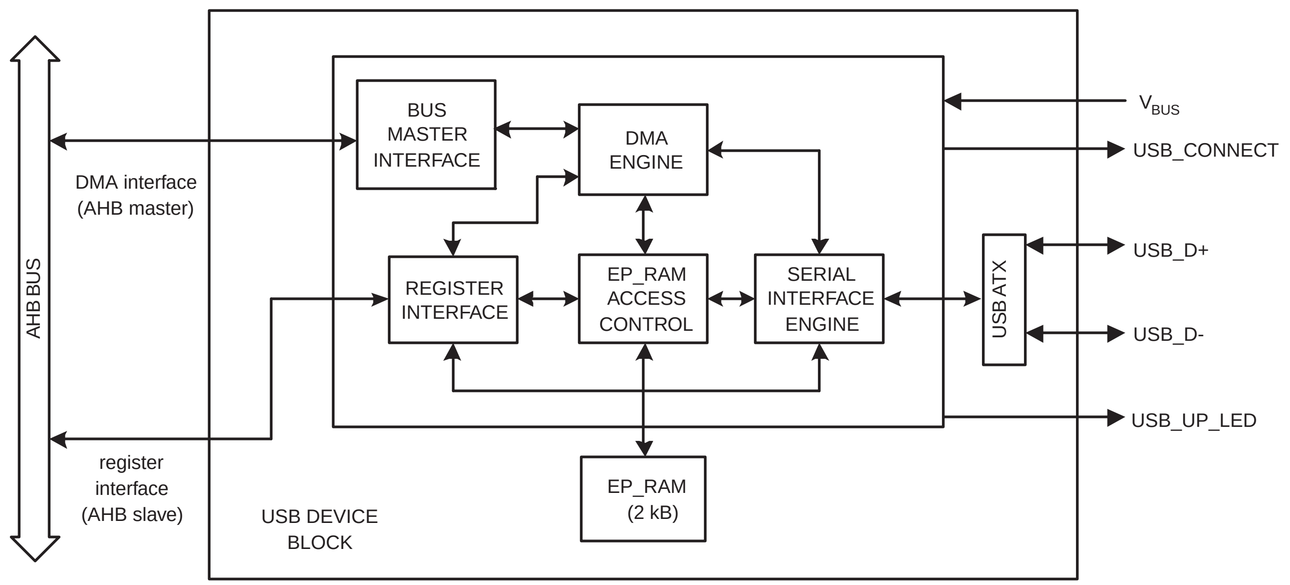


Fig 41. USB device controller block diagram

## 5.1 Analog transceiver

The USB Device Controller has a built-in analog transceiver (ATX). The USB ATX sends/receives the bi-directional D+ and D- signals of the USB bus.

## 5.2 Serial Interface Engine (SIE)

The SIE implements the full USB protocol layer. It is completely hardwired for speed and needs no firmware intervention. It handles transfer of data between the endpoint buffers in EP_RAM and the USB bus. The functions of this block include: synchronization pattern recognition, parallel/serial conversion, bit stuffing/de-stuffing, CRC checking/generation, PID verification/generation, address recognition, and handshake evaluation/generation.

## 5.3 Endpoint RAM (EP_RAM)

Each endpoint buffer is implemented as an SRAM based FIFO. The SRAM dedicated for this purpose is called the EP_RAM. Each realized endpoint has a reserved space in the EP_RAM. The total EP_RAM space required depends on the number of realized endpoints, the maximum packet size of the endpoint, and whether the endpoint supports double buffering.

## 5.4 EP_RAM access control

The EP_RAM Access Control logic handles transfer of data from/to the EP_RAM and the three sources that can access it: the CPU (via the Register Interface), the SIE, and the DMA Engine.

## 5.5 DMA engine and bus master interface

When enabled for an endpoint, the DMA Engine transfers data between RAM on the AHB bus and the endpoint’s buffer in EP_RAM. A single DMA channel is shared between all endpoints. When transferring data, the DMA Engine functions as a master on the AHB bus through the bus master interface.

## 5.6 Register interface

The Register Interface allows the CPU to control the operation of the USB Device Controller. It also provides a way to write transmit data to the controller and read receive data from the controller.

## 5.7 SoftConnect

The connection to the USB is accomplished by bringing D+ (for a full-speed device) HIGH through a 1.5 kOhm pull-up resistor. The SoftConnect feature can be used to allow software to finish its initialization sequence before deciding to establish connection to the USB. Re-initialization of the USB bus connection can also be performed without having to unplug the cable.

To use the SoftConnect feature, the CONNECT signal should control an external switch that connects the 1.5 kOhm resistor between D+ and +3.3V. Software can then control the CONNECT signal by writing to the CON bit using the SIE Set Device Status command.

### 5.8 GoodLink

Good USB connection indication is provided through GoodLink technology. When the device is successfully enumerated and configured, the LED indicator will be permanently ON. During suspend, the LED will be OFF.

This feature provides a user-friendly indicator on the status of the USB device. It is a useful field diagnostics tool to isolate faulty equipment.

To use the GoodLink feature the UP_LED signal should control an LED. The UP_LED signal is controlled using the SIE Configure Device command.

## 6. Operational overview

Transactions on the USB bus transfer data between device endpoints and the host. The direction of a transaction is defined with respect to the host. OUT transactions transfer data from the host to the device. IN transactions transfer data from the device to the host. All transactions are initiated by the host controller.

For an OUT transaction, the USB ATX receives the bi-directional D+ and D- signals of the USB bus. The Serial Interface Engine (SIE) receives the serial data from the ATX and converts it into a parallel data stream. The parallel data is written to the corresponding endpoint buffer in the EP_RAM.

For IN transactions, the SIE reads the parallel data from the endpoint buffer in EP_RAM, converts it into serial data, and transmits it onto the USB bus using the USB ATX.

Once data has been received or sent, the endpoint buffer can be read or written. How this is accomplished depends on the endpoint's type and operating mode. The two operating modes for each endpoint are Slave (CPU-controlled) mode, and DMA mode.

In Slave mode, the CPU transfers data between RAM and the endpoint buffer using the Register Interface. See Section 13–13 "Slave mode operation" for a detailed description of this mode.

In DMA mode, the DMA transfers data between RAM and the endpoint buffer. See Section 13–14 "DMA operation" for a detailed description of this mode.

## 7. Pin description

Table 115. USB external interface

| Name | Direction | Description |
|---|---|---|
| $V_{BUS}$ | I | $V_{BUS}$ status input. When this function is not enabled via its corresponding PINSEL register, it is driven HIGH internally. |
| USB_CONNECT[1] | O | SoftConnect control signal. |
| USB_UP_LED[1] | O | GoodLink LED control signal. |
| USB_D+[1] | I/O | Positive differential data. |
| USB_D-[1] | I/O | Negative differential data. |

[1] For LPC2930/39 use port 2 for a USB device connection (see Figure 15–51).

## 8. Clocking and power management

This section describes the clocking and power management features of the USB Device Controller.

### 8.1 Power requirements

The USB protocol insists on power management by the device. This becomes very critical if the device draws power from the bus (bus-powered device). The following constraints should be met by a bus-powered device:

1. A device in the non-configured state should draw a maximum of 100 mA from the bus.
2. A configured device can draw only up to what is specified in the Max Power field of the configuration descriptor. The maximum value is 500 mA.
3. A suspended device can draw a maximum of 500 μA.

### 8.2 Clocks

The USB device controller clocks are shown in Table 13–116

**Table 116. USB device controller clock sources**

| Clock source | Description |
|---|---|
| AHB master clock | Clock for the AHB master bus interface and DMA |
| AHB slave clock | Clock for the AHB slave interface |
| usbclk | Dedicated 48 MHz clock from CGU1 (BASE_USB_CLK) |

### 8.3 Power management support

To help conserve power, the USB device controller automatically disables the AHB master clock and usbclk when not in use.

When the USB Device Controller goes into the suspend state (bus is idle for 3 ms), the usbclk input to the device controller is automatically disabled, helping to conserve power. However, if software wishes to access the device controller registers, usbclk must be active. To allow access to the device controller registers while in the suspend state, the USBClkCtrl and USBClkSt registers are provided.

When software wishes to access the device controller registers, it should first ensure usbclk is enabled by setting DEV_CLK_EN in the USBClkCtrl register, and then poll the corresponding DEV_CLK_ON bit in USBClkSt until set. Once set, usbclk will remain enabled until DEV_CLK_EN is cleared by software.

When a DMA transfer occurs, the device controller automatically turns on the AHB master clock. Once asserted, it remains active for a minimum of 2 ms (2 frames), to help ensure that DMA throughput is not affected by turning off the AHB master clock. 2 ms after the last DMA access, the AHB master clock is automatically disabled to help conserve power. If desired, software also has the capability of forcing this clock to remain enabled using the USBClkCtrl register.

Note that the AHB slave clock is always enabled as long as the PCUSB bit of PCONP is set. When the device controller is not in use, all of the device controller clocks may be disabled by clearing PCUSB.

The USB_NEED_CLK signal is used to facilitate going into and waking up from chip Power-down mode. USB_NEED_CLK is asserted if any of the bits of the USBClkSt register are asserted.

After entering the suspend state with DEV_CLK_EN and AHB_CLK_EN cleared, the DEV_CLK_ON and AHB_CLK_ON will be cleared when the corresponding clock turns off. When both bits are zero, USB_NEED_CLK will be low, indicating that the chip can be put into Power-down mode by writing to the PCON register. The status of USB_NEED_CLK can be read from the USBIntSt register.

Any bus activity in the suspend state will cause the USB_NEED_CLK signal to be asserted. When the USB is configured to be a wakeup source from Power-down (USBWAKE bit set in the INTWAKE register), the assertion of USB_NEED_CLK causes the chip to wake up from Power-down mode.

### 8.4 Remote wake-up

The USB device controller supports software initiated remote wake-up. Remote wake-up involves resume signaling on the USB bus initiated from the device. This is done by clearing the SUS bit in the SIE Set Device Status register. Before writing into the register, all the clocks to the device controller have to be enabled using the USBClkCtrl register.

## 9. Register overview

Table 13–117 shows the USB Device Controller registers directly accessible by the CPU. The Serial Interface Engine (SIE) has other registers that are indirectly accessible via the SIE command registers. See Section 13–11 “Serial interface engine command description” for more info.

Table 117. Register overview: USB device register map (base address 0xE010 0000)

| Name | Access | Address | Description | Reset value[1] |
|---|---|---|---|---|
| **Clock control registers** | | | | |
| USBClkCtrl | R/W | 0xFF4 | USB Clock Control | 0x0000 0000 |
| USBClkSt | RO | 0xFF8 | USB Clock Status | 0x0000 0000 |
| **Device interrupt registers** | | | | |
| USBDevIntSt | RO | 0x200 | USB Device Interrupt Status | 0x0000 0010 |
| USBDevIntEn | R/W | 0x204 | USB Device Interrupt Enable | 0x0000 0000 |
| USBDevIntClr | WO | 0x208 | USB Device Interrupt Clear | 0x0000 0000 |
| USBDevIntSet | WO | 0x20C | USB Device Interrupt Set | 0x0000 0000 |
| USBDevIntPri | WO | 0x22C | USB Device Interrupt Priority | 0x00 |
| **Endpoint interrupt registers** | | | | |
| USBEpIntSt | RO | 0x230 | USB Endpoint Interrupt Status | 0x0000 0000 |
| USBEpIntEn | R/W | 0x234 | USB Endpoint Interrupt Enable | 0x0000 0000 |
| USBEpIntClr | WO | 0x238 | USB Endpoint Interrupt Clear | 0x0000 0000 |
| USBEpIntSet | WO | 0x23C | USB Endpoint Interrupt Set | 0x0000 0000 |

Table 117. Register overview: USB device register map (base address 0xE010 0000)

| Name | Access | Address | Description | Reset value[1] |
|---|---|---|---|---|
| USBEpIntPri | WO[2] | 0x240 | USB Endpoint Priority | 0x0000 0000 |
| **Endpoint realization registers** | | | | |
| USBReEp | R/W | 0x244 | USB Realize Endpoint | 0x0000 0003 |
| USBEpInd | WO[2] | 0x248 | USB Endpoint Index | 0x0000 0000 |
| USBMaxPSize | R/W | 0x24C | USB MaxPacketSize | 0x0000 0008 |
| **USB transfer registers** | | | | |
| USBRxData | RO | 0x218 | USB Receive Data | 0x0000 0000 |
| USBRxPLen | RO | 0x220 | USB Receive Packet Length | 0x0000 0000 |
| USBTxData | WO[2] | 0x21C | USB Transmit Data | 0x0000 0000 |
| USBTxPLen | WO[2] | 0x224 | USB Transmit Packet Length | 0x0000 0000 |
| USBCtrl | R/W | 0x228 | USB Control | 0x0000 0000 |
| **SIE Command registers** | | | | |
| USBCmdCode | WO[2] | 0x210 | USB Command Code | 0x0000 0000 |
| USBCmdData | RO | 0x214 | USB Command Data | 0x0000 0000 |
| **DMA registers** | | | | |
| USBDMARSt | RO | 0x250 | USB DMA Request Status | 0x0000 0000 |
| USBDMARClr | WO[2] | 0x254 | USB DMA Request Clear | 0x0000 0000 |
| USBDMARSet | WO[2] | 0x258 | USB DMA Request Set | 0x0000 0000 |
| USBUDCAH | R/W | 0x280 | USB UDCA Head | 0x0000 0000 |
| USBEpDMASt | RO | 0x284 | USB Endpoint DMA Status | 0x0000 0000 |
| USBEpDMAEn | WO[2] | 0x288 | USB Endpoint DMA Enable | 0x0000 0000 |
| USBEpDMADis | WO[2] | 0x28C | USB Endpoint DMA Disable | 0x0000 0000 |
| USBDMAIntSt | RO | 0x290 | USB DMA Interrupt Status | 0x0000 0000 |
| USBDMAIntEn | R/W | 0x294 | USB DMA Interrupt Enable | 0x0000 0000 |
| USBEoTIntSt | RO | 0x2A0 | USB End of Transfer Interrupt Status | 0x0000 0000 |
| USBEoTIntClr | WO[2] | 0x2A4 | USB End of Transfer Interrupt Clear | 0x0000 0000 |
| USBEoTIntSet | WO[2] | 0x2A8 | USB End of Transfer Interrupt Set | 0x0000 0000 |
| USBNDDRIntSt | RO | 0x2AC | USB New DD Request Interrupt Status | 0x0000 0000 |
| USBNDDRIntClr | WO[2] | 0x2B0 | USB New DD Request Interrupt Clear | 0x0000 0000 |
| USBNDDRIntSet | WO[2] | 0x2B4 | USB New DD Request Interrupt Set | 0x0000 0000 |
| USBSysErrIntSt | RO | 0x2B8 | USB System Error Interrupt Status | 0x0000 0000 |
| USBSysErrIntClr | WO[2] | 0x2BC | USB System Error Interrupt Clear | 0x0000 0000 |
| USBSysErrIntSet | WO[2] | 0x2C0 | USB System Error Interrupt Set | 0x0000 0000 |

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

[2] Reading WO register will return an invalid value.

## 9.1 Clock control registers

### 9.1.1 USB Clock Control register (USBClkCtrl - 0xE010 0FF4)

This register controls the clocking of the USB Device Controller. Whenever software wants to access the device controller registers, both DEV_CLK_EN and AHB_CLK_EN must be set. The PORTSEL_CLK_EN bit need only be set when accessing the USBPortSel register.

The software does not have to repeat this exercise for every register access, provided that the corresponding USBClkCtrl bits are already set. Note that this register is functional only when the PCUSB bit of PCONP is set; when PCUSB is cleared, all clocks to the device controller are disabled irrespective of the contents of this register. USBClkCtrl is a read/write register.

**Table 118. USBClkCtrl register (USBClkCtrl - address 0xE010 0FF4) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:5 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | AHB_CLK_EN | AHB clock enable | 0 |
| 3 | PORTSEL_CLK_EN | Port select register clock enable. | NA |
| 2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 1 | DEV_CLK_EN | Device clock enable. Enables the usbclk input to the device controller | 0 |
| 0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 9.1.2 USB Clock Status register (USBClkSt - 0xE010 0FF8)

This register holds the clock availability status. The bits of this register are ORed together to form the USB_NEED_CLK signal. When enabling a clock via USBClkCtrl, software should poll the corresponding bit in USBClkSt. If it is set, then software can go ahead with the register access. Software does not have to repeat this exercise for every access, provided that the USBClkCtrl bits are not disturbed. USBClkSt is a read only register.

**Table 119. USB Clock Status register (USBClkSt - address 0xE010 0FF8) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:5 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | AHB_CLK_ON | AHB clock on. | 0 |
| 3 | PORTSEL_CLK_ON | Port select register clock on. | NA |

**Table 119. USB Clock Status register (USBClkSt - address 0xE010 0FF8) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 2 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 1 | DEV_CLK_ON | Device clock on. The usbclk input to the device controller is active. | 0 |
| 0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 9.2 Device interrupt registers

### 9.2.1 USB Device Interrupt Status register (USBDevIntSt - 0xE010 0200)

The USBDevIntSt register holds the status of each interrupt. A 0 indicates no interrupt and 1 indicates the presence of the interrupt. USBDevIntSt is a read only register.

**Table 120. USB Device Interrupt Status register (USBDevIntSt - address 0xE010 0200) bit allocation**
*Reset value: 0x0000 0000*

| **Bit** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | - | - | - | - | - | - | - | - |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| **Symbol** | - | - | - | - | - | - | - | - |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Symbol** | - | - | - | - | - | - | ERR_INT | EP_RLZED |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Symbol** | TxENDPKT | Rx ENDPKT | CDFULL | CCEMPTY | DEV_STAT | EP_SLOW | EP_FAST | FRAME |

**Table 121. USB Device Interrupt Status register (USBDevIntSt - address 0xE010 0200) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:10 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 9 | ERR_INT | Error Interrupt. Any bus error interrupt from the USB device. Refer to Section 13–11.9 “Read Error Status (Command: 0xFB, Data: read 1 byte)” on page 191 | 0 |
| 8 | EP_RLZED | Endpoints realized. Set when Realize Endpoint register (USBReEp) or MaxPacketSize register (USBMaxPSize) is updated and the corresponding operation is completed. | 0 |
| 7 | TxENDPKT | The number of data bytes transferred to the endpoint buffer equals the number of bytes programmed in the TxPacket length register (USBTxPLen). | 0 |
| 6 | RxENDPKT | The current packet in the endpoint buffer is transferred to the CPU. | 0 |
| 5 | CDFULL | Command data register (USBCmdData) is full (Data can be read now). | 0 |
| 4 | CCEMPTY | The command code register (USBCmdCode) is empty (New command can be written). | 1 |
| 3 | DEV_STAT | Set when USB Bus reset, USB suspend change or Connect change event occurs. Refer to Section 13–11.6 “Set Device Status (Command: 0xFE, Data: write 1 byte)” on page 189. | 0 |

**Table 121. USB Device Interrupt Status register (USBDevIntSt - address 0xE010 0200) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 2 | EP_SLOW | Slow endpoints interrupt. If an Endpoint Interrupt Priority Register (USBEpIntPri) bit is not set, the corresponding endpoint interrupt will be routed to this bit. | 0 |
| 1 | EP_FAST | Fast endpoint interrupt. If an Endpoint Interrupt Priority register (USBEpIntPri) bit is set, the corresponding endpoint interrupt will be routed to this bit. | 0 |
| 0 | FRAME | The frame interrupt occurs every 1 ms. This is used in isochronous packet transfers. | 0 |

## 9.2.2 USB Device Interrupt Enable register (USBDevIntEn - 0xE010 0204)

Writing a one to a bit in this register enables the corresponding bit in USBDevIntSt to generate an interrupt on one of the interrupt lines when set. By default, the interrupt is routed to the USB_INT_REQ_LP interrupt line. Optionally, either the EP_FAST or FRAME interrupt may be routed to the USB_INT_REQ_HP interrupt line by changing the value of USBDevIntPri. USBDevIntEn is a read/write register.

**Table 122. USB Device Interrupt Enable register (USBDevIntEn - address 0xE010 0204) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | - | - | - | - | - | - | - |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| Symbol | - | - | - | - | - | - | - | - |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Symbol | - | - | - | - | - | - | ERR_INT | EP_RLZED |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Symbol | TxENDPKT | Rx ENDPKT | CDFULL | CCEMPTY | DEV_STAT | EP_SLOW | EP_FAST | FRAME |

**Table 123. USB Device Interrupt Enable register (USBDevIntEn - address 0xE010 0204) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBDevIntEn bit allocation table above | 0 | No interrupt is generated. | 0 |
| | | 1 | An interrupt will be generated when the corresponding bit in the Device Interrupt Status (USBDevIntSt) register (Table 13–120) is set. By default, the interrupt is routed to the USB_INT_REQ_LP interrupt line. Optionally, either the EP_FAST or FRAME interrupt may be routed to the USB_INT_REQ_HP interrupt line by changing the value of USBDevIntPri. | |

## 9.2.3 USB Device Interrupt Clear register (USBDevIntClr - 0xE010 0208)

Writing one to a bit in this register clears the corresponding bit in USBDevIntSt. Writing a zero has no effect.

**Remark:** Before clearing the EP_SLOW or EP_FAST interrupt bits, the corresponding endpoint interrupts in USBEpIntSt should be cleared.

USBDevIntClr is a write only register.

**Table 124. USB Device Interrupt Clear register (USBDevIntClr - address 0xE010 0208) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | - | - | - | - | - | - | - |

| Bit | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | - | - | - | - | - | - | - |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Symbol | - | - | - | - | - | - | ERR_INT | EP_RLZED |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Symbol | TxENDPKT | Rx ENDPKT | CDFULL | CCEMPTY | DEV_STAT | EP_SLOW | EP_FAST | FRAME |

**Table 125. USB Device Interrupt Clear register (USBDevIntClr - address 0xE010 0208) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBDevIntClr bit allocation table above | 0 | No effect. | 0 |
| | | 1 | The corresponding bit in USBDevIntSt (Section 13–9.2.1) is cleared. | |

## 9.2.4 USB Device Interrupt Set register (USBDevIntSet - 0xE010 020C)

Writing one to a bit in this register sets the corresponding bit in the USBDevIntSt. Writing a zero has no effect

USBDevIntSet is a write only register.

**Table 126. USB Device Interrupt Set register (USBDevIntSet - address 0xE010 020C) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | - | - | - | - | - | - | - |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| Symbol | - | - | - | - | - | - | - | - |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Symbol | - | - | - | - | - | - | ERR_INT | EP_RLZED |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Symbol | TxENDPKT | Rx ENDPKT | CDFULL | CCEMPTY | DEV_STAT | EP_SLOW | EP_FAST | FRAME |

**Table 127. USB Device Interrupt Set register (USBDevIntSet - address 0xE010 020C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBDevIntSet bit allocation table above | 0 | No effect. | 0 |
| | | 1 | The corresponding bit in USBDevIntSt (Section 13–9.2.1) is set. | |

## 9.2.5 USB Device Interrupt Priority register (USBDevIntPri - 0xE010 022C)

Writing one to a bit in this register causes the corresponding interrupt to be routed to the USB_INT_REQ_HP interrupt line. Writing zero causes the interrupt to be routed to the USB_INT_REQ_LP interrupt line. Either the EP_FAST or FRAME interrupt can be routed to USB_INT_REQ_HP, but not both. If the software attempts to set both bits to one, no interrupt will be routed to USB_INT_REQ_HP. USBDevIntPri is a write only register.

**Table 128. USB Device Interrupt Priority register (USBDevIntPri - address 0xE010 022C) bit description**

| Bit | Symbol | Value | 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 |
| 1 | EP_FAST | 0 | EP_FAST interrupt is routed to USB_INT_REQ_LP. | 0 |
| | | 1 | EP_FAST interrupt is routed to USB_INT_REQ_HP. | |
| 0 | FRAME | 0 | FRAME interrupt is routed to USB_INT_REQ_LP. | 0 |
| | | 1 | FRAME interrupt is routed to USB_INT_REQ_HP. | |

## 9.3 Endpoint interrupt registers

The registers in this group facilitate handling of endpoint interrupts. Endpoint interrupts are used in Slave mode operation.

### 9.3.1 USB Endpoint Interrupt Status register (USBEpIntSt - 0xE010 0230)

Each physical non-isochronous endpoint is represented by a bit in this register to indicate that it has generated an interrupt. All non-isochronous OUT endpoints generate an interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet is successfully transmitted, or when a NAK handshake is sent on the bus and the interrupt on NAK feature is enabled (see Section 13–11.3 “Set Mode (Command: 0xF3, Data: write 1 byte)” on page 188). A bit set to one in this register causes either the EP_FAST or EP_SLOW bit of USBDevIntSt to be set depending on the value of the corresponding bit of USBEpDevIntPri. USBEpIntSt is a read only register.

Note that for Isochronous endpoints, handling of packet data is done when the FRAME interrupt occurs.

**Table 129. USB Endpoint Interrupt Status register (USBEpIntSt - address 0xE010 0230) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | EP15TX | EP15RX | EP14TX | EP14RX | EP13TX | EP13RX | EP12TX | EP12RX |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| Symbol | EP11TX | EP11RX | EP10TX | EP10RX | EP9TX | EP9RX | EP8TX | EP8RX |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Symbol | EP7TX | EP7RX | EP6TX | EP6RX | EP5TX | EP5RX | EP4TX | EP4RX |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Symbol | EP3TX | EP3RX | EP2TX | EP2RX | EP1TX | EP1RX | EP0TX | EP0RX |

**Table 130. USB Endpoint Interrupt Status register (USBEpIntSt - address 0xE010 0230) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | EP0RX | Endpoint 0, Data Received Interrupt bit. | 0 |
| 1 | EP0TX | Endpoint 0, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 2 | EP1RX | Endpoint 1, Data Received Interrupt bit. | 0 |
| 3 | EP1TX | Endpoint 1, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 4 | EP2RX | Endpoint 2, Data Received Interrupt bit. | 0 |
| 5 | EP2TX | Endpoint 2, Data Transmitted Interrupt bit or sent a NAK. | 0 |

**Table 130. USB Endpoint Interrupt Status register (USBEpIntSt - address 0xE010 0230) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 6 | EP3RX | Endpoint 3, Isochronous endpoint. | NA |
| 7 | EP3TX | Endpoint 3, Isochronous endpoint. | NA |
| 8 | EP4RX | Endpoint 4, Data Received Interrupt bit. | 0 |
| 9 | EP4TX | Endpoint 4, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 10 | EP5RX | Endpoint 5, Data Received Interrupt bit. | 0 |
| 11 | EP5TX | Endpoint 5, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 12 | EP6RX | Endpoint 6, Isochronous endpoint. | NA |
| 13 | EP6TX | Endpoint 6, Isochronous endpoint. | NA |
| 14 | EP7RX | Endpoint 7, Data Received Interrupt bit. | 0 |
| 15 | EP7TX | Endpoint 7, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 16 | EP8RX | Endpoint 8, Data Received Interrupt bit. | 0 |
| 17 | EP8TX | Endpoint 8, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 18 | EP9RX | Endpoint 9, Isochronous endpoint. | NA |
| 19 | EP9TX | Endpoint 9, Isochronous endpoint. | NA |
| 20 | EP10RX | Endpoint 10, Data Received Interrupt bit. | 0 |
| 21 | EP10TX | Endpoint 10, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 22 | EP11RX | Endpoint 11, Data Received Interrupt bit. | 0 |
| 23 | EP11TX | Endpoint 11, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 24 | EP12RX | Endpoint 12, Isochronous endpoint. | NA |
| 25 | EP12TX | Endpoint 12, Isochronous endpoint. | NA |
| 26 | EP13RX | Endpoint 13, Data Received Interrupt bit. | 0 |
| 27 | EP13TX | Endpoint 13, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 28 | EP14RX | Endpoint 14, Data Received Interrupt bit. | 0 |
| 29 | EP14TX | Endpoint 14, Data Transmitted Interrupt bit or sent a NAK. | 0 |
| 30 | EP15RX | Endpoint 15, Data Received Interrupt bit. | 0 |
| 31 | EP15TX | Endpoint 15, Data Transmitted Interrupt bit or sent a NAK. | 0 |

### 9.3.2 USB Endpoint Interrupt Enable register (USBEpIntEn - 0xE010 0234)

Setting a bit to 1 in this register causes the corresponding bit in USBEpIntSt to be set when an interrupt occurs for the associated endpoint. Setting a bit to 0 causes the corresponding bit in USBDMARSt to be set when an interrupt occurs for the associated endpoint. USBEpIntEn is a read/write register.

**Table 131. USB Endpoint Interrupt Enable register (USBEpIntEn - address 0xE010 0234) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | EP15TX | EP15RX | EP14TX | EP14RX | EP13TX | EP13RX | EP12TX | EP12RX |
| Bit | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Symbol | EP11TX | EP11RX | EP10TX | EP10RX | EP9TX | EP9RX | EP8TX | EP8RX |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Symbol | EP7TX | EP7RX | EP6TX | EP6RX | EP5TX | EP5RX | EP4TX | EP4RX |

| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | EP3TX | EP3RX | EP2TX | EP2RX | EP1TX | EP1RX | EP0TX | EP0RX |

**Table 132. USB Endpoint Interrupt Enable register (USBEpIntEn - address 0xE010 0234) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBEpIntEn bit allocation table above | 0 | The corresponding bit in USBDMARSt is set when an interrupt occurs for this endpoint. | 0 |
| | | 1 | The corresponding bit in USBEpIntSt is set when an interrupt occurs for this endpoint. Implies Slave mode for this endpoint. | |

## 9.3.3 USB Endpoint Interrupt Clear register (USBEpIntClr - 0xE010 0238)

Writing a one to this a bit in this register causes the SIE Select Endpoint/Clear Interrupt command to be executed (Table 13–176) for the corresponding physical endpoint. Writing zero has no effect. Before executing the Select Endpoint/Clear Interrupt command, the CDFULL bit in USBDevIntSt is cleared by hardware. On completion of the command, the CDFULL bit is set, USBCmdData contains the status of the endpoint, and the corresponding bit in USBEpIntSt is cleared.

Notes:

- When clearing interrupts using USBEpIntClr, software should wait for CDFULL to be set to ensure the corresponding interrupt has been cleared before proceeding.
- While setting multiple bits in USBEpIntClr simultaneously is possible, it is not recommended; only the status of the endpoint corresponding to the least significant interrupt bit cleared will be available at the end of the operation.
- Alternatively, the SIE Select Endpoint/Clear Interrupt command can be directly invoked using the SIE command registers, but using USBEpIntClr is recommended because of its ease of use.

Each physical endpoint has its own reserved bit in this register. The bit field definition is the same as that of USBEpIntSt shown in Table 13–129 . USBEpIntClr is a write only register.

**Table 133. USB Endpoint Interrupt Clear register (USBEpIntClr - address 0xE010 0238) bit allocation**
*Reset value: 0x0000 0000*

| **Bit** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | EP15TX | EP15RX | EP14TX | EP14RX | EP13TX | EP13RX | EP12TX | EP12RX |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| **Symbol** | EP11TX | EP11RX | EP10TX | EP10RX | EP9TX | EP9RX | EP8TX | EP8RX |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Symbol** | EP7TX | EP7RX | EP6TX | EP6RX | EP5TX | EP5RX | EP4TX | EP4RX |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Symbol** | EP3TX | EP3RX | EP2TX | EP2RX | EP1TX | EP1RX | EP0TX | EP0RX |

**Table 134. USB Endpoint Interrupt Clear register (USBEpIntClr - address 0xE010 0238) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBEpIntClr bit allocation table above | 0 | No effect. | 0 |
| | | 1 | Clears the corresponding bit in USBEpIntSt, by executing the SIE Select Endpoint/Clear Interrupt command for this endpoint. | |

## 9.3.4 USB Endpoint Interrupt Set register (USBEpIntSet - 0xE010 023C)

Writing a one to a bit in this register sets the corresponding bit in USBEpIntSt. Writing zero has no effect. Each endpoint has its own bit in this register. USBEpIntSet is a write only register.

**Table 135. USB Endpoint Interrupt Set register (USBEpIntSet - address 0xE010 023C) bit allocation**
*Reset value: 0x0000 0000*

| **Bit** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | EP15TX | EP15RX | EP14TX | EP14RX | EP13TX | EP13RX | EP12TX | EP12RX |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| **Symbol** | EP11TX | EP11RX | EP10TX | EP10RX | EP9TX | EP9RX | EP8TX | EP8RX |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Symbol** | EP7TX | EP7RX | EP6TX | EP6RX | EP5TX | EP5RX | EP4TX | EP4RX |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Symbol** | EP3TX | EP3RX | EP2TX | EP2RX | EP1TX | EP1RX | EP0TX | EP0RX |

**Table 136. USB Endpoint Interrupt Set register (USBEpIntSet - address 0xE010 023C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBEpIntSet bit allocation table above | 0 | No effect. | 0 |
| | | 1 | Sets the corresponding bit in USBEpIntSt. | |

## 9.3.5 USB Endpoint Interrupt Priority register (USBEpIntPri - 0xE010 0240)

This register determines whether an endpoint interrupt is routed to the EP_FAST or EP_SLOW bits of USBDevIntSt. If a bit in this register is set to one, the interrupt is routed to EP_FAST, if zero it is routed to EP_SLOW. Routing of multiple endpoints to EP_FAST or EP_SLOW is possible.

Note that the USBDevIntPri register determines whether the EP_FAST interrupt is routed to the USB_INT_REQ_HP or USB_INT_REQ_LP interrupt line.

USBEpIntPri is a write only register.

**Table 137. USB Endpoint Interrupt Priority register (USBEpIntPri - address 0xE010 0240) bit allocation**
*Reset value: 0x0000 0000*

| **Bit** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | EP15TX | EP15RX | EP14TX | E14RX | EP13TX | EP13RX | EP12TX | EP12RX |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| **Symbol** | EP11TX | EP11RX | EP10TX | EP10RX | EP9TX | EP9RX | EP8TX | EP8RX |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Symbol** | EP7TX | EP7RX | EP6TX | EP6RX | EP5TX | EP5RX | EP4TX | EP4RX |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | EP3TX | EP3RX | EP2TX | EP2RX | EP1TX | EP1RX | EP0TX | EP0RX |

Table 138. USB Endpoint Interrupt Priority register (USBEpIntPri - address 0xE010 0240) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | See USBEpIntPri bit allocation table above | 0 | The corresponding interrupt is routed to the EP_SLOW bit of USBDevIntSt | 0 |
| | | 1 | The corresponding interrupt is routed to the EP_FAST bit of USBDevIntSt | |

## 9.4 Endpoint realization registers

The registers in this group allow realization and configuration of endpoints at run time.

### 9.4.1 EP RAM requirements

The USB device controller uses a RAM based FIFO for each endpoint buffer. The RAM dedicated for this purpose is called the Endpoint RAM (EP_RAM). Each endpoint has space reserved in the EP_RAM. The EP_RAM space required for an endpoint depends on its MaxPacketSize and whether it is double buffered. 32 words of EP_RAM are used by the device for storing the endpoint buffer pointers. The EP_RAM is word aligned but the MaxPacketSize is defined in bytes hence the RAM depth has to be adjusted to the next word boundary. Also, each buffer has one word header showing the size of the packet length received.

The EP_ RAM space (in words) required for the physical endpoint can be expressed as

$$EPRAMspace = \left(\frac{MaxPacketSize + 3}{4} + 1\right) \times dbstatus$$

where dbstatus = 1 for a single buffered endpoint and 2 for double a buffered endpoint.

Since all the realized endpoints occupy EP_RAM space, the total EP_RAM requirement is

$$TotalEPRAMspace = 32 + \sum_{n=0}^{N} EPRAMspace(n)$$

where N is the number of realized endpoints. Total EP_RAM space should not exceed 2048 bytes.

### 9.4.2 USB Realize Endpoint register (USBReEp - 0xE010 0244)

Writing one to a bit in this register causes the corresponding endpoint to be realized. Writing zeros causes it to be unrealized. This register returns to its reset state when a bus reset occurs. USBReEp is a read/write register.

**Table 139. USB Realize Endpoint register (USBReEp - address 0xE010 0244) bit allocation**
*Reset value: 0x0000 0003*

| **Bit** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** |
|---|---|---|---|---|---|---|---|---|
| **Symbol** | EP31 | EP30 | EP29 | EP28 | EP27 | EP26 | EP25 | EP24 |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| **Symbol** | EP23 | EP22 | EP21 | EP20 | EP19 | EP18 | EP17 | EP16 |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Symbol** | EP15 | EP14 | EP13 | EP12 | EP11 | EP10 | EP9 | EP8 |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Symbol** | EP7 | EP6 | EP5 | EP4 | EP3 | EP2 | EP1 | EP0 |

**Table 140. USB Realize Endpoint register (USBReEp - address 0xE010 0244) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx | 0 | Endpoint EPxx is not realized. | 0 |
| | | 1 | Endpoint EPxx is realized. | |
| 1 | EP1 | 0 | Control endpoint EP1 is not realized. | 1 |
| | | 1 | Control endpoint EP1 is realized. | |
| 0 | EP0 | 0 | Control endpoint EP0 is not realized. | 1 |
| | | 1 | Control endpoint EP0 is realized. | |

On reset, only the control endpoints are realized. Other endpoints, if required, are realized by programming the corresponding bits in USBReEp. To calculate the required EP_RAM space for the realized endpoints, see Section 13–9.4.1.

Realization of endpoints is a multi-cycle operation. Pseudo code for endpoint realization is shown below.

```
Clear EP_RLZED bit in USBDevIntSt;

for every endpoint to be realized,
{
    /* OR with the existing value of the Realize Endpoint register */
    USBReEp |= (UInt32) ((0x1 << endpt));
    /* Load Endpoint index Reg with physical endpoint no.*/
    USBEpIn = (UInt32) endpointnumber;

    /* load the max packet size Register */
    USBEpMaxPSize = MPS;

    /* check whether the EP_RLZED bit in the Device Interrupt Status register is set
    */
    while (!(USBDevIntSt & EP_RLZED))
    {
        /* wait until endpoint realization is complete */
    }
    /* Clear the EP_RLZED bit */
    Clear EP_RLZED bit in USBDevIntSt;
}
```

The device will not respond to any transactions to unrealized endpoints. The SIE Configure Device command will only cause realized and enabled endpoints to respond to transactions. For details see Table 13–171.

### 9.4.3 USB Endpoint Index register (USBEpIn - 0xE010 0248)

Each endpoint has a register carrying the MaxPacketSize value for that endpoint. This is in fact a register array. Hence before writing, this register is addressed through the USBEpIn register.

The USBEpIn register will hold the physical endpoint number. Writing to USBMaxPSize will set the array element pointed to by USBEpIn. USBEpIn is a write only register.

**Table 141. USB Endpoint Index register (USBEpIn - address 0xE010 0248) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:5 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4:0 | PHY_EP | Physical endpoint number (0-31) | 0 |

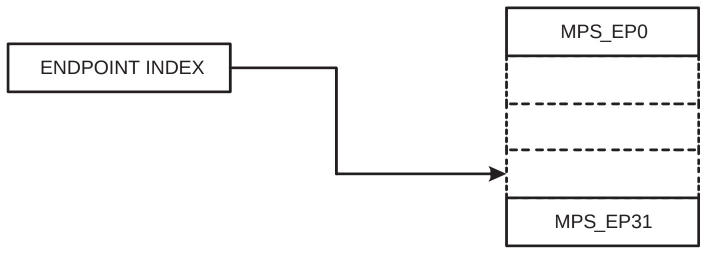
### 9.4.4 USB MaxPacketSize register (USBMaxPSize - 0xE010 024C)

On reset, the control endpoint is assigned the maximum packet size of 8 bytes. Other endpoints are assigned 0. Modifying USBMaxPSize will cause the endpoint buffer addresses within the EP_RAM to be recalculated. This is a multi-cycle process. At the end, the EP_RLZED bit will be set in USBDevIntSt (Table 13–120). USBMaxPSize array indexing is shown in Figure 13–42. USBMaxPSize is a read/write register.

**Table 142. USB MaxPacketSize register (USBMaxPSize - address 0xE010 024C) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:10 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 9:0 | MPS | The maximum packet size value. | 0x008[1] |

[1] Reset value for EP0 and EP1. All other endpoints have a reset value of 0x0.



The Endpoint Index is set via the USBEpIn register. MPS_EP0 to MPS_EP31 are accessed via the USBMaxPSize register.

**Fig 42. USB MaxPacketSize register array indexing**

## 9.5 USB transfer registers

The registers in this group are used for transferring data between endpoint buffers and RAM in Slave mode operation. See Section 13–13 "Slave mode operation".

### 9.5.1 USB Receive Data register (USBRxData - 0xE010 0218)

For an OUT transaction, the CPU reads the endpoint buffer data from this register. Before reading this register, the RD_EN bit and LOG_ENDPOINT field of the USBCtrl register should be set appropriately. On reading this register, data from the selected endpoint buffer is fetched. The data is in little endian format: the first byte received from the USB bus will be available in the least significant byte of USBRxData. USBRxData is a read only register.

**Table 143. USB Receive Data register (USBRxData - address 0xE010 0218) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | RX_DATA | Data received. | 0x0000 0000 |

### 9.5.2 USB Receive Packet Length register (USBRxPLen - 0xE010 0220)

This register contains the number of bytes remaining in the endpoint buffer for the current packet being read via the USBRxData register, and a bit indicating whether the packet is valid or not. Before reading this register, the RD_EN bit and LOG_ENDPOINT field of the USBCtrl register should be set appropriately. This register is updated on each read of the USBRxData register. USBRxPLen is a read only register.

**Table 144. USB Receive Packet Length register (USBRxPlen - address 0xE010 0220) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:12 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 11 | PKT_RDY | - | The PKT_LNGTH field is valid and the packet is ready for reading. | 0 |
| 10 | DV | | Data valid. This bit is useful for isochronous endpoints. Non-isochronous endpoints do not raise an interrupt when an erroneous data packet is received. But invalid data packet can be produced with a bus reset. For isochronous endpoints, data transfer will happen even if an erroneous packet is received. In this case DV bit will not be set for the packet. | 0 |
| | | 0 | Data is invalid. | |
| | | 1 | Data is valid. | |
| 9:0 | PKT_LNGTH | - | The remaining number of bytes to be read from the currently selected endpoint's buffer. When this field decrements to 0, the RxENDPKT bit will be set in USBDevIntSt. | 0 |

### 9.5.3 USB Transmit Data register (USBTxData - 0xE010 021C)

For an IN transaction, the CPU writes the endpoint data into this register. Before writing to this register, the WR_EN bit and LOG_ENDPOINT field of the USBCtrl register should be set appropriately, and the packet length should be written to the USBTxPlen register. On writing this register, the data is written to the selected endpoint buffer. The data is in little endian format: the first byte sent on the USB bus will be the least significant byte of USBTxData. USBTxData is a write only register.

**Table 145. USB Transmit Data register (USBTxData - address 0xE010 021C) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:0 | TX_DATA | Transmit Data. | 0x0000 0000 |

## 9.5.4 USB Transmit Packet Length register (USBTxPLen - 0xE010 0224)

This register contains the number of bytes transferred from the CPU to the selected endpoint buffer. Before writing data to USBTxData, software should first write the packet length (≤ MaxPacketSize) to this register. After each write to USBTxData, hardware decrements USBTxPLen by 4. The WR_EN bit and LOG_ENDPOINT field of the USBCtrl register should be set to select the desired endpoint buffer before starting this process.

For data buffers larger than the endpoint's MaxPacketSize, software should submit data in packets of MaxPacketSize, and send the remaining extra bytes in the last packet. For example, if the MaxPacketSize is 64 bytes and the data buffer to be transferred is of length 130 bytes, then the software sends two 64-byte packets and the remaining 2 bytes in the last packet. So, a total of 3 packets are sent on USB. USBTxPLen is a write only register.

**Table 146. USB Transmit Packet Length register (USBTxPLen - address 0xE010 0224) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:10 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 9:0 | PKT_LNGTH | - | The remaining number of bytes to be written to the selected endpoint buffer. This field is decremented by 4 by hardware after each write to USBTxData. When this field decrements to 0, the TxENDPKT bit will be set in USBDevIntSt. | 0x000 |

## 9.5.5 USB Control register (USBCtrl - 0xE010 0228)

This register controls the data transfer operation of the USB device. It selects the endpoint buffer that is accessed by the USBRxData and USBTxData registers, and enables reading and writing them. USBCtrl is a read/write register.

**Table 147. USB Control register (USBCtrl - address 0xE010 0228) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:6 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 5:2 | LOG_ENDPOINT | - | Logical Endpoint number. | 0x0 |
| 1 | WR_EN | | Write mode control. Enables writing data to the IN endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBTxData register. This bit is cleared by hardware when the number of bytes in USBTxLen have been sent. | 0 |
| | | 0 | Write mode is disabled. | |
| | | 1 | Write mode is enabled. | |

**Table 147. USB Control register (USBCtrl - address 0xE010 0228) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 0 | RD_EN | | Read mode control. Enables reading data from the OUT endpoint buffer for the endpoint specified in the LOG_ENDPOINT field using the USBRxData register. This bit is cleared by hardware when the last word of the current packet is read from USBRxData. | 0 |
| | | 0 | Read mode is disabled. | |
| | | 1 | Read mode is enabled. | |

## 9.6 SIE command code registers

The SIE command code registers are used for communicating with the Serial Interface Engine. See Section 13–11 “Serial interface engine command description” for more information.

### 9.6.1 USB Command Code register (USBCmdCode - 0xE010 0210)

This register is used for sending the command and write data to the SIE. The commands written here are propagated to the SIE and executed there. After executing the command, the register is empty, and the CCEMPTY bit of USBDevIntSt register is set. See Section 13–11 for details. USBCmdCode is a write only register.

**Table 148. USB Command Code register (USBCmdCode - address 0xE010 0210) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:24 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 23:16 | CMD_CODE/ CMD_WDATA | | This is a multi-purpose field. When CMD_PHASE is Command or Read, this field contains the code for the command (CMD_CODE). When CMD_PHASE is Write, this field contains the command write data (CMD_WDATA). | 0x00 |
| 15:8 | CMD_PHASE | | The command phase: | 0x00 |
| | | 0x01 | Write | |
| | | 0x02 | Read | |
| | | 0x05 | Command | |
| 7:0 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

### 9.6.2 USB Command Data register (USBCmdData - 0xE010 0214)

This register contains the data retrieved after executing a SIE command. When the data is ready to be read, the CD_FULL bit of the USBDevIntSt register is set. See Table 13–120 for details. USBCmdData is a read only register.

**Table 149. USB Command Data register (USBCmdData - address 0xE010 0214) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:8 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7:0 | CMD_RDATA | Command Read Data. | 0x00 |

## 9.7 DMA registers

The registers in this group are used for the DMA mode of operation (see Section 13–14 “DMA operation”)

### 9.7.1 USB DMA Request Status register (USBDMARSt - 0xE010 0250)

A bit in this register associated with a non-isochronous endpoint is set by hardware when an endpoint interrupt occurs (see the description of USBEpIntSt) and the corresponding bit in USBEpIntEn is 0. A bit associated with an isochronous endpoint is set when the corresponding bit in USBEpIntEn is 0 and a FRAME interrupt occurs. A set bit serves as a flag for the DMA engine to start the data transfer if the DMA is enabled for the corresponding endpoint in the USBEpDMASt register. The DMA cannot be enabled for control endpoints (EP0 and EP1). USBDMARSt is a read only register.

**Table 150. USB DMA Request Status register (USBDMARSt - address 0xE010 0250) bit allocation**
*Reset value: 0x0000 0000*

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| Symbol | EP31 | EP30 | EP29 | EP28 | EP27 | EP26 | EP25 | EP24 |
| **Bit** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| Symbol | EP23 | EP22 | EP21 | EP20 | EP19 | EP18 | EP17 | EP16 |
| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Symbol | EP15 | EP14 | EP13 | EP12 | EP11 | EP10 | EP9 | EP8 |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Symbol | EP7 | EP6 | EP5 | EP4 | EP3 | EP2 | EP1 | EP0 |

**Table 151. USB DMA Request Status register (USBDMARSt - address 0xE010 0250) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx | | Endpoint xx ($2 \le xx \le 31$) DMA request. | 0 |
| | | 0 | DMA not requested by endpoint xx. | |
| | | 1 | DMA requested by endpoint xx. | |
| 0 | EP0 | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and EP0 bit must be 0). | 0 |
| 1 | EP1 | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and EP1 bit must be 0). | 0 |

[1] DMA can not be enabled for this endpoint and the corresponding bit in the USBDMARSt must be 0.

### 9.7.2 USB DMA Request Clear register (USBDMARClr - 0xE010 0254)

Writing one to a bit in this register will clear the corresponding bit in the USBDMARSt register. Writing zero has no effect.

This register is intended for initialization prior to enabling the DMA for an endpoint. When the DMA is enabled for an endpoint, hardware clears the corresponding bit in USBDMARSt on completion of a packet transfer. Therefore, software should not clear the bit using this register while the endpoint is enabled for DMA operation.

USBDMARClr is a write only register.

The USBDMARClr bit allocation is identical to the USBDMARSt register (Table 13–150).

**Table 152. USB DMA Request Clear register (USBDMARClr - address 0xE010 0254) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx | | Clear the endpoint xx (2 ≤ xx ≤ 31) DMA request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Clear the corresponding bit in USBDMARSt. | |
| 1 | EP1 | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1 bit must be 0). | 0 |
| 0 | EP0 | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0 bit must be 0). | 0 |

## 9.7.3 USB DMA Request Set register (USBDMARSet - 0xE010 0258)

Writing one to a bit in this register sets the corresponding bit in the USBDMARSt register. Writing zero has no effect.

This register allows software to raise a DMA request. This can be useful when switching from Slave to DMA mode of operation for an endpoint: if a packet to be processed in DMA mode arrives before the corresponding bit of USBEpIntEn is cleared, the DMA request is not raised by hardware. Software can then use this register to manually start the DMA transfer.

Software can also use this register to initiate a DMA transfer to proactively fill an IN endpoint buffer before an IN token packet is received from the host.

USBDMARSet is a write only register.

The USBDMARSet bit allocation is identical to the USBDMARSt register (Table 13–150).

**Table 153. USB DMA Request Set register (USBDMARSet - address 0xE010 0258) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx | | Set the endpoint xx (2 ≤ xx ≤ 31) DMA request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Set the corresponding bit in USBDMARSt. | |
| 1 | EP1 | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1 bit must be 0). | 0 |
| 0 | EP0 | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0 bit must be 0). | 0 |

### 9.7.4 USB UDCA Head register (USBUDCAH - 0xE010 0280)

The UDCA (USB Device Communication Area) Head register maintains the address where the UDCA is located in on-chip RAM. Refer to Section 13–14.2 "USB device communication area" and Section 13–14.4 "The DMA descriptor" for more details on the UDCA and DMA descriptors. USBUDCAH is a read/write register.

**Table 154. USB UDCA Head register (USBUDCAH - address 0xE010 0280) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:7 | UDCA_ADDR | Start address of the UDCA. | 0 |
| 6:0 | - | Reserved. Software should not write ones to reserved bits. The UDCA is aligned to 128-byte boundaries. | 0x00 |

### 9.7.5 USB EP DMA Status register (USBEpDMASt - 0xE010 0284)

Bits in this register indicate whether DMA operation is enabled for the corresponding endpoint. A DMA transfer for an endpoint can start only if the corresponding bit is set in this register. USBEpDMASt is a read only register.

**Table 155. USB EP DMA Status register (USBEpDMASt - address 0xE010 0284) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx_DMA_ENABLE | | endpoint xx ($2 \le xx \le 31$) DMA enabled bit. | 0 |
| | | 0 | The DMA for endpoint EPxx is disabled. | |
| | | 1 | The DMA for endpoint EPxx is enabled. | |
| 1 | EP1_DMA_ENABLE | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_ENABLE bit must be 0). | 0 |
| 0 | EP0_DMA_ENABLE | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_ENABLE bit must be 0). | 0 |

### 9.7.6 USB EP DMA Enable register (USBEpDMAEn - 0xE010 0288)

Writing one to a bit to this register will enable the DMA operation for the corresponding endpoint. Writing zero has no effect.The DMA cannot be enabled for control endpoints EP0 and EP1. USBEpDMAEn is a write only register.

**Table 156. USB EP DMA Enable register (USBEpDMAEn - address 0xE010 0288) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx_DMA_ENABLE | | Endpoint xx($2 \le xx \le 31$) DMA enable control bit. | 0 |
| | | 0 | No effect. | |
| | | 1 | Enable the DMA operation for endpoint EPxx. | |
| 1 | EP1_DMA_ENABLE | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_ENABLE bit must be 0). | 0 |
| 0 | EP0_DMA_ENABLE | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_ENABLE bit value must be 0). | 0 |

### 9.7.7 USB EP DMA Disable register (USBEpDMADis - 0xE010 028C)

Writing a one to a bit in this register clears the corresponding bit in USBEpDMASt. Writing zero has no effect on the corresponding bit of USBEpDMASt. Any write to this register clears the internal DMA_PROCEED flag. Refer to Section 13–14.5.4 "Optimizing descriptor fetch" for more information on the DMA_PROCEED flag. If a DMA transfer is in progress for an endpoint when its corresponding bit is cleared, the transfer is completed before the DMA is disabled. When an error condition is detected during a DMA transfer, the corresponding bit is cleared by hardware. USBEpDMADis is a write only register.

Table 157. USB EP DMA Disable register (USBEpDMADis - address 0xE010 028C) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:2 | EPxx_DMA_DISABLE | | Endpoint xx (2 ≤ xx ≤ 31) DMA disable control bit. | 0 |
| | | 0 | No effect. | |
| | | 1 | Disable the DMA operation for endpoint EPxx. | |
| 1 | EP1_DMA_DISABLE | 0 | Control endpoint IN (DMA cannot be enabled for this endpoint and the EP1_DMA_DISABLE bit value must be 0). | 0 |
| 0 | EP0_DMA_DISABLE | 0 | Control endpoint OUT (DMA cannot be enabled for this endpoint and the EP0_DMA_DISABLE bit value must be 0). | 0 |

### 9.7.8 USB DMA Interrupt Status register (USBDMAIntSt - 0xE010 0290)

Each bit of this register reflects whether any of the 32 bits in the corresponding interrupt status register are set. USBDMAIntSt is a read only register.

Table 158. USB DMA Interrupt Status register (USBDMAIntSt - address 0xE010 0290) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:3 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | ERR | | System Error Interrupt bit. | 0 |
| | | 0 | All bits in the USBSysErrIntSt register are 0. | |
| | | 1 | At least one bit in the USBSysErrIntSt is set. | |
| 1 | NDDR | | New DD Request Interrupt bit. | 0 |
| | | 0 | All bits in the USBNDDRIntSt register are 0. | |
| | | 1 | At least one bit in the USBNDDRIntSt is set. | |
| 0 | EOT | | End of Transfer Interrupt bit. | 0 |
| | | 0 | All bits in the USBEoTIntSt register are 0. | |
| | | 1 | At least one bit in the USBEoTIntSt is set. | |

### 9.7.9 USB DMA Interrupt Enable register (USBDMAIntEn - 0xE010 0294)

Writing a one to a bit in this register enables the corresponding bit in USBDMAIntSt to generate an interrupt on the USB_INT_REQ_DMA interrupt line when set. USBDMAIntEn is a read/write register.

**Table 159. USB DMA Interrupt Enable register (USBDMAIntEn - address 0xE010 0294) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:3 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | ERR | | System Error Interrupt enable bit. | 0 |
| | | 0 | The System Error Interrupt is disabled. | |
| | | 1 | The System Error Interrupt is enabled. | |
| 1 | NDDR | | New DD Request Interrupt enable bit. | 0 |
| | | 0 | The New DD Request Interrupt is disabled. | |
| | | 1 | The New DD Request Interrupt is enabled. | |
| 0 | EOT | | End of Transfer Interrupt enable bit. | 0 |
| | | 0 | The End of Transfer Interrupt is disabled. | |
| | | 1 | The End of Transfer Interrupt is enabled. | |

## 9.7.10 USB End of Transfer Interrupt Status register (USBEoTIntSt - 0xE010 02A0)

When the DMA transfer completes for the current DMA descriptor, either normally (descriptor is retired) or because of an error, the bit corresponding to the endpoint is set in this register. The cause of the interrupt is recorded in the DD_status field of the descriptor. USBEoTIntSt is a read only register.

**Table 160. USB End of Transfer Interrupt Status register (USBEoTIntSt - address 0xE010 02A0s) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Endpoint xx ($2 \leq xx \leq 31$) End of Transfer Interrupt request. | 0 |
| | | 0 | There is no End of Transfer interrupt request for endpoint xx. | |
| | | 1 | There is an End of Transfer Interrupt request for endpoint xx. | |

## 9.7.11 USB End of Transfer Interrupt Clear register (USBEoTIntClr - 0xE010 02A4)

Writing one to a bit in this register clears the corresponding bit in the USBEoTIntSt register. Writing zero has no effect. USBEoTIntClr is a write only register.

**Table 161. USB End of Transfer Interrupt Clear register (USBEoTIntClr - address 0xE010 02A4) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Clear endpoint xx ($2 \leq xx \leq 31$) End of Transfer Interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Clear the EPxx End of Transfer Interrupt request in the USBEoTIntSt register. | |

### 9.7.12 USB End of Transfer Interrupt Set register (USBEoTIntSet - 0xE010 02A8)

Writing one to a bit in this register sets the corresponding bit in the USBEoTIntSt register. Writing zero has no effect. USBEoTIntSet is a write only register.

**Table 162. USB End of Transfer Interrupt Set register (USBEoTIntSet - address 0xE010 02A8) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Set endpoint xx (2 ≤ xx ≤ 31) End of Transfer Interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Set the EPxx End of Transfer Interrupt request in the USBEoTIntSt register. | |

### 9.7.13 USB New DD Request Interrupt Status register (USBNDDRIntSt - 0xE010 02AC)

A bit in this register is set when a transfer is requested from the USB device and no valid DD is detected for the corresponding endpoint. USBNDDRIntSt is a read only register.

**Table 163. USB New DD Request Interrupt Status register (USBNDDRIntSt - address 0xE010 02AC) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Endpoint xx (2 ≤ xx ≤ 31) new DD interrupt request. | 0 |
| | | 0 | There is no new DD interrupt request for endpoint xx. | |
| | | 1 | There is a new DD interrupt request for endpoint xx. | |

### 9.7.14 USB New DD Request Interrupt Clear register (USBNDDRIntClr - 0xE010 02B0)

Writing one to a bit in this register clears the corresponding bit in the USBNDDRIntSt register. Writing zero has no effect. USBNDDRIntClr is a write only register.

**Table 164. USB New DD Request Interrupt Clear register (USBNDDRIntClr - address 0xE010 02B0) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Clear endpoint xx (2 ≤ xx ≤ 31) new DD interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Clear the EPxx new DD interrupt request in the USBNDDRIntSt register. | |

### 9.7.15 USB New DD Request Interrupt Set register (USBNDDRIntSet - 0xE010 02B4)

Writing one to a bit in this register sets the corresponding bit in the USBNDDRIntSt register. Writing zero has no effect. USBNDDRIntSet is a write only register

Table 165. USB New DD Request Interrupt Set register (USBNDDRIntSet - address 0xE010 02B4) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Set endpoint xx (2 ≤ xx ≤ 31) new DD interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Set the EPxx new DD interrupt request in the USBNDDRIntSt register. | |

## 9.7.16 USB System Error Interrupt Status register (USBSysErrIntSt - 0xE010 02B8)

If a system error (AHB bus error) occurs when transferring the data or when fetching or updating the DD the corresponding bit is set in this register. USBSysErrIntSt is a read only register.

Table 166. USB System Error Interrupt Status register (USBSysErrIntSt - address 0xE010 02B8) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Endpoint xx (2 ≤ xx ≤ 31) System Error Interrupt request. | 0 |
| | | 0 | There is no System Error Interrupt request for endpoint xx. | |
| | | 1 | There is a System Error Interrupt request for endpoint xx. | |

## 9.7.17 USB System Error Interrupt Clear register (USBSysErrIntClr - 0xE010 02BC)

Writing one to a bit in this register clears the corresponding bit in the USBSysErrIntSt register. Writing zero has no effect. USBSysErrIntClr is a write only register.

Table 167. USB System Error Interrupt Clear register (USBSysErrIntClr - address 0xE010 02BC) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Clear endpoint xx (2 ≤ xx ≤ 31) System Error Interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Clear the EPxx System Error Interrupt request in the USBSysErrIntSt register. | |

## 9.7.18 USB System Error Interrupt Set register (USBSysErrIntSet - 0xE010  02C0)

Writing one to a bit in this register sets the corresponding bit in the USBSysErrIntSt register. Writing zero has no effect. USBSysErrIntSet is a write only register.

Table 168. USB System Error Interrupt Set register (USBSysErrIntSet - address 0xE010 02C0) bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | EPxx | | Set endpoint xx (2 ≤ xx ≤ 31) System Error Interrupt request. | 0 |
| | | 0 | No effect. | |
| | | 1 | Set the EPxx System Error Interrupt request in the USBSysErrIntSt register. | |

## 10. Interrupt handling

This section describes how an interrupt event on any of the endpoints is routed to the Nested Vectored Interrupt Controller (NVIC). For a diagram showing interrupt event handling, see Figure 13–43.

All non-isochronous OUT endpoints (control, bulk, and interrupt endpoints) generate an interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet has been successfully transmitted or when a NAK signal is sent and interrupts on NAK are enabled by the SIE Set Mode command, see Section 13–11.3. For isochronous endpoints, a frame interrupt is generated every 1 ms.

The interrupt handling is different for Slave and DMA mode.

### Slave mode

If an interrupt event occurs on an endpoint and the endpoint interrupt is enabled in the USBEpIntEn register, the corresponding status bit in the USBEpIntSt is set. For non-isochronous endpoints, all endpoint interrupt events are divided into two types by the corresponding USBEpIntPri[n] registers: fast endpoint interrupt events and slow endpoint interrupt events. All fast endpoint interrupt events are ORed and routed to bit EP_FAST in the USBDevIntSt register. All slow endpoint interrupt events are ORed and routed to the EP_SLOW bit in USBDevIntSt.

For isochronous endpoints, the FRAME bit in USBDevIntSt is set every 1 ms.

The USBDevIntSt register holds the status of all endpoint interrupt events as well as the status of various other interrupts (see Section 13–9.2.1). By default, all interrupts (if enabled in USBDevIntEn) are routed to the USB_INT_REQ_LP bit in the USBIntSt register to request low priority interrupt handling. However, the USBDevIntPri register can route either the FRAME or the EP_FAST bit to the USB_INT_REQ_HP bit in the USBIntSt register.

Only one of the EP_FAST and FRAME interrupt events can be routed to the USB_INT_REQ_HP bit. If routing both bits to USB_INT_REQ_HP is attempted, both interrupt events are routed to USB_INT_REQ_LP.

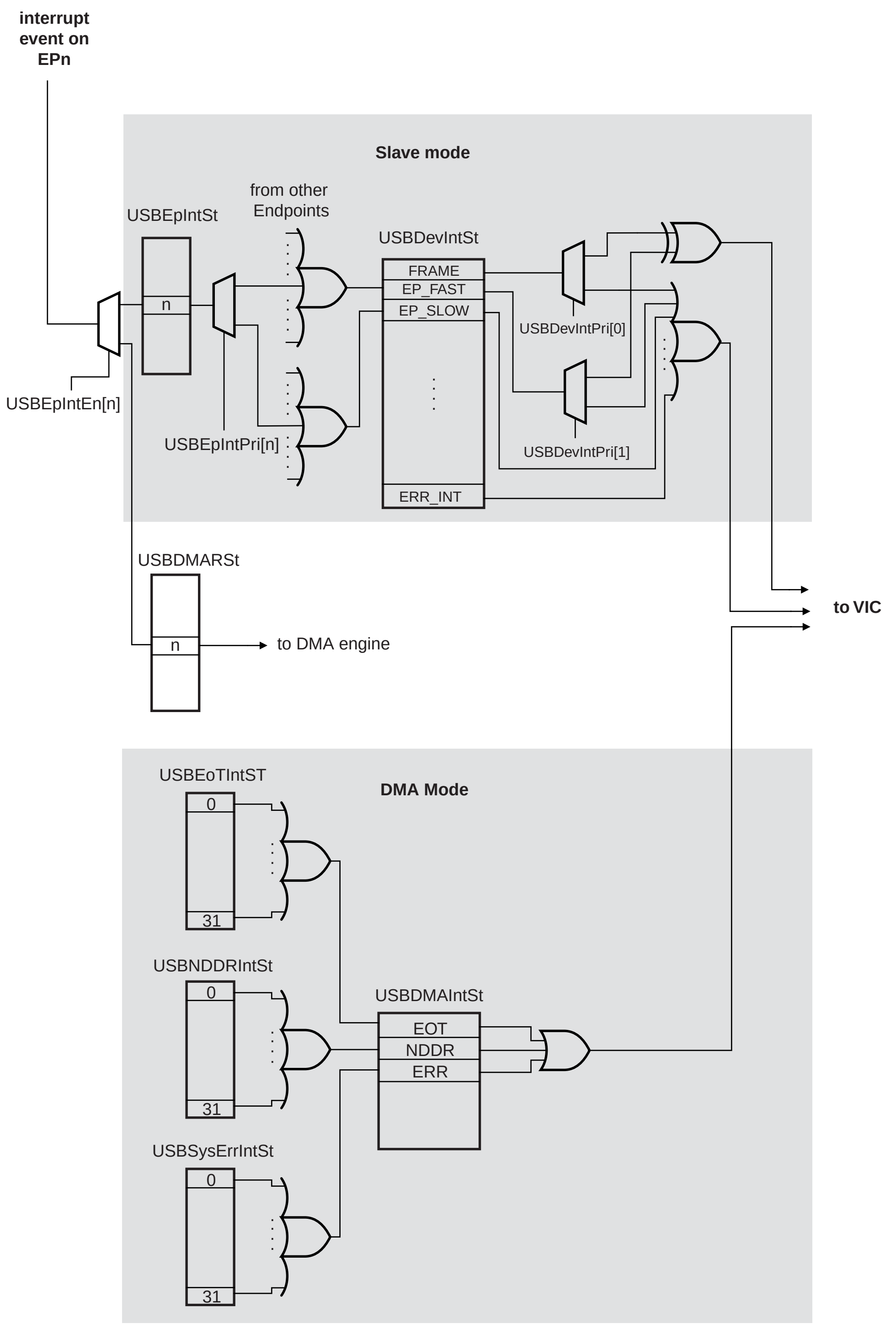
Slow endpoint interrupt events are always routed directly to the USB_INT_REQ_LP bit for low priority interrupt handling by software. The USB_INT_REQ_LP and USB_INT_REQ_HP are routed separately to the VIC (see Table 9–88).

### DMA mode

If an interrupt event occurs on a non-control endpoint and the endpoint interrupt is not enabled in the USBEpIntEn register, the corresponding status bit in the USBDMARSt is set by hardware. This serves as a flag for the DMA engine to transfer data if DMA transfer is enabled for the corresponding endpoint in the USBEpDMASt register.

Three types of interrupts can occur for each endpoint for data transfers in DMA mode: End of transfer interrupt, new DD request interrupt, and system error interrupt. These interrupt events set a bit for each endpoint in the respective registers USBEoTIntSt, USBNDDRIntSt, and USBSysErrIntSt. The End of transfer interrupts from all endpoints are then Ored and routed to the EOT bit in USBDMAIntSt. Likewise, all New DD request interrupts and system error interrupt events are routed to the NDDR and ERR bits respectively in the USBDMAStInt register.

The EOT, NDDR, and ERR bits (if enabled in USBDMAIntEn) are ORed to provide the USB device DMA interrup in the VIC (see Table 9–88).



For simplicity, USBDevIntEn and USBDMAIntEn are not shown.

**Fig 43. Interrupt event handling**

## 11. Serial interface engine command description

The functions and registers of the Serial Interface Engine (SIE) are accessed using commands, which consist of a command code followed by optional data bytes (read or write action). The USBCmdCode (Table 13–148) and USBCmdData (Table 13–149) registers are used for these accesses.

A complete access consists of two phases:

1. **Command phase:** the USBCmdCode register is written with the CMD_PHASE field set to the value 0x05 (Command), and the CMD_CODE field set to the desired command code. On completion of the command, the CCEMPTY bit of USBDevIntSt is set.
2. **Data phase (optional):** for writes, the USBCmdCode register is written with the CMD_PHASE field set to the value 0x01 (Write), and the CMD_WDATA field set with the desired write data. On completion of the write, the CCEMPTY bit of USBDevIntSt is set. For reads, USBCmdCode register is written with the CMD_PHASE field set to the value 0x02 (Read), and the CMD_CODE field set with command code the read corresponds to. On completion of the read, the CDFULL bit of USBDevInSt will be set, indicating the data is available for reading in the USBCmdData register. In the case of multi-byte registers, the least significant byte is accessed first.

An overview of the available commands is given in Table 13–169.

Here is an example of the Read Current Frame Number command (reading 2 bytes):

```
USBDevIntClr = 0x30;           // Clear both CCEMPTY & CDFULL
USBCmdCode = 0x00F50500;       // CMD_CODE=0xF5, CMD_PHASE=0x05(Command)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY interrupt bit.
USBCmdCode = 0x00F50200;       // CMD_CODE=0xF5, CMD_PHASE=0x02(Read)
while (!(USBDevIntSt & 0x20)); // Wait for CDFULL.
USBDevIntClr = 0x20;           // Clear CDFULL.
CurFrameNum = USBCmdData;      // Read Frame number LSB byte.
USBCmdCode = 0x00F50200;       // CMD_CODE=0xF5, CMD_PHASE=0x02(Read)
while (!(USBDevIntSt & 0x20)); // Wait for CDFULL.
Temp = USBCmdData;             // Read Frame number MSB byte
USBDevIntClr = 0x20;           // Clear CDFULL interrupt bit.
CurFrameNum = CurFrameNum | (Temp << 8);
```

Here is an example of the Set Address command (writing 1 byte):

```
USBDevIntClr = 0x10;           // Clear CCEMPTY.
USBCmdCode = 0x00D00500;       // CMD_CODE=0xD0, CMD_PHASE=0x05(Command)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY.
USBCmdCode = 0x008A0100;       // CMD_WDATA=0x8A(DEV_EN=1, DEV_ADDR=0xA),
                               // CMD_PHASE=0x01(Write)
while (!(USBDevIntSt & 0x10)); // Wait for CCEMPTY.
USBDevIntClr = 0x10;           // Clear CCEMPTY.
```

Table 169. SIE command code table

| Command name | Recipient | Code (Hex) | Data phase |
|---|---|---|---|
| **Device commands** | | | |
| Set Address | Device | D0 | Write 1 byte |
| Configure Device | Device | D8 | Write 1 byte |
| Set Mode | Device | F3 | Write 1 byte |
| Read Current Frame Number | Device | F5 | Read 1 or 2 bytes |
| Read Test Register | Device | FD | Read 2 bytes |
| Set Device Status | Device | FE | Write 1 byte |
| Get Device Status | Device | FE | Read 1 byte |
| Get Error Code | Device | FF | Read 1 byte |
| Read Error Status | Device | FB | Read 1 byte |
| **Endpoint Commands** | | | |
| Select Endpoint | Endpoint 0 | 00 | Read 1 byte (optional) |
| | Endpoint 1 | 01 | Read 1 byte (optional) |
| | Endpoint xx | xx | Read 1 byte (optional) |
| Select Endpoint/Clear Interrupt | Endpoint 0 | 40 | Read 1 byte |
| | Endpoint 1 | 41 | Read 1 byte |
| | Endpoint xx | xx + 40 | Read 1 byte |
| Set Endpoint Status | Endpoint 0 | 40 | Write 1 byte |
| | Endpoint 1 | 41 | Write 1 byte |
| | Endpoint xx | xx + 40 | Write 1 byte |
| Clear Buffer | Selected Endpoint | F2 | Read 1 byte (optional) |
| Validate Buffer | Selected Endpoint | FA | None |

## 11.1 Set Address (Command: 0xD0, Data: write 1 byte)

The Set Address command is used to set the USB assigned address and enable the (embedded) function. The address set in the device will take effect after the status stage of the control transaction. After a bus reset, DEV_ADDR is set to 0x00, and DEV_EN is set to 1. The device will respond to packets for function address 0x00, endpoint 0 (default endpoint).

Table 170. Device Set Address Register bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7 | DEV_EN | Device Enable. After a bus reset this bit is set to 1.<br>0: Device will not respond to any packets.<br>1: Device will respond to packets for function address DEV_ADDR. | 0 |
| 6:0 | DEV_ADDR | Device address set by the software. After a bus reset this field is set to 0x00. | 0x00 |

## 11.2 Configure Device (Command: 0xD8, Data: write 1 byte)

A value of 1 written to the register indicates that the device is configured and all the enabled non-control endpoints will respond. Control endpoints are always enabled and respond even if the device is not configured, in the default state.

**Table 171. Configure Device Register bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:1 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 0 | CONF_DEVICE | Device is configured. All enabled non-control endpoints will respond. This bit is cleared by hardware when a bus reset occurs. When set, the UP_LED signal is driven LOW if the device is not in the suspended state (SUS=0). | |

## 11.3 Set Mode (Command: 0xF3, Data: write 1 byte)

**Table 172. Set Mode Register bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 6 | INAK_BO[2] | | Interrupt on NAK for Bulk OUT endpoints. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed OUT transactions generate interrupts. | |
| 5 | INAK_BI | | Interrupt on NAK for Bulk IN endpoints. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed IN transactions generate interrupts. | |
| 4 | INAK_IO[1] | | Interrupt on NAK for Interrupt OUT endpoints. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed OUT transactions generate interrupts. | |
| 3 | INAK_II | | Interrupt on NAK for Interrupt IN endpoint. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed IN transactions generate interrupts. | |
| 2 | INAK_CO | | Interrupt on NAK for Control OUT endpoint. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed OUT transactions generate interrupts. | |
| 1 | INAK_CI | | Interrupt on NAK for Control IN endpoint. | 0 |
| | | 0 | Only successful transactions generate an interrupt. | |
| | | 1 | Both successful and NAKed IN transactions generate interrupts. | |
| 0 | AP_CLK | | Always PLL Clock. | 0 |
| | | 0 | USB_NEED_CLK is functional; the 48 MHz clock can be stopped when the device enters suspend state. | |
| | | 1 | USB_NEED_CLK is fixed to 1; the 48 MHz clock cannot be stopped when the device enters suspend state. | |

[1] This bit should be reset to 0 if the DMA is enabled for any of the Interrupt OUT endpoints.

[2] This bit should be reset to 0 if the DMA is enabled for any of the Bulk OUT endpoints.

### 11.4 Read Current Frame Number (Command: 0xF5, Data: read 1 or 2 bytes)

Returns the frame number of the last successfully received SOF. The frame number is eleven bits wide. The frame number returns least significant byte first. In case the user is only interested in the lower 8 bits of the frame number, only the first byte needs to be read.

- In case no SOF was received by the device at the beginning of a frame, the frame number returned is that of the last successfully received SOF.
- In case the SOF frame number contained a CRC error, the frame number returned will be the corrupted frame number as received by the device.

### 11.5 Read Test Register (Command: 0xFD, Data: read 2 bytes)

The test register is 16 bits wide. It returns the value of 0xA50F if the USB clocks (usbclk and AHB slave clock) are running.

### 11.6 Set Device Status (Command: 0xFE, Data: write 1 byte)

The Set Device Status command sets bits in the Device Status Register.

**Table 173. Set Device Status Register bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7:5 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | RST | | Bus Reset bit. On a bus reset, the device will automatically go to the default state. In the default state:<br>• Device is unconfigured.<br>• Will respond to address 0.<br>• Control endpoint will be in the Stalled state.<br>• All endpoints are unrealized except control endpoints EP0 and EP1.<br>• Data toggling is reset for all endpoints.<br>• All buffers are cleared.<br>• There is no change to the endpoint interrupt status.<br>• DEV_STAT interrupt is generated.<br>Note: Bus resets are ignored when the device is not connected (CON=0). | 0 |
| | | 0 | This bit is cleared when read. | |
| | | 1 | This bit is set when the device receives a bus reset. A DEV_STAT interrupt is generated. | |

Table 173. Set Device Status Register bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 3 | SUS_CH | | Suspend (SUS) bit change indicator. The SUS bit can toggle because:<br>• The device goes into the suspended state.<br>• The device is disconnected.<br>• The device receives resume signalling on its upstream port.<br>This bit is cleared when read. | 0 |
| | | 0 | SUS bit not changed. | |
| | | 1 | SUS bit changed. At the same time a DEV_STAT interrupt is generated. | |
| 2 | SUS | | Suspend: The Suspend bit represents the current suspend state. When the device is suspended (SUS = 1) and the CPU writes a 0 into it, the device will generate a remote wakeup. This will only happen when the device is connected (CON = 1). When the device is not connected or not suspended, writing a 0 has no effect. Writing a 1 to this bit has no effect. | 0 |
| | | 0 | This bit is reset to 0 on any activity. | |
| | | 1 | This bit is set to 1 when the device hasn't seen any activity on its upstream port for more than 3 ms. | |
| 1 | CON_CH | | Connect Change. | 0 |
| | | 0 | This bit is cleared when read. | |
| | | 1 | This bit is set when the device's pull-up resistor is disconnected because $V_{BUS}$ disappeared. The DEV_STAT interrupt is generated when this bit is 1. | |
| 0 | CON | | The Connect bit indicates the current connect status of the device. It controls the CONNECT output pin, used for SoftConnect. Reading the connect bit returns the current connect status. This bit is cleared by hardware when the $V_{BUS}$ status input is LOW for more than 3 ms. The 3 ms delay filters out temporary dips in the $V_{BUS}$ voltage. | 0 |
| | | 0 | Writing a 0 will make the CONNECT pin go HIGH. | |
| | | 1 | Writing a 1 will make the CONNECT pin go LOW.. | |

**Remark:** Once software is ready to respond to USB traffic, it enables the SoftConnect feature by writing a one to the CON bit position, setting AllowConnect. Whenever VBUS is present and CON is high, the SoftConnect pin function (CONNECT) is pulled LOW (if this function is enabled on this pin), and is intended to enable a 1.5 kΩ pull-up resistor on the D+ line. When CON is low, SoftConnect is driven HIGH, disabling the 1.5 kΩ resistor when VBUS goes away. Thus the 1.5 kΩ resistor is only enabled when VBUS is high.

The CON_CH interrupt is generated on a disconnect event, allowing software to take whatever steps necessary when this occurs.

On reconnecting, the host will send a USB reset, causing the RST interrupt to occur, and causing the USB hardware to go to its default state.

## 11.7 Get Device Status (Command: 0xFE, Data: read 1 byte)

The Get Device Status command returns the Device Status Register. Reading the device status returns 1 byte of data. The bit field definition is same as the Set Device Status Register as shown in Table 13–173.

**Remark:** To ensure correct operation, the DEV_STAT bit of USBDevIntSt must be cleared before executing the Get Device Status command.

## 11.8 Get Error Code (Command: 0xFF, Data: read 1 byte)

Different error conditions can arise inside the SIE. The Get Error Code command returns the last error code that occurred. The 4 least significant bits form the error code.

**Table 174. Get Error Code Register bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7:5 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | EA | - | The Error Active bit will be reset once this register is read. | |
| 3:0 | EC | | Error Code. | 0x0 |
| | | 0000 | No Error. | |
| | | 0001 | PID Encoding Error. | |
| | | 0010 | Unknown PID. | |
| | | 0011 | Unexpected Packet - any packet sequence violation from the specification. | |
| | | 0100 | Error in Token CRC. | |
| | | 0101 | Error in Data CRC. | |
| | | 0110 | Time Out Error. | |
| | | 0111 | Babble. | |
| | | 1000 | Error in End of Packet. | |
| | | 1001 | Sent/Received NAK. | |
| | | 1010 | Sent Stall. | |
| | | 1011 | Buffer Overrun Error. | |
| | | 1100 | Sent Empty Packet (ISO Endpoints only). | |
| | | 1101 | Bitstuff Error. | |
| | | 1110 | Error in Sync. | |
| | | 1111 | Wrong Toggle Bit in Data PID, ignored data. | |

## 11.9 Read Error Status (Command: 0xFB, Data: read 1 byte)

This command reads the 8-bit Error register from the USB device. This register records which error events have recently occurred in the SIE. If any of these bits are set, the ERR_INT bit of USBDevIntSt is set. The error bits are cleared after reading this register.

Table 175. Read Error Status Register bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7 | TGL_ERR | Wrong toggle bit in data PID, ignored data. | 0 |
| 6 | BTSTF | Bit stuff error. | 0 |
| 5 | B_OVRN | Buffer Overrun. | 0 |
| 4 | EOP | End of packet error. | 0 |
| 3 | TIMEOUT | Time out error. | 0 |
| 2 | DCRC | Data CRC error. | 0 |
| 1 | UEPKT | Unexpected Packet - any packet sequence violation from the specification. | 0 |
| 0 | PID_ERR | PID encoding error or Unknown PID or Token CRC. | 0 |

## 11.10 Select Endpoint (Command: 0x00 - 0x1F, Data: read 1 byte (optional))

The Select Endpoint command initializes an internal pointer to the start of the selected buffer in EP_RAM. Optionally, this command can be followed by a data read, which returns some additional information on the packet(s) in the endpoint buffer(s). The command code of the Select Endpoint command is equal to the physical endpoint number. In the case of a single buffered endpoint the B_2_FULL bit is not valid.

Table 176. Select Endpoint Register bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 6 | B_2_FULL | | The buffer 2 status. | 0 |
| | | 0 | Buffer 2 is empty. | |
| | | 1 | Buffer 2 is full. | |
| 5 | B_1_FULL | | The buffer 1 status. | 0 |
| | | 0 | Buffer 1 is empty. | |
| | | 1 | Buffer 1 is full. | |
| 4 | EPN | | EP NAKed bit indicates sending of a NAK. If the host sends an OUT packet to a filled OUT buffer, the device returns NAK. If the host sends an IN token packet to an empty IN buffer, the device returns NAK. | 0 |
| | | 0 | The EPN bit is reset after the device has sent an ACK after an OUT packet or when the device has seen an ACK after sending an IN packet. | |
| | | 1 | The EPN bit is set when a NAK is sent and the interrupt on NAK feature is enabled. | |
| 3 | PO | | Packet over-written bit. | 0 |
| | | 0 | The PO bit is cleared by the ‘Select Endpoint/Clear Interrupt’ command. | |
| | | 1 | The previously received packet was over-written by a SETUP packet. | |

Table 176. Select Endpoint Register bit description

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 2 | STP | | SETUP bit: the value of this bit is updated after each successfully received packet (i.e. an ACKed package on that particular physical endpoint). | 0 |
| | | 0 | The STP bit is cleared by doing a Select Endpoint/Clear Interrupt on this endpoint. | |
| | | 1 | The last received packet for the selected endpoint was a SETUP packet. | |
| 1 | ST | | Stalled endpoint indicator. | 0 |
| | | 0 | The selected endpoint is not stalled. | |
| | | 1 | The selected endpoint is stalled. | |
| 0 | FE | | Full/Empty. This bit indicates the full or empty status of the endpoint buffer(s). For IN endpoints, the FE bit gives the ANDed result of the B_1_FULL and B_2_FULL bits. For OUT endpoints, the FE bit gives ORed result of the B_1_FULL and B_2_FULL bits. For single buffered endpoints, this bit simply reflects the status of B_1_FULL. | 0 |
| | | 0 | For an IN endpoint, at least one write endpoint buffer is empty. | |
| | | 1 | For an OUT endpoint, at least one endpoint read buffer is full. | |

## 11.11 Select Endpoint/Clear Interrupt (Command: 0x40 - 0x5F, Data: read 1 byte)

Commands 0x40 to 0x5F are identical to their Select Endpoint equivalents, with the following differences:

- They clear the bit corresponding to the endpoint in the USBEpIntSt register.
- In case of a control OUT endpoint, they clear the STP and PO bits in the corresponding Select Endpoint Register.
- Reading one byte is obligatory.

**Remark:** This command may be invoked by using the USBCmdCode and USBCmdData registers, or by setting the corresponding bit in USBEpIntClr. For ease of use, using the USBEpIntClr register is recommended.

## 11.12 Set Endpoint Status (Command: 0x40 - 0x55, Data: write 1 byte (optional))

The Set Endpoint Status command sets status bits 7:5 and 0 of the endpoint. The Command Code of Set Endpoint Status is equal to the sum of 0x40 and the physical endpoint number in hex. Not all bits can be set for all types of endpoints.

**Table 177. Set Endpoint Status Register bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7 | CND_ST | | Conditional Stall bit. | 0 |
| | | 0 | Unstalls both control endpoints. | |
| | | 1 | Stall both control endpoints, unless the STP bit is set in the Select Endpoint register. It is defined only for control OUT endpoints. | |
| 6 | RF_MO | | Rate Feedback Mode. | 0 |
| | | 0 | Interrupt endpoint is in the Toggle mode. | |
| | | 1 | Interrupt endpoint is in the Rate Feedback mode. This means that transfer takes place without data toggle bit. | |
| 5 | DA | | Disabled endpoint bit. | 0 |
| | | 0 | The endpoint is enabled. | |
| | | 1 | The endpoint is disabled. | |
| 4:1 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 0 | ST | | Stalled endpoint bit. A Stalled control endpoint is automatically unstalled when it receives a SETUP token, regardless of the content of the packet. If the endpoint should stay in its stalled state, the CPU can stall it again by setting this bit. When a stalled endpoint is unstalled - either by the Set Endpoint Status command or by receiving a SETUP token - it is also re-initialized. This flushes the buffer: in case of an OUT buffer it waits for a DATA 0 PID; in case of an IN buffer it writes a DATA 0 PID. There is no change of the interrupt status of the endpoint. When already unstalled, writing a zero to this bit initializes the endpoint. When an endpoint is stalled by the Set Endpoint Status command, it is also re-initialized. | 0 |
| | | 0 | The endpoint is unstalled. | |
| | | 1 | The endpoint is stalled. | |

## 11.13 Clear Buffer (Command: 0xF2, Data: read 1 byte (optional))

When an OUT packet sent by the host has been received successfully, an internal hardware FIFO status Buffer_Full flag is set. All subsequent packets will be refused by returning a NAK. When the device software has read the data, it should free the buffer by issuing the Clear Buffer command. This clears the internal Buffer_Full flag. When the buffer is cleared, new packets will be accepted.

When bit 0 of the optional data byte is 1, the previously received packet was over-written by a SETUP packet. The Packet over-written bit is used only in control transfers. According to the USB specification, a SETUP packet should be accepted irrespective of the buffer status. The software should always check the status of the PO bit after reading the SETUP data. If it is set then it should discard the previously read data, clear the PO bit by issuing a Select Endpoint/Clear Interrupt command, read the new SETUP data and again check the status of the PO bit.

See Section 13–13 “Slave mode operation” for a description of when this command is used.

**Table 178. Clear Buffer Register bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7:1 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 0 | PO | | Packet over-written bit. This bit is only applicable to the control endpoint EP0. | 0 |
| | | 0 | The previously received packet is intact. | |
| | | 1 | The previously received packet was over-written by a later SETUP packet. | |

## 11.14 Validate Buffer (Command: 0xFA, Data: none)

When the CPU has written data into an IN buffer, software should issue a Validate Buffer command. This tells hardware that the buffer is ready for sending on the USB bus. Hardware will send the contents of the buffer when the next IN token packet is received.

Internally, there is a hardware FIFO status flag called Buffer_Full. This flag is set by the Validate Buffer command and cleared when the data has been sent on the USB bus and the buffer is empty.

A control IN buffer cannot be validated when its corresponding OUT buffer has the Packet Over-written (PO) bit (see the Clear Buffer Register) set or contains a pending SETUP packet. For the control endpoint the validated buffer will be invalidated when a SETUP packet is received.

See Section 13–13 “Slave mode operation” for a description of when this command is used.

## 12. USB device controller initialization

The LPC29xx USB device controller initialization includes the following steps:

1. Enable the USB device block through the PMU.
2. Configure and enable the USB PLL in the CGU1, see Table 3–15.
3. Enable the device controller clocks by setting DEV_CLK_EN and AHB_CLK_EN bits in the USBClkCtrl register. Poll the respective clock bits in the USBClkSt register until they are set.
4. Enable the USB pin functions by writing to the corresponding port configuration register, see Table 6–58.
5. Disable the pull-up resistor on the USB_VBUS pin using the corresponding port configuration register, see Table 6–59.
6. Set USBEpIn and USBMaxPSize registers for EP0 and EP1, and wait until the EP_RLZED bit in USBDevIntSt is set so that EP0 and EP1 are realized.
7. Enable endpoint interrupts (Slave mode):
   – Clear all endpoint interrupts using USBEpIntClr.
   – Clear any device interrupts using USBDevIntClr.
   – Enable Slave mode for the desired endpoints by setting the corresponding bits in USBEpIntEn.

- Set the priority of each enabled interrupt using USBEpIntPri.
- Configure the desired interrupt mode using the SIE Set Mode command.
- Enable device interrupts using USBDevIntEn (normally DEV_STAT, EP_SLOW, and possibly EP_FAST).

8. Configure the DMA (DMA mode):
   - Disable DMA operation for all endpoints using USBEpDMADis.
   - Clear any pending DMA requests using USBDMARClr.
   - Clear all DMA interrupts using USBEoTIntClr, USBNDDRIntClr, and USBSysErrIntClr.
   - Prepare the UDCA in system memory.
   - Write the desired RAM address for the UDCA to USBUDCAH.
   - Enable the desired endpoints for DMA operation using USBEpDMAEn.
   - Set EOT, DDR, and ERR bits in USBDMAIntEn.
9. Install USB interrupt handler in the VIC by writing its address to the appropriate vector table location and enabling the USB interrupt in the VIC.
10. Set default USB address to 0x0 and DEV_EN to 1 using the SIE Set Address command. A bus reset will also cause this to happen.
11. Set CON bit to 1 to make CONNECT active using the SIE Set Device Status command.

The configuration of the endpoints varies depending on the software application. By default, all the endpoints are disabled except control endpoints EP0 and EP1. Additional endpoints are enabled and configured by software after a SET_CONFIGURATION or SET_INTERFACE device request is received from the host.

## 13. Slave mode operation

In Slave mode, the CPU transfers data between RAM and the endpoint buffer using the Register Interface.

### 13.1 Interrupt generation

In slave mode, data packet transfer between RAM and an endpoint buffer can be initiated in response to an endpoint interrupt. Endpoint interrupts are enabled using the USBEpIntEn register, and are observable in the USBEpIntSt register.

All non-isochronous OUT endpoints generate an endpoint interrupt when they receive a packet without an error. All non-isochronous IN endpoints generate an interrupt when a packet is successfully transmitted, or when a NAK handshake is sent on the bus and the interrupt on NAK feature is enabled.

For Isochronous endpoints, transfer of data is done when the FRAME interrupt (in USBDevIntSt) occurs.

### 13.2 Data transfer for OUT endpoints

When the software wants to read the data from an endpoint buffer it should set the RD_EN bit and program LOG_ENDPOINT with the desired endpoint number in the USBCtrl register. The control logic will fetch the packet length to the USBRxPLen register, and set the PKT_RDY bit (Table 13–144 ).

Software can now start reading the data from the USBRxData register (Table 13–143). When the end of packet is reached, the RD_EN bit is cleared, and the RxENDPKT bit is set in the USBDevSt register. Software now issues a Clear Buffer (refer to Table 13–178) command. The endpoint is now ready to accept the next packet. For OUT isochronous endpoints, the next packet will be received irrespective of whether the buffer has been cleared. Any data not read from the buffer before the end of the frame is lost. See Section 13–15 “Double buffered endpoint operation” for more details.

If the software clears RD_EN before the entire packet is read, reading is terminated, and the data remains in the endpoint’s buffer. When RD_EN is set again for this endpoint, the data will be read from the beginning.

### 13.3 Data transfer for IN endpoints

When writing data to an endpoint buffer, WR_EN (Section 13–9.5.5 “USB Control register (USBCtrl - 0xE010 0228)”) is set and software writes to the number of bytes it is going to send in the packet to the USBTxPLen register (Section 13–9.5.4). It can then write data continuously in the USBTxData register.

When the the number of bytes programmed in USBTxPLen have been written to USBTxData, the WR_EN bit is cleared, and the TxENDPKT bit is set in the USBDevIntSt register. Software issues a Validate Buffer (Section 13–11.14 “Validate Buffer (Command: 0xFA, Data: none)”) command. The endpoint is now ready to send the packet. For IN isochronous endpoints, the data in the buffer will be sent only if the buffer is validated before the next FRAME interrupt occurs; otherwise, an empty packet will be sent in the next frame. If the software clears WR_EN before the entire packet is written, writing will start again from the beginning the next time WR_EN is set for this endpoint.

Both RD_EN and WR_EN can be high at the same time for the same logical endpoint. Interleaved read and write operation is possible.

## 14. DMA operation

In DMA mode, the DMA transfers data between RAM and the endpoint buffer.

The following sections discuss DMA mode operation. Background information is given in sections Section 13–14.2 “USB device communication area” and Section 13–14.3 “Triggering the DMA engine”. The fields of the DMA Descriptor are described in section Section 13–14.4 “The DMA descriptor”. The last three sections describe DMA operation: Section 13–14.5 “Non-isochronous endpoint operation”, Section 13–14.6 “Isochronous endpoint operation”, and Section 13–14.7 “Auto Length Transfer Extraction (ATLE) mode operation”.

### 14.1 Transfer terminology

Within this section three types of transfers are mentioned:

1. USB transfers – transfer of data over the USB bus. The USB 2.0 specification refers to these simply as transfers. Within this section they are referred to as USB transfers to distinguish them from DMA transfers. A USB transfer is composed of transactions. Each transaction is composed of packets.
2. DMA transfers – the transfer of data between an endpoint buffer and system memory (RAM).
3. Packet transfers – in this section, a packet transfer refers to the transfer of a packet of data between an endpoint buffer and system memory (RAM). A DMA transfer is composed of one or more packet transfers.

## 14.2 USB device communication area

The CPU and DMA controller communicate through a common area of memory, called the USB Device Communication Area, or UDCA. The UDCA is a 32-word array of DMA Descriptor Pointers (DDPs), each of which corresponds to a physical endpoint. Each DDP points to the start address of a DMA Descriptor, if one is defined for the endpoint. DDPs for unrealized endpoints and endpoints disabled for DMA operation are ignored and can be set to a NULL (0x0) value.

The start address of the UDCA is stored in the USBUDCAH register. The UDCA can reside at any 128-byte boundary of RAM that is accessible to both the CPU and DMA controller.

Figure 13–44 illustrates the UDCA and its relationship to the UDCA Head (USBUDCAH) register and DMA Descriptors.

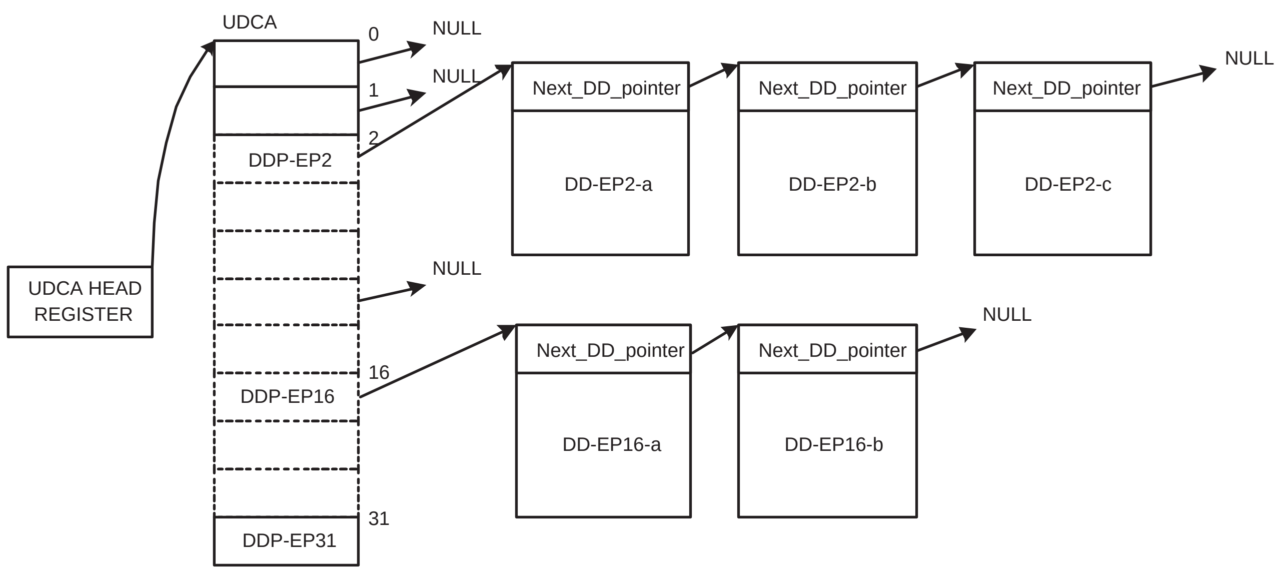


Fig 44. UDCA Head register and DMA Descriptors

## 14.3 Triggering the DMA engine

An endpoint raises a DMA request when Slave mode is disabled by setting the corresponding bit in the USBEpIntEn register to 0 (Section 13–9.3.2) and an endpoint interrupt occurs (see Section 13–9.7.1 “USB DMA Request Status register (USBDMARSt - 0xE010 0250)”).

A DMA transfer for an endpoint starts when the endpoint is enabled for DMA operation in USBEpDMASt, the corresponding bit in USBDMARSt is set, and a valid DD is found for the endpoint.

All endpoints share a single DMA channel to minimize hardware overhead. If more than one DMA request is active in USBDMARSt, the endpoint with the lowest physical endpoint number is processed first.

In DMA mode, the bits corresponding to Interrupt on NAK for Bulk OUT and Interrupt OUT endpoints (INAK_BO and INAK_IO) should be set to 0 using the SIE Set Mode command (Section 13–11.3).

## 14.4 The DMA descriptor

DMA transfers are described by a data structure called the DMA Descriptor (DD).

DDs are placed in RAM. These descriptors can be located anywhere in the on-chip RAM at word-aligned addresses.

DDs for non-isochronous endpoints are four words long. DDs for isochronous endpoints are five words long.

The parameters associated with a DMA transfer are:

- The start address of the DMA buffer
- The length of the DMA buffer
- The start address of the next DMA descriptor
- Control information
- Count information (number of bytes transferred)
- Status information

Table 13–179 lists the DMA descriptor fields.

Table 179. DMA descriptor

| Word position | Access (H/W) | Access (S/W) | Bit position | Description |
|---|---|---|---|---|
| 0 | R | R/W | 31:0 | Next_DD_pointer (RAM address) |

**Table 179. DMA descriptor**

| Word position | Access (H/W) | Access (S/W) | Bit position | Description |
|---|---|---|---|---|
| 1 | R | R/W | 1:0 | DMA_mode (00 -Normal; 01 - ATLE) |
| | R | R/W | 2 | Next_DD_valid (1 - valid; 0 - invalid) |
| | - | - | 3 | Reserved |
| | R | R/W | 4 | Isochronous_endpoint (1 - isochronous; 0 - non-isochronous) |
| | R | R/W | 15:5 | Max_packet_size |
| | R/W[1] | R/W | 31:16 | DMA_buffer_length<br>This value is specified in bytes for non-isochronous endpoints and in number of packets for isochronous endpoints. |
| 2 | R/W | R/W | 31:0 | DMA_buffer_start_addr |
| 3 | R/W | R/I | 0 | DD_retired (To be initialized to 0) |
| | W | R/I | 4:1 | DD_status (To be initialized to 0000):<br>0000 - NotServiced<br>0001 - BeingServiced<br>0010 - NormalCompletion<br>0011 - DataUnderrun (short packet)<br>1000 - DataOverrun<br>1001 - SystemError |
| | W | R/I | 5 | Packet_valid (To be initialized to 0) |
| | W | R/I | 6 | LS_byte_extracted (ATLE mode) (To be initialized to 0) |
| | W | R/I | 7 | MS_byte_extracted (ATLE mode) (To be initialized to 0) |
| | R | W | 13:8 | Message_length_position (ATLE mode) |
| | - | - | 15:14 | Reserved |
| | R/W | R/I | 31:16 | Present_DMA_count (To be initialized to 0) |
| 4 | R/W | R/W | 31:0 | Isochronous_packetsize_memory_address |

[1] Write only in ATLE mode

Legend: R - Read; W - Write; I - Initialize

### 14.4.1 Next_DD_pointer

Pointer to the memory location from where the next DMA descriptor will be fetched.

### 14.4.2 DMA_mode

Specifies the DMA mode of operation. Two modes have been defined: Normal and Automatic Transfer Length Extraction (ATLE) mode. In normal mode, software initializes the DMA_buffer_length for OUT endpoints. In ATLE mode, the DMA_buffer_length is extracted from the incoming data. See Section 13–14.7 “Auto Length Transfer Extraction (ATLE) mode operation” on page 206 for more details.

### 14.4.3 Next_DD_valid

This bit indicates whether the software has prepared the next DMA descriptor. If set, the DMA engine fetches the new descriptor when it is finished with the current one.

### 14.4.4 Isochronous_endpoint

When set, this bit indicates that the descriptor belongs to an isochronous endpoint. Hence 5 words have to be read when fetching it.

### 14.4.5 Max_packet_size

The maximum packet size of the endpoint. This parameter is used while transferring the data for IN endpoints from the memory. It is used for OUT endpoints to detect the short packet. This is applicable to non-isochronous endpoints only. This field should be set to the same MPS value that is assigned for the endpoint using the USBMaxPSize register.

### 14.4.6 DMA_buffer_length

This indicates the depth of the DMA buffer allocated for transferring the data. The DMA engine will stop using this descriptor when this limit is reached and will look for the next descriptor.

In Normal mode operation, software sets this value for both IN and OUT endpoints. In ATLE mode operation, software sets this value for IN endpoints only. For OUT endpoints, hardware sets this value using the extracted length of the data stream.

For isochronous endpoints, DMA_buffer_length is specified in number of packets, for non-isochronous endpoints in bytes.

### 14.4.7 DMA_buffer_start_addr

The address where the data is read from or written to. This field is updated each time the DMA engine finishes transferring a packet.

### 14.4.8 DD_retired

This bit is set by hardware when the DMA engine finishes the current descriptor. This happens when the end of the buffer is reached, a short packet is transferred (non-isochronous endpoints), or an error condition is detected.

### 14.4.9 DD_status

The status of the DMA transfer is encoded in this field. The following codes are defined:

- **NotServiced** - No packet has been transferred yet.
- **BeingServiced** - At least one packet is transferred.
- **NormalCompletion** - The DD is retired because the end of the buffer is reached and there were no errors. The DD_retired bit is also set.
- **DataUnderrun** - Before reaching the end of the DMA buffer, the USB transfer is terminated because a short packet is received. The DD_retired bit is also set.
- **DataOverrun** - The end of the DMA buffer is reached in the middle of a packet transfer. This is an error situation. The DD_retired bit is set. The present DMA count field is equal to the value of DMA_buffer_length. The packet must be re-transmitted from the endpoint buffer in another DMA transfer. The corresponding EPxx_DMA_ENABLE bit in USBEpDMASt is cleared.
- **SystemError** - The DMA transfer being serviced is terminated because of an error on the AHB bus. The DD_retired bit is not set in this case. The corresponding EPxx_DMA_ENABLE in USBEpDMASt is cleared. Since a system error can happen while updating the DD, the DD fields in RAM may be unreliable.

### 14.4.10 Packet_valid

This bit is used for isochronous endpoints. It indicates whether the last packet transferred to the memory is received with errors or not. This bit is set if the packet is valid, i.e., it was received without errors. See Section 13–14.6 “Isochronous endpoint operation” on page 204 for isochronous endpoint operation.

This bit is unnecessary for non-isochronous endpoints because a DMA request is generated only for packets without errors, and thus Packet_valid will always be set when the request is generated.

### 14.4.11 LS_byte_extracted

Used in ATLE mode. When set, this bit indicates that the Least Significant Byte (LSB) of the transfer length has been extracted. The extracted size is reflected in the DMA_buffer_length field, bits 23:16.

### 14.4.12 MS_byte_extracted

Used in ATLE mode. When set, this bit indicates that the Most Significant Byte (MSB) of the transfer size has been extracted. The size extracted is reflected in the DMA_buffer_length field, bits 31:24. Extraction stops when LS_Byte_extracted and MS_byte_extracted bits are set.

### 14.4.13 Present_DMA_count

The number of bytes transferred by the DMA engine. The DMA engine updates this field after completing each packet transfer.

For isochronous endpoints, Present_DMA_count is the number of packets transferred; for non-isochronous endpoints, Present_DMA_count is the number of bytes.

### 14.4.14 Message_length_position

Used in ATLE mode. This field gives the offset of the message length position embedded in the incoming data packets. This is applicable only for OUT endpoints. Offset 0 indicates that the message length starts from the first byte of the first packet.

### 14.4.15 Isochronous_packetsize_memory_address

The memory buffer address where the packet size information along with the frame number has to be transferred or fetched. See Figure 13–45. This is applicable to isochronous endpoints only.

## 14.5 Non-isochronous endpoint operation

### 14.5.1 Setting up DMA transfers

Software prepares the DMA Descriptors (DDs) for those physical endpoints to be enabled for DMA transfer. These DDs are present in the system RAM. The start address of the first DD is programmed into the DMA Description pointer (DDP) location for the corresponding endpoint in the UDCA. Software then sets the EPxx_DMA_ENABLE bit for this endpoint in the USBEpDMAEn register (Section 13–9.7.6).The DMA_mode bit field in the descriptor is set to ‘00’ for normal mode operation. All other DD fields are initialized as specified in Table 13–179.

DMA operation is not supported for physical endpoints 0 and 1 (default control endpoints).

### 14.5.2 Finding DMA Descriptor

When there is a trigger for a DMA transfer for an endpoint, the DMA engine will first determine whether a new descriptor has to the fetched or not. A new descriptor does not have to be fetched if the last packet transferred was for the same endpoint and the DD is not yet in the retired state. An internal flag called DMA_PROCEED is used to identify this condition (see Section 13–14.5.4 “Optimizing descriptor fetch” on page 203).

If a new descriptor has to be read, the DMA engine will calculate the location of the DDP for this endpoint and will fetch the start address of the DD from this location. A DD start address at location zero is considered invalid. In this case the NDDR interrupt is raised. All other word-aligned addresses are considered valid.

When the DD is fetched, the DD status word (word 3) is read first and the status of the DD_retired bit is checked. If not set, DDP points to a valid DD. If DD_retired is set, the DMA engine will read the control word (word 1) of the DD.

If Next_DD_valid bit is set, the DMA engine will fetch the Next_DD_pointer field (word 0) of the DD and load it to the DDP. The new DDP is written to the UDCA area.

The full DD (4 words) will then be fetched from the address in the DDP. The DD will give the details of the DMA transfer to be done. The DMA engine will load its hardware resources with the information fetched from the DD (start address, DMA count etc.).

If Next_DD_valid is not set and DD_retired bit is set, the DMA engine raises the NDDR interrupt for this endpoint and clears the corresponding EPxx_DMA_ENABLE bit.

### 14.5.3 Transferring the data

For OUT endpoints, the current packet is read from the EP_RAM by the DMA Engine and transferred to the on-chip RAM memory locations starting from DMA_buffer_start_addr. For IN endpoints, the data is fetched from RAM at DMA_buffer_start_addr and written to the EP_RAM. The DMA_buffer_start_addr and Present_DMA_count fields are updated after each packet is transferred.

### 14.5.4 Optimizing descriptor fetch

A DMA transfer normally involves multiple packet transfers. Hardware will not re-fetch a new DD from memory unless the endpoint changes. To indicate an ongoing multi-packet transfer, hardware sets an an internal flag called DMA_PROCEED.

The DMA_PROCEED flag is cleared after the required number of bytes specified in the DMA_buffer_length field is transferred. It is also cleared when the software writes into the USBEpDMADis register. The ability to clear the DMA_PROCEED flag allows software to to force the DD to be re-fetched for the next packet transfer. Writing all zeros into the USBEpDMADis register clears the DMA_PROCEED flag without disabling DMA operation for any endpoint.

### 14.5.5 Ending the packet transfer

On completing a packet transfer, the DMA engine writes back the DD with updated status information to the same memory location from where it was read. The DMA_buffer_start_addr, Present_DMA_count, and the DD_status fields in the DD are updated.

A DD can have the following types of completion:

**Normal completion** - If the current packet is fully transferred and the Present_DMA_count field equals the DMA_buffer_length, the DD has completed normally. The DD will be written back to memory with DD_retired set and DD_status set to NormalCompletion. The EOT interrupt is raised for this endpoint.

**USB transfer end completion** - If the current packet is fully transferred and its size is less than the Max_packet_size field, and the end of the DMA buffer is still not reached, the USB transfer end completion occurs. The DD will be written back to the memory with DD_retired set and DD_Status set to the DataUnderrun completion code. The EOT interrupt is raised for this endpoint.

**Error completion** - If the current packet is partially transferred i.e. the end of the DMA buffer is reached in the middle of the packet transfer, an error situation occurs. The DD is written back with DD_retired set and DD_status set to the DataOverrun status code. The EOT interrupt is raised for this endpoint and the corresponding bit in USBEpDMASt register is cleared. The packet will be re-sent from the endpoint buffer to memory when the corresponding EPxx_DMA_ENABLE bit is set again using the USBEpDMAEn register.

### 14.5.6 No_Packet DD

For an IN transfer, if the system does not have any data to send for a while, it can respond to an NDDR interrupt by programming a No_Packet DD. This is done by setting both the Max_packet_size and DMA_buffer_length fields in the DD to 0. On processing a No_Packet DD, the DMA engine clears the DMA request bit in USBDMARSt corresponding to the endpoint without transferring a packet. The DD is retired with a status code of NormalCompletion. This can be repeated as often as necessary. The device will respond to IN token packets on the USB bus with a NAK until a DD with a data packet is programmed and the DMA transfers the packet into the endpoint buffer.

## 14.6 Isochronous endpoint operation

For isochronous endpoints, the packet size can vary for each packet. There is one packet per isochronous endpoint for each frame.

### 14.6.1 Setting up DMA transfers

Software sets the isochronous endpoint bit to 1 in the DD, and programs the initial value of the Isochronous_packetsize_memory_address field. All other fields are initialized the same as for non-isochronous endpoints.

For isochronous endpoints, the DMA_buffer_length and Present_DMA_count fields are in frames rather than bytes.

### 14.6.2 Finding the DMA Descriptor

Finding the descriptors is done in the same way as that for a non-isochronous endpoint.

A DMA request will be placed for DMA-enabled isochronous endpoints on every FRAME interrupt. On processing the request, the DMA engine will fetch the descriptor and if Isochronous_endpoint is set, will fetch the Isochronous_packetsize_memory_address from the fifth word of the DD.

### 14.6.3 Transferring the Data

The data is transferred to or from the memory location DMA_buffer_start_addr. After the end of the packet transfer the Present_DMA_count value is incremented by 1.

The isochronous packet size is stored in memory as shown in Figure 13–45. Each word in the packet size memory shown is divided into fields: Frame_number (bits 31 to 17), Packet_valid (bit 16), and Packet_length (bits 15 to 0). The space allocated for the packet size memory for a given DD should be DMA_buffer_length words in size – one word for each packet to transfer.

#### OUT endpoints

At the completion of each frame, the packet size is written to the address location in Isochronous_packet_size_memory_address, and Isochronous_packet_size_memory_address is incremented by 4.

#### IN endpoints

Only the Packet_length field of the isochronous packet size word is used. For each frame, an isochronous data packet of size specified by this field is transferred from the USB device to the host, and Isochronous_packet_size_memory_address is incremented by 4 at the end of the packet transfer. If Packet_length is zero, an empty packet will be sent by the USB device.

### 14.6.4 DMA descriptor completion

DDs for isochronous endpoints can only end with a status code of NormalCompletion since there is no short packet on Isochronous endpoints, and the USB transfer continues indefinitely until a SystemError occurs. There is no DataOverrun detection for isochronous endpoints.

### 14.6.5 Isochronous OUT Endpoint Operation Example

Assume that an isochronous endpoint is programmed for the transfer of 10 frames and that the transfer begins when the frame number is 21. After transferring four frames with packet sizes of 10,15, 8 and 20 bytes without errors, the descriptor and memory map appear as shown in Figure 13–45.

The_total_number_of_bytes_transferred = 0x0A + 0x0F + 0x08 + 0x14 = 0x35.

The Packet_valid bit (bit 16) of all the words in the packet length memory is set to 1.

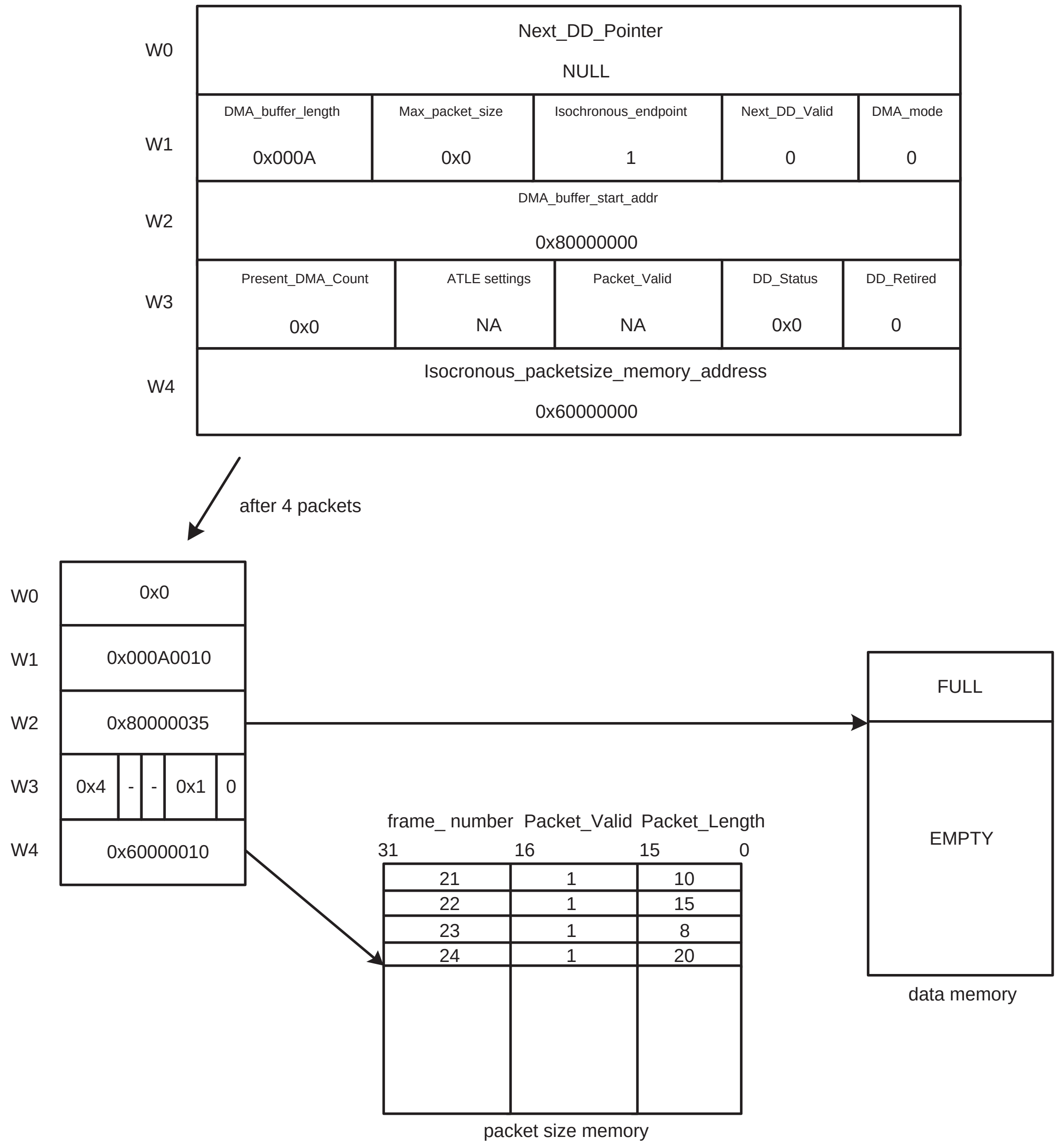


**Fig 45. Isochronous OUT endpoint operation example**

## 14.7 Auto Length Transfer Extraction (ATLE) mode operation

Some host drivers such as NDIS (Network Driver Interface Specification) host drivers are capable of concatenating small USB transfers (delta transfers) to form a single large USB transfer. For OUT USB transfers, the device hardware has to break up this concatenated transfer back into the original delta transfers and transfer them to separate DMA buffers. This is achieved by setting the DMA mode to Auto Transfer Length Extraction (ATLE) mode in the DMA descriptor. ATLE mode is supported for Bulk endpoints only.

### OUT transfers in ATLE mode

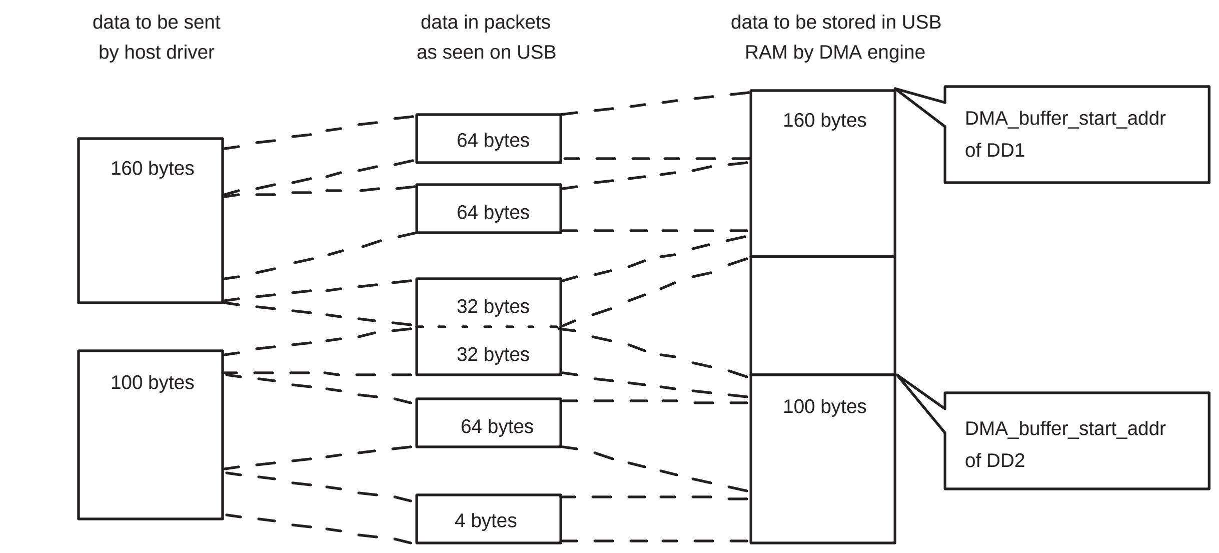


**Fig 46. Data transfer in ATLE mode**

Figure 13–46 shows a typical OUT USB transfer in ATLE mode, where the host concatenates two USB transfers of 160 bytes and 100 bytes, respectively. Given a MaxPacketSize of 64, the device hardware interprets this USB transfer as four packets of 64 bytes and a short packet of 4 bytes. The third and fourth packets are concatenated. Note that in Normal mode, the USB transfer would be interpreted as packets of 64, 64, 32, and 64 and 36 bytes.

It is now the responsibility of the DMA engine to separate these two USB transfers and put them in the memory locations in the DMA_buffer_start_addr field of DMA Descriptor 1 (DD1) and DMA Descriptor 2 (DD2).

Hardware reads the two-byte-wide DMA_buffer_length at the offset (from the start of the USB transfer) specified by Message_length_position from the incoming data packets and writes it in the DMA_buffer_length field of the DD. To ensure that both bytes of the DMA_buffer_length are extracted in the event they are split between two packets, the flags LS_byte_extracted and MS_byte_extracted are set by hardware after the respective byte is extracted. After the extraction of the MS byte, the DMA transfer continues as in the normal mode.

The flags LS_byte_extracted and MS_byte_extracted are set to 0 by software when preparing a new DD. Therefore, once a DD is retired, the transfer length is extracted again for the next DD.

If DD1 is retired during the transfer of a concatenated packet (such as the third packet in Figure 13–46), and DD2 is not programmed (Next_DD_valid field of DD1 is 0), then DD1 is retired with DD_status set to the DataOverrun status code. This is treated as an error condition and the corresponding EPxx_DMA_ENABLE bit of USBEpDMASt is cleared by hardware.

In ATLE mode, the last buffer length to be transferred always ends with a short or empty packet indicating the end of the USB transfer. If the concatenated transfer lengths are such that the USB transfer ends on a MaxPacketSize packet boundary, the (NDIS) host will send an empty packet to mark the end of the USB transfer.

#### IN transfers in ATLE mode

For IN USB transfers from the device to the host, DMA_buffer_length is set by the device software as in normal mode.

In ATLE mode, the device concatenates data from multiple DDs to form a single USB transfer. If a DD is retired in the middle of a packet (packet size is less than MaxPacketSize), the next DD referenced by Next_DD_pointer is fetched, and the remaining bytes to form a packet of MaxPacketSize are transferred from the next DD's buffer.

If the next DD is not programmed (i.e. Next_DD_valid field in DD is 0), and the DMA buffer length for the current DD has completed before the MaxPacketSize packet boundary, then the available bytes from current DD are sent as a short packet on USB, which marks the end of the USB transfer for the host.

If the last buffer length completes on a MaxPacketSize packet boundary, the device software must program the next DD with DMA_buffer_length field 0, so that an empty packet is sent by the device to mark the end of the USB transfer for the host.

### 14.7.1 Setting up the DMA transfer

For OUT endpoints, the host hardware needs to set the field Message_length_position in the DD. This indicates the start location of the message length in the incoming data packets. Also the device software has to set the DMA_buffer_length field to 0 for OUT endpoints because this field is updated by the device hardware after the extraction of the buffer length.

For IN endpoints, descriptors are set in the same way as in normal mode operation.

Since a single packet can be split between two DDs, software should always keep two DDs ready, except for the last DMA transfer which ends with a short or empty packet.

### 14.7.2 Finding the DMA Descriptor

DMA descriptors are found in the same way as the normal mode operation.

### 14.7.3 Transferring the Data

#### OUT endpoints

If the LS_byte_extracted or MS_byte_extracted bit in the status field is not set, the hardware will extract the transfer length from the data stream and program DMA_buffer_length. Once the extraction is complete both the LS_byte_extracted and MS_byte_extracted bits will be set.

#### IN endpoints

The DMA transfer proceeds as in normal mode and continues until the number of bytes transferred equals the DMA_buffer_length.

### 14.7.4 Ending the packet transfer

The DMA engine proceeds with the transfer until the number of bytes specified in the field DMA_buffer_length is transferred to or from on-chip RAM. Then the EOT interrupt will be generated. If this happens in the middle of the packet, the linked DD will get loaded and the remaining part of the packet gets transferred to or from the address pointed by the new DD.

#### OUT endpoints

If the linked DD is not valid and the packet is partially transferred to memory, the DD ends with DataOverrun status code set, and the DMA will be disabled for this endpoint. Otherwise DD_status will be updated with the NormalCompletion status code.

#### IN endpoints

If the linked DD is not valid and the packet is partially transferred to USB, the DD ends with a status code of NormalCompletion in the DD_status field. This situation corresponds to the end of the USB transfer, and the packet will be sent as a short packet. Also, when the linked DD is valid and buffer length is 0, an empty packet will be sent to indicate the end of the USB transfer.

## 15. Double buffered endpoint operation

The Bulk and Isochronous endpoints of the USB Device Controller are double buffered to increase data throughput.

When a double-buffered endpoint is realized, enough space for both endpoint buffers is automatically allocated in the EP_RAM. See Section 13–9.4.1.

For the following discussion, the endpoint buffer currently accessible to the CPU or DMA engine for reading or writing is said to be the active buffer.

### 15.1 Bulk endpoints

For Bulk endpoints, the active endpoint buffer is switched by the SIE Clear Buffer or Validate Buffer commands.

The following example illustrates how double buffering works for a Bulk OUT endpoint in Slave mode:

Assume that both buffer 1 (B_1) and buffer 2 (B_2) are empty, and that the active buffer is B_1.

1. The host sends a data packet to the endpoint. The device hardware puts the packet into B_1, and generates an endpoint interrupt.
2. Software clears the endpoint interrupt and begins reading the packet data from B_1. While B_1 is still being read, the host sends a second packet, which device hardware places in B_2, and generates an endpoint interrupt.
3. Software is still reading from B_1 when the host attempts to send a third packet. Since both B_1 and B_2 are full, the device hardware responds with a NAK.
4. Software finishes reading the first packet from B_1 and sends a SIE Clear Buffer command to free B_1 to receive another packet. B_2 becomes the active buffer.

5. Software sends the SIE Select Endpoint command to read the Select Endpoint Register and test the FE bit. Software finds that the active buffer (B_2) has data (FE=1). Software clears the endpoint interrupt and begins reading the contents of B_2.
6. The host resends the third packet which device hardware places in B_1. An endpoint interrupt is generated.
7. Software finishes reading the second packet from B_2 and sends a SIE Clear Buffer command to free B_2 to receive another packet. B_1 becomes the active buffer. Software waits for the next endpoint interrupt to occur (it already has been generated back in step 6).
8. Software responds to the endpoint interrupt by clearing it and begins reading the third packet from B_1.
9. Software finishes reading the third packet from B_1 and sends a SIE Clear Buffer command to free B_1 to receive another packet. B_2 becomes the active buffer.
10. Software tests the FE bit and finds that the active buffer (B_2) is empty (FE=0).
11. Both B_1 and B_2 are empty. Software waits for the next endpoint interrupt to occur. The active buffer is now B_2. The next data packet sent by the host will be placed in B_2.

The following example illustrates how double buffering works for a Bulk IN endpoint in Slave mode:

Assume that both buffer 1 (B_1) and buffer 2 (B_2) are empty and that the active buffer is B_1. The interrupt on NAK feature is enabled.

1. The host requests a data packet by sending an IN token packet. The device responds with a NAK and generates an endpoint interrupt.
2. Software clears the endpoint interrupt. The device has three packets to send. Software fills B_1 with the first packet and sends a SIE Validate Buffer command. The active buffer is switched to B_2.
3. Software sends the SIE Select Endpoint command to read the Select Endpoint Register and test the FE bit. It finds that B_2 is empty (FE=0) and fills B_2 with the second packet. Software sends a SIE Validate Buffer command, and the active buffer is switched to B_1.
4. Software waits for the endpoint interrupt to occur.
5. The device successfully sends the packet in B_1 and clears the buffer. An endpoint interrupt occurs.
6. Software clears the endpoint interrupt. Software fills B_1 with the third packet and validates it using the SIE Validate Buffer command. The active buffer is switched to B_2.
7. The device successfully sends the second packet from B_2 and generates an endpoint interrupt.
8. Software has no more packets to send, so it simply clears the interrupt.
9. The device successfully sends the third packet from B_1 and generates an endpoint interrupt.
10. Software has no more packets to send, so it simply clears the interrupt.

11. Both B_1 and B_2 are empty, and the active buffer is B_2. The next packet written by software will go into B_2.

In DMA mode, switching of the active buffer is handled automatically in hardware. For Bulk IN endpoints, proactively filling an endpoint buffer to take advantage of the double buffering can be accomplished by manually starting a packet transfer using the USBDMARSet register.

## 15.2 Isochronous endpoints

For isochronous endpoints, the active data buffer is switched by hardware when the FRAME interrupt occurs. The SIE Clear Buffer and Validate Buffer commands do not cause the active buffer to be switched.

Double buffering allows the software to make full use of the frame interval writing or reading a packet to or from the active buffer, while the packet in the other buffer is being sent or received on the bus.

For an OUT isochronous endpoint, any data not read from the active buffer before the end of the frame is lost when it switches.

For an IN isochronous endpoint, if the active buffer is not validated before the end of the frame, an empty packet is sent on the bus when the active buffer is switched, and its contents will be overwritten when it becomes active again.

# UM10316

## Chapter 14: LPC29xx USB Host controller

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The USB host controller is available on LPC2930 and LPC2939 only (see also Table 1–4).

## 2. Introduction

This section describes the host portion of the USB 2.0 OTG dual role core which integrates the host controller (OHCI compliant), device controller and I2C. The I2C interface controls the external OTG ATX.

The USB is a 4 wire bus that supports communication between a host and a number (127 max.) of peripherals. The host controller allocates the USB bandwidth to attached devices through a token based protocol. The bus supports hot plugging, un-plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The host controller enables data exchange with various USB devices attached to the bus. It consists of register interface, serial interface engine and DMA controller. The register interface complies to the OHCI specification.

**Table 180. USB (OHCI) related acronyms and abbreviations used in this chapter**

| Acronym/abbreviation | Description |
|---|---|
| AHB | Advanced High-Performance Bus |
| ATX | Analog Transceiver |
| DMA | Direct Memory Access |
| FS | Full Speed |
| LS | Low Speed |
| OHCI | Open Host Controller Interface |
| USB | Universal Serial Bus |

### 2.1 Features

- OHCI compliant.
- OpenHCI specifies the operation and interface of the USB Host Controller and SW Driver
  - USBOperational: Process Lists and generate SOF Tokens.
  - USBReset: Forces reset signaling on the bus, SOF disabled.
  - USBSuspend: Monitor USB for wakeup activity.
  - USBResume: Forces resume signaling on the bus.
- The Host Controller has four USB states visible to the SW Driver.
- HCCA register points to Interrupt and Isochronous Descriptors List.
- ControlHeadED and BulkHeadED registers point to Control and Bulk Descriptors List.

### 2.2 Architecture

The architecture of the USB host controller is shown below in Figure 14–47.

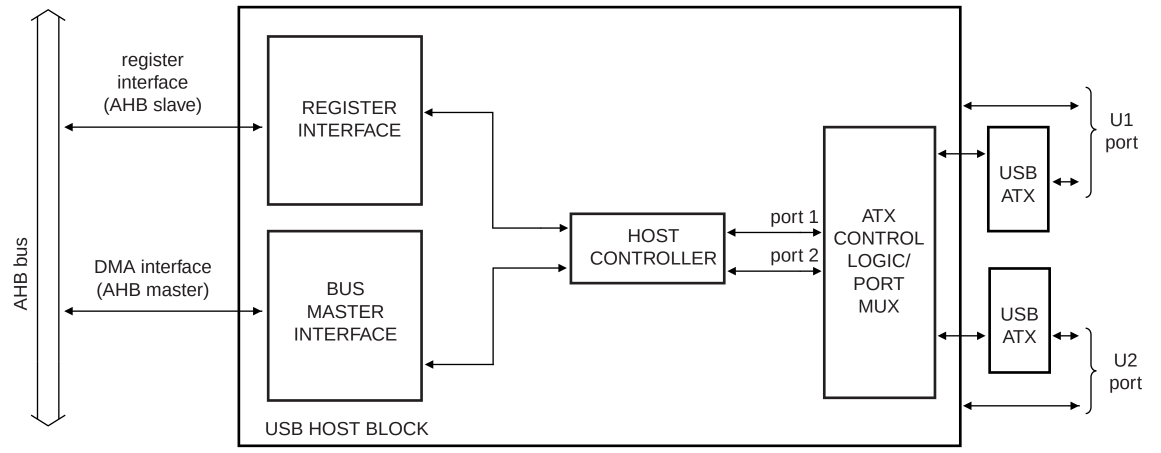


**Fig 47. USB Host controller block diagram**

## 3. Interfaces

The OTG controller has two USB ports indicated by suffixes 1 and 2 in the USB pin names and referred to as USB port 1 (U1) and USB port 2 (U2) in the following text.

### 3.1 Pin description

**Table 181. USB OTG port pins**

| Pin name | Direction | Description | Pin category |
|---|---|---|---|
| USB_VBUS | I | $V_{BUS}$ status input. When this function is not enabled via its corresponding PINSEL register, it is driven HIGH internally. | USB Connector |
| **Port U1** | | | |
| USB_D+1 | I/O | Positive differential data | USB Connector |
| USB_D–1 | I/O | Negative differential data | USB Connector |
| USB_CONNECT1 | O | SoftConnect control signal | Control |
| USB_UP_LED1 | O | GoodLink LED control signal | Control |
| $\overline{\mathrm{USB\_PPWR1}}$ | O | Port power enable | Host power switch |
| USB_PWRD1 | I | Port power status | Host power switch |
| $\overline{\mathrm{USB\_OVRCR1}}$ | I | Over-current status | Host power switch |
| $\overline{\mathrm{USB\_HSTEN1}}$ | O | Host enabled status | |
| **Port U2** | | | |
| USB_D+2 | I/O | Positive differential data | USB Connector |
| USB_D–2 | I/O | Negative differential data | USB Connector |
| USB_CONNECT2 | O | SoftConnect control signal | Control |
| USB_UP_LED2 | O | GoodLink LED control signal | Control |
| $\overline{\mathrm{USB\_PPWR2}}$ | O | Port power enable | Host power switch |

Table 181. USB OTG port pins

| Pin name | Direction | Description | Pin category |
|---|---|---|---|
| U2PWRD2 | I | Port power status | Host power switch |
| $\overline{\text{USB\_OVRCR2}}$ | I | Over-current status | Host power switch |
| $\overline{\text{USB\_HSTEN2}}$ | O | Host enabled status | Control |

### 3.1.1 USB host usage note

Both ports can be configured as USB hosts. For details on how to connect the USB ports, see the USB OTG chapter, Section 15–5.

## 3.2 Software interface

The software interface of the USB host block consists of a register view and the format definitions for the endpoint descriptors. For details on these two aspects see the OHCI specification. The register map is shown in the next section.

## 4. Register overview

The following registers are located in the AHB clock ‘cclk’ domain. They can be accessed directly by the processor. All registers are 32 bit wide and aligned in the word address boundaries.

Table 182. Register overview: USB Host (base address 0xE010 0000)

| Name | R/W[1] | Address offset | Description | Reset value |
|---|---|---|---|---|
| HcRevision | R | 0x00 | BCD representation of the version of the HCI specification that is implemented by the Host Controller. | 0x10 |
| HcControl | R/W | 0x04 | Defines the operating modes of the HC. | 0x0 |
| HcCommandStatus | R/W | 0x08 | This register is used to receive the commands from the Host Controller Driver (HCD). It also indicates the status of the HC. | 0x0 |
| HcInterruptStatus | R/W | 0x0C | Indicates the status on various events that cause hardware interrupts by setting the appropriate bits. | 0x0 |
| HcInterruptEnable | R/W | 0x10 | Controls the bits in the HcInterruptStatus register and indicates which events will generate a hardware interrupt. | 0x0 |
| HcInterruptDisable | R/W | 0x14 | The bits in this register are used to disable corresponding bits in the HCInterruptStatus register and in turn disable that event leading to hardware interrupt. | 0x0 |
| HcHCCA | R/W | 0x18 | Contains the physical address of the host controller communication area. | 0x0 |
| HcPeriodCurrentED | R | 0x1C | Contains the physical address of the current isochronous or interrupt endpoint descriptor. | 0x0 |
| HcControlHeadED | R/W | 0x20 | Contains the physical address of the first endpoint descriptor of the control list. | 0x0 |
| HcControlCurrentED | R/W | 0x24 | Contains the physical address of the current endpoint descriptor of the control list | 0x0 |
| HcBulkHeadED | R/W | 0x28 | Contains the physical address of the first endpoint descriptor of the bulk list. | 0x0 |

Table 182. Register overview: USB Host (base address 0xE010 0000) *…continued*

| Name | R/W[1] | Address offset | Description | Reset value |
|---|---|---|---|---|
| HcBulkCurrentED | R/W | 0x2C | Contains the physical address of the current endpoint descriptor of the bulk list. | 0x0 |
| HcDoneHead | R | 0x30 | Contains the physical address of the last transfer descriptor added to the ‘Done’ queue. | 0x0 |
| HcFmInterval | R/W | 0x34 | Defines the bit time interval in a frame and the full speed maximum packet size which would not cause an overrun. | 0x2EDF |
| HcFmRemaining | R | 0x38 | A 14-bit counter showing the bit time remaining in the current frame. | 0x0 |
| HcFmNumber | R | 0x3C | Contains a 16-bit counter and provides the timing reference among events happening in the HC and the HCD. | 0x0 |
| HcPeriodicStart | R/W | 0x40 | Contains a programmable 14-bit value which determines the earliest time HC should start processing a periodic list. | 0x0 |
| HcLSThreshold | R/W | 0x44 | Contains 11-bit value which is used by the HC to determine whether to commit to transfer a maximum of 8-byte LS packet before EOF. | 0x628h |
| HcRhDescriptorA | R/W | 0x48 | First of the two registers which describes the characteristics of the root hub. | 0xFF000902 |
| HcRhDescriptorB | R/W | 0x4C | Second of the two registers which describes the characteristics of the Root Hub. | 0x60000h |
| HcRhStatus | R/W | 0x50 | This register is divided into two parts. The lower D-word represents the hub status field and the upper word represents the hub status change field. | 0x0 |
| HcRhPortStatus[1] | R/W | 0x54 | Controls and reports the port events on a per-port basis. | 0x0 |
| HcRhPortStatus[2] | R/W | 0x58 | Controls and reports the port events on a per port basis. | 0x0 |
| Module_ID/Ver_Rev_ID | R | 0xFC | IP number, where yy (0x00) is unique version number and zz (0x00) is a unique revision number. | 0x3505yyzz |

[1] The R/W column lists the accessibility of the register:

a) Registers marked ‘R’ for access will return their current value when read.

b) Registers marked ‘R/W’ allow both read and write.

## 4.1 USB Host Register Definitions

Refer to the OHCI specification document for register definitions.

# UM10316

# Chapter 15: LPC29xx USB OTG interface

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The USB OTG controller is available in **LPC2926/27/29, LPC2930, and LPC2939** only. Note that the host controller is **not** implemented on the **LPC2926/27** and **LPC2929** (see also Table 1–4). Depending on the LPC29xx part, different USB port configurations are available.

Table 183. LPC29xx USB port configurations

| Part | Port 1 device | Port 1 host | Port 1 OTG | Port 2 host | Port 2 device |
|---|---|---|---|---|---|
| LPC2921/23/25 | yes | no | no | - | - |
| LPC2917/19/01 | - | - | - | - | - |
| LPC2926/27/29 | yes | no | yes | - | - |
| LPC2930 | no | yes | yes | yes | yes |
| LPC2939 | no | yes | yes | yes | yes |

## 2. Introduction

This chapter describes the OTG and I$^2$C portions of the USB 2.0 OTG dual role device controller which integrates the (OHCI) host controller, device controller, and I$^2$C. The I$^2$C interface (Master only) controls an external OTG transceiver.

USB OTG (On-The-Go) is a supplement to the USB 2.0 specification that augments the capability of existing mobile devices and USB peripherals by adding host functionality for connection to USB peripherals. The specification and more information on USB OTG can be found on the USB Implementers Forum web site.

## 3. Features

- Fully compliant with On-The-Go supplement to the *USB 2.0 Specification, Revision 1.0a*.
- Hardware support for Host Negotiation Protocol (HNP).
- Includes a programmable timer required for HNP and SRP.
- Supports any OTG transceiver compliant with the *OTG Transceiver Specification (CEA-2011), Rev. 1.0*.

## 4. Architecture

The architecture of the USB OTG controller is shown below in the block diagram.

The host, device, OTG, and I2C controllers can be programmed through the register interface. The OTG controller enables dynamic switching between host and device roles through the HNP protocol. One port may be connected to an external OTG transceiver to support an OTG connection. The communication between the register interface and an external OTG transceiver is handled through an I$^2$C interface and through the external OTG transceiver interrupt signal.

For USB connections that use the device or host controller only (not OTG), the ports use an embedded USB Analog Transceiver (ATX).

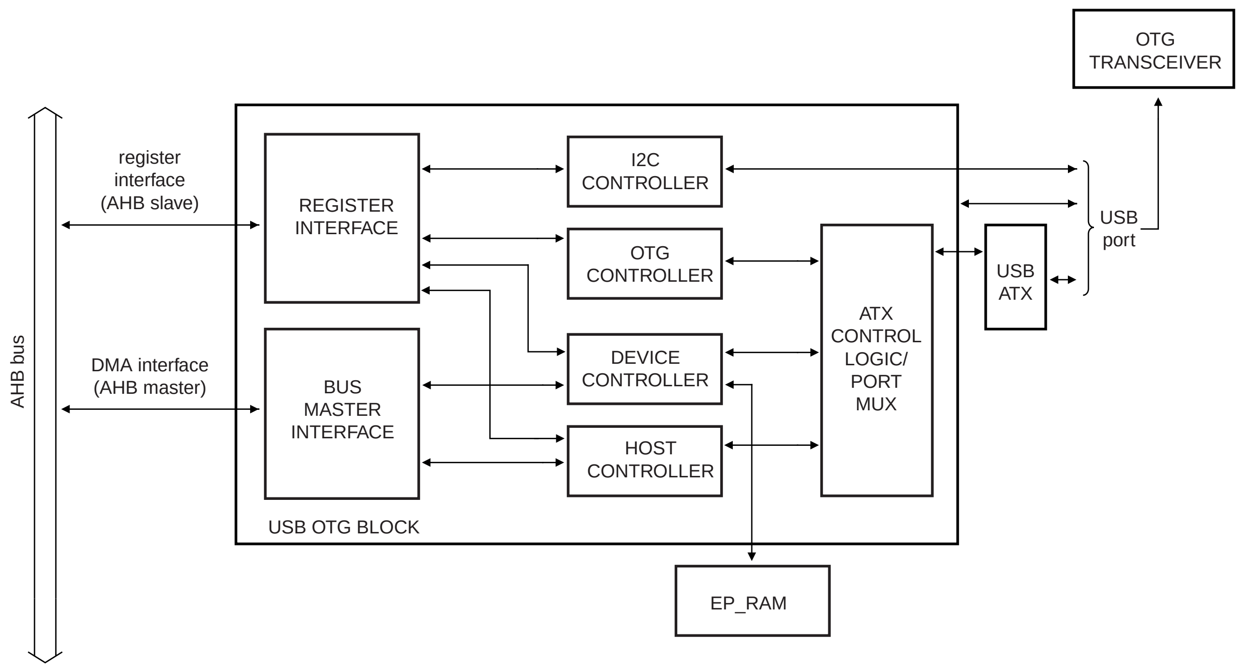


**Fig 48. USB OTG controller block diagram**

## 5. Pin configuration

See Table 15–183 for port configurations for the different LPC29xx parts.

**Table 184. USB OTG port pins**

| Pin name | Direction | Description | Interfacing |
|---|---|---|---|
| **Port 1** | | | |
| USB_VBUS1 | I | $V_{BUS}$ status input. When this function is not enabled via its corresponding SFSP register, it is driven HIGH internally (see Table 6–59). | - |
| USB_D+1 | I/O | Positive differential data | - |
| USB_D–1 | I/O | Negative differential data | - |
| $\overline{\text{USB\_CONNECT1}}$ | O | SoftConnect control signal | - |
| $\overline{\text{USB\_UP\_LED1}}$ | O | GoodLink LED control signal | - |
| USB_SCL1 | I/O | I$^2$C serial clock | External OTG transceiver |
| USB_SDA1 | I/O | I$^2$C serial data | External OTG transceiver |
| USB_LS1 | O | Low speed status (applies to host functionality only) | External OTG transceiver |
| $\overline{\text{USB\_RST1}}$ | O | USB reset status | External OTG transceiver |
| $\overline{\text{USB\_INT1}}$ | O | USB transceiver interrupt | External OTG transceiver |

**Table 184. USB OTG port pins**

| Pin name | Direction | Description | Interfacing |
|---|---|---|---|
| USB_SSPND1 | O | Bus suspend status | External OTG transceiver |
| USB_PWRD1 | I | Port power status | USB host |
| USB_PPWR1 | O | Port power enable | USB host |
| USB_OVRCR1 | I | Over-current status | USB host |
| **Port 2** | | | |
| USB_VBUS2 | I | $V_{BUS}$ status input. When this function is not enabled via its corresponding SFSP register, it is driven HIGH internally (see Table 6–59). | - |
| USB_D+2 | I/O | Positive differential data | - |
| USB_D–2 | I/O | Negative differential data | - |
| USB_CONNECT2 | O | SoftConnect control signal | - |
| USB_UP_LED2 | O | GoodLink LED control signal | - |
| USB_PWRD2 | I | Port power status | USB host |
| USB_PPWR2 | O | Port power enable | USB host |
| USB_OVRCR2 | I | Over-current status | USB host |

The following figures show different ways to realize connections to an USB device. The example described here uses an ISP1302 (NXP) for the external OTG transceiver and the USB Host power switch LM3526-L (National Semiconductors).

**Remark:** To select the USB functions, see the SPSP registers in Table 6–59, Table 6–60, and Table 6–61. For input only pins (e.g. USB_OVRCR), select PAD_TYPE in the corresponding SFSP register to digital input without pullup or pulldown enabled.

## 5.1 Suggested USB interface solutions

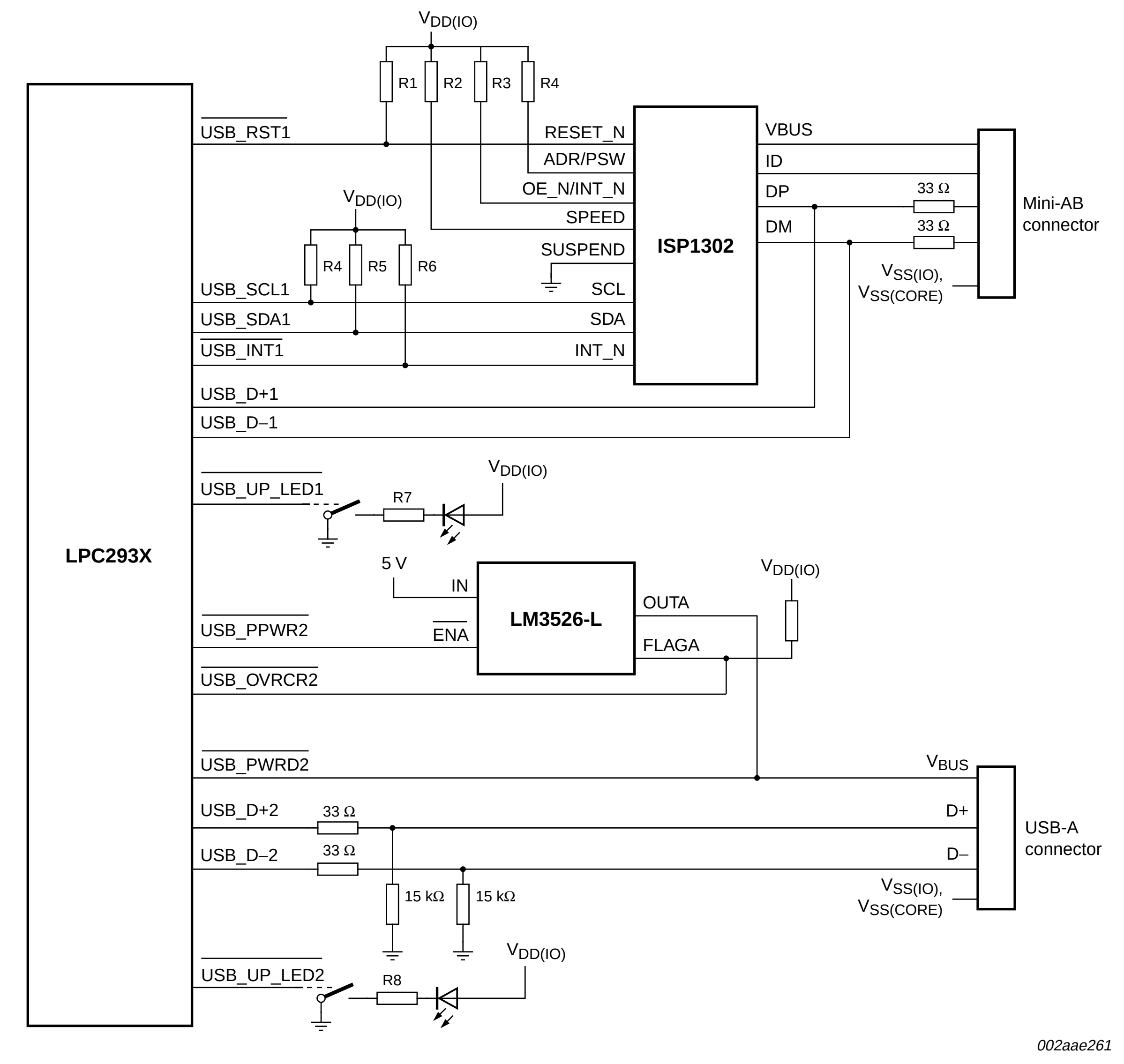


Fig 49. LPC29xx USB OTG port configuration: USB port 1 OTG dual-role device, USB port 2 host

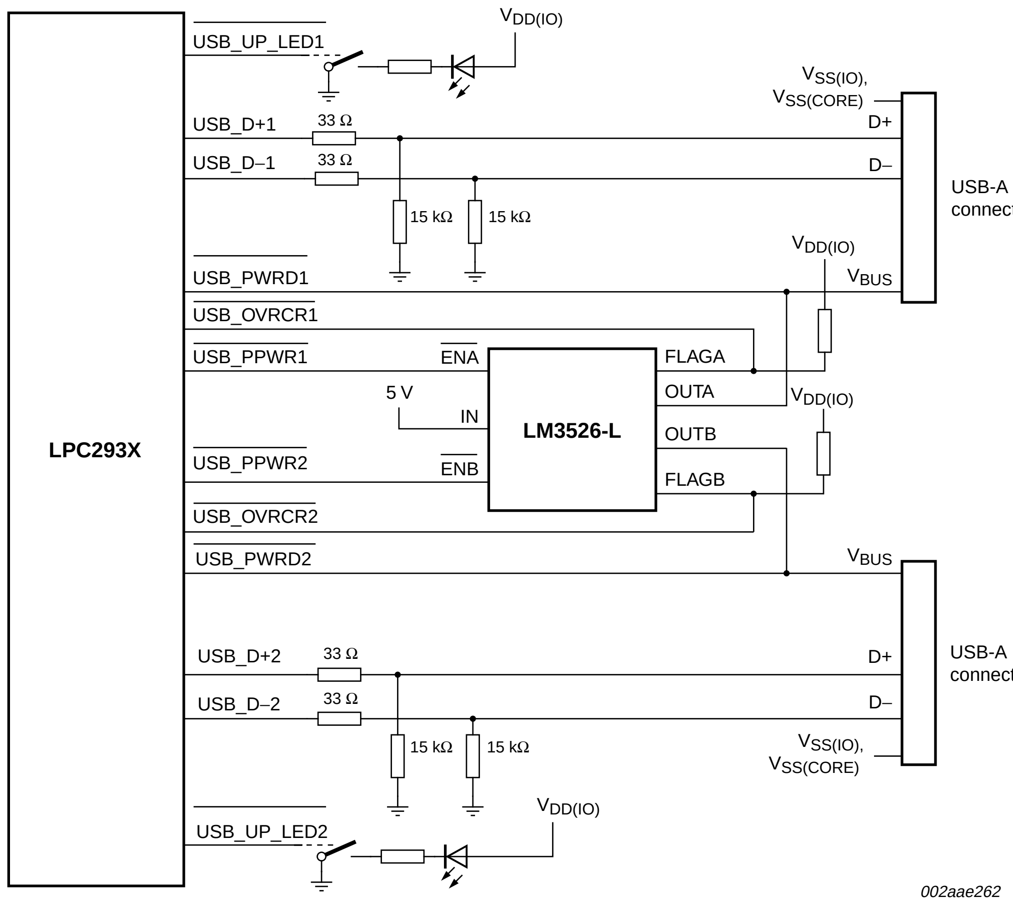


**Fig 50. LPC29xx USB OTG port configuration: USB port 1 host, USB port 2 host**

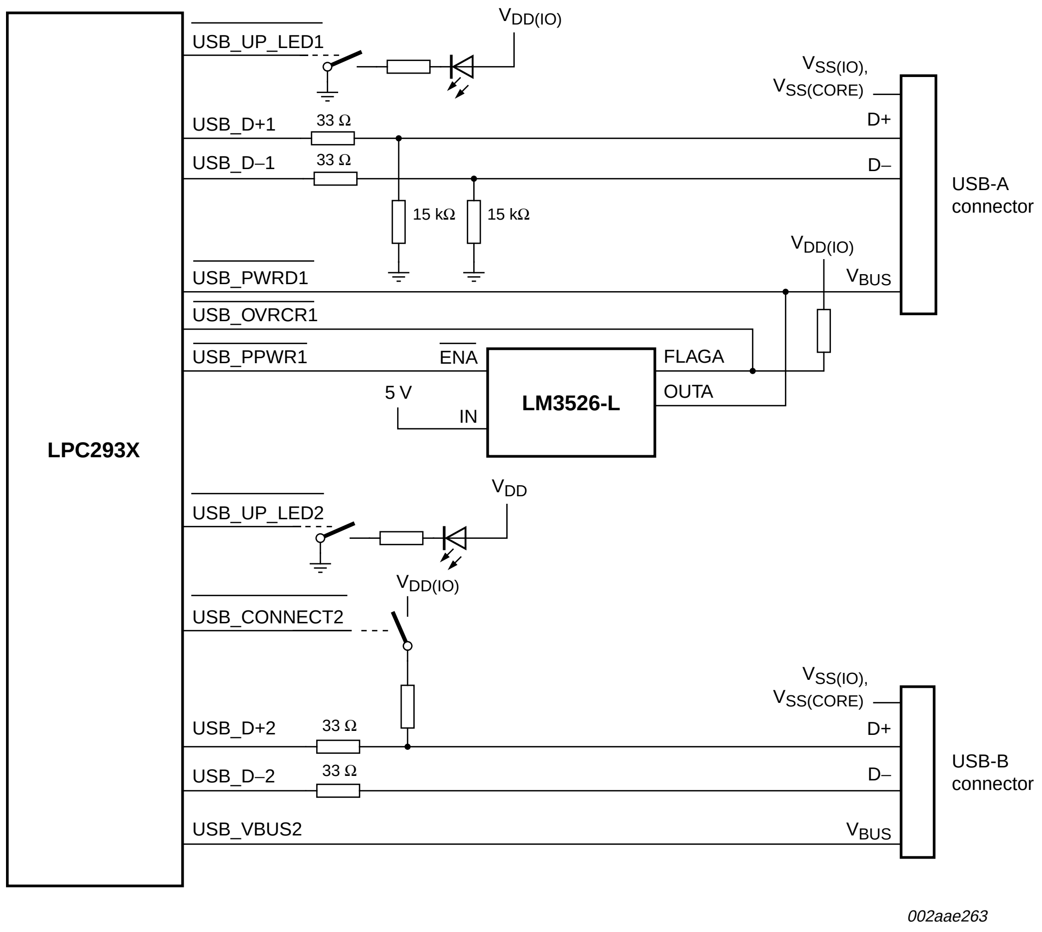


Fig 51. LPC29xx USB OTG port configuration: USB port 2 device, USB port 1 host

## 6. Register overview

The OTG and I$^2$C registers are summarized in the following table.

The Device and Host registers are explained in Table 13–117 and in the USB Device Controller and USB Host (OHCI) Controller chapters. All registers are 32 bits wide and aligned to word address boundaries.

Table 185. Register overview: USB OTG and I$^2$C registers (base address 0xE010 0000)

| Name | Access | Address offset | Description |
|---|---|---|---|
| **OTG registers** | | | |
| OTGIntSt | RO | 0x100 | OTG Interrupt Status |
| OTGIntEn | R/W | 0x104 | OTG Interrupt Enable |
| OTGIntSet | WO | 0x108 | OTG Interrupt Set |
| OTGIntClr | WO | 0x10C | OTG Interrupt Clear |
| OTGStCtrl | R/W | 0x110 | OTG Status and Control |
| OTGTmr | R/W | 0x114 | OTG Timer |

Table 185. Register overview: USB OTG and I$^2$C registers (base address 0xE010 0000)

| Name | Access | Address offset | Description |
|---|---|---|---|
| **I$^2$C registers** | | | |
| I2C_RX | RO | 0x300 | I2C Receive |
| I2C_TX | WO | 0x300 | I2C Transmit |
| I2C_STS | RO | 0x304 | I2C Status |
| I2C_CTL | R/W | 0x308 | I2C Control |
| I2C_CLKHI | R/W | 0x30C | I2C Clock High |
| I2C_CLKLO | WO | 0x310 | I2C Clock Low |
| **Clock control registers** | | | |
| OTGClkCtrl | R/W | 0xFF4 | OTG clock controller |
| OTGClkSt | RO | 0xFF8 | OTG clock status |

## 6.1 OTG Interrupt Status Register (OTGIntSt - 0xE01F 0100)

Bits in this register are set by hardware when the interrupt event occurs during the HNP handoff sequence. See Section 15–7 for more information on when these bits are set.

Table 186. OTG Interrupt Status register (OTGIntSt - address 0xE01F 0100) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:4 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 3 | HNP_SUCCESS | HNP succeeded.<br>This bit is set by hardware to indicate that the HNP switching has succeeded. | 0 |
| 2 | HNP_FAILURE | HNP failed.<br>This bit is set by hardware to indicate that the HNP switching has failed. | 0 |
| 1 | REMOVE_PU | Remove pull-up.<br>This bit is set by hardware to indicate that software needs to disable the D+ pull-up resistor. | 0 |
| 0 | TMR | Timer time-out. | 0 |

## 6.2 OTG Interrupt Enable Register (OTGIntEn - 0xE010 0104)

Writing a one to a bit in this register enables the corresponding bit in OTGIntSt to generate an interrupt on one of the interrupt lines. The interrupt is routed to the USB_OTG_INT interrupt line in the USBIntSt register.

The bit allocation and reset value of OTGIntEn is the same as OTGIntSt.

## 6.3 OTG Interrupt Set Register (OTGIntSet - 0xE010 C20C)

Writing a one to a bit in this register will set the corresponding bit in the OTGIntSt register. Writing a zero has no effect. The bit allocation of OTGIntSet is the same as in OTGIntSt.

## 6.4 OTG Interrupt Clear Register (OTGIntClr - 0xE010 010C)

Writing a one to a bit in this register will clear the corresponding bit in the OTGIntSt register. Writing a zero has no effect. The bit allocation of OTGIntClr is the same as in OTGIntSt.

## 6.5 OTG Status and Control Register (OTGStCtrl - 0xE010 0110)

The OTGStCtrl register allows enabling hardware tracking during the HNP hand over sequence, controlling the OTG timer, monitoring the timer count, and controlling the functions mapped to port U1 and U2.

Time critical events during the switching sequence are controlled by the OTG timer. The timer can operate in two modes:

1. Monoshot mode: an interrupt is generated at the end of TIMEOUT_CNT (see Section 15–6.6 “OTG Timer Register (OTGTmr - 0xE010 0114)”), the TMR bit is set in OTGIntSt, and the timer will be disabled.
2. Free running mode: an interrupt is generated at the end of TIMEOUT_CNT (see Section 15–6.6 “OTG Timer Register (OTGTmr - 0xE010 0114)”), the TMR bit is set, and the timer value is reloaded into the counter. The timer is not disabled in this mode.

**Table 187. OTG Status Control register (OTGStCtrl - address 0xE010 0110) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:16 | TMR_CNT | Current timer count value. | 0x0 |
| 15:11 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 10 | PU_REMOVED | When the B-device changes its role from peripheral to host, software sets this bit when it removes the D+ pull-up, see Section 15–7. Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set. | 0 |
| 9 | A_HNP_TRACK | Enable HNP tracking for A-device (host), see Section 15–7. Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set. | 0 |
| 8 | B_HNP_TRACK | Enable HNP tracking for B-device (peripheral), see Section 15–7. Hardware clears this bit when HNP_SUCCESS or HNP_FAILURE is set. | 0 |
| 7 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 6 | TMR_RST | Timer reset. Writing one to this bit resets TMR_CNT to 0. This provides a single bit control for the software to restart the timer when the timer is enabled. | 0 |
| 5 | TMR_EN | Timer enable. When set, TMR_CNT increments. When cleared, TMR_CNT is reset to 0. | 0 |

**Table 187. OTG Status Control register (OTGStCtrl - address 0xE010 0110) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 4 | TMR_MODE | Timer mode selection.<br>0: monoshot<br>1: free running | 0 |
| 3:2 | TMR_SCALE | Timer scale selection. This field determines the duration of each timer count.<br>00: 10 μs (100 KHz)<br>01: 100 μs (10 KHz)<br>10: 1000 μs (1 KHz)<br>11: Reserved | 0x0 |
| 1:0 | PORT_FUNC | Controls port function. Bit 0 is set or cleared by hardware when B_HNP_TRACK or A_HNP_TRACK is set and HNP succeeds. See Section 15–7. Bit 1 is reserved. | - |

## 6.6 OTG Timer Register (OTGTmr - 0xE010 0114)

**Table 188. OTG Timer register (OTGTmr - address 0xE010 0114) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:16 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 15:0 | TIMEOUT_CNT | The TMR interrupt is set when TMR_CNT reaches this value. | 0xFFFF |

## 6.7 OTG Clock Control Register (OTGClkCtrl - 0xE010 0FF4)

This register controls the clocking of the OTG controller. Whenever software wants to access the registers, the corresponding clock control bit needs to be set. The software does not have to repeat this exercise for every register access, provided that the corresponding OTGClkCtrl bits are already set.

**Table 189. OTG_clock_control register (OTG_clock_control - address 0xE010 0FF4) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:5 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | AHB_CLK_EN | | AHB master clock enable | 0 |
| | | 0 | Disable the AHB clock. | |
| | | 1 | Enable the AHB clock. | |
| 3 | OTG_CLK_EN | | OTG clock enable | 0 |
| | | 0 | Disable the OTG clock. | |
| | | 1 | Enable the OTG clock. | |
| 2 | I2C_CLK_EN | | I2C clock enable | 0 |
| | | 0 | Disable the I$^2$C clock. | |
| | | 1 | Enable the I$^2$C clock. | |

Table 189. OTG_clock_control register (OTG_clock_control - address 0xE010 0FF4) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 1 | DEV_CLK_EN | | Device clock enable | 0 |
| | | 0 | Disable the Device clock. | |
| | | 1 | Enable the Device clock. | |
| 0 | HOST_CLK_EN | | Host clock enable | 0 |
| | | 0 | Disable the Host clock. | |
| | | 1 | Enable the Host clock. | |

## 6.8 OTG Clock Status Register (OTGClkSt - 0xE010 0FF8)

This register holds the clock availability status. When enabling a clock via OTGClkCtrl, software should poll the corresponding bit in this register. If it is set, then software can go ahead with the register access. Software does not have to repeat this exercise for every access, provided that the OTGClkCtrl bits are not disturbed.

Table 190. OTG_clock_status register (OTGClkSt - address 0xE010 0FF8) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:5 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 4 | AHB_CLK_ON | | AHB master clock status. | 0 |
| | | 0 | AHB clock is not available. | |
| | | 1 | AHB clock is available. | |
| 3 | OTG_CLK_ON | | OTG clock status. | 0 |
| | | 0 | OTG clock is not available. | |
| | | 1 | OTG clock is available. | |
| 2 | I2C_CLK_ON | | I2C clock status. | 0 |
| | | 0 | I2C clock is not available. | |
| | | 1 | I2C clock is available. | |
| 1 | DEV_CLK_ON | | Device clock status. | 0 |
| | | 0 | Device clock is not available. | |
| | | 1 | Device clock is available. | |
| 0 | HOST_CLK_ON | | Host clock status. | 0 |
| | | 0 | Host clock is not available. | |
| | | 1 | Host clock is available. | |

## 6.9 I2C Receive Register (I2C_RX - 0xE010 0300)

This register is the top byte of the receive FIFO. The receive FIFO is 4 bytes deep. The Rx FIFO is flushed by a hard reset or by a soft reset (I2C_CTL bit 7). Reading an empty FIFO gives unpredictable data results.

Table 191. I2C Receive register (I2C_RX - address 0xE010 0300) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | RX Data | Receive data. | - |

## 6.10 I2C Transmit Register (I2C_TX - 0xE010 0300)

This register is the top byte of the transmit FIFO. The transmit FIFO is 4 bytes deep.

The Tx FIFO is flushed by a hard reset, soft reset (I2C_CTL bit 7) or if an arbitration failure occurs (I2C_STS bit 3). Data writes to a full FIFO are ignored.

I2C_TX must be written for both write and read operations to transfer each byte. Bits [7:0] are ignored for master-receive operations. The master-receiver must write a dummy byte to the TX FIFO for each byte it expects to receive in the RX FIFO. When the STOP bit is set or the START bit is set to cause a RESTART condition on a byte written to the TX FIFO (master-receiver), then the byte read from the slave is not acknowledged. That is, the last byte of a master-receive operation is not acknowledged.

Table 192. I2C Transmit register (I2C_TX - address 0xE010 0300) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:10 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | - |
| 9 | STOP | When 1, issue a STOP condition after transmitting this byte. | - |
| 8 | START | When 1, issue a START condition before transmitting this byte. | - |
| 7:0 | TX Data | Transmit data. | - |

## 6.11 I2C Status Register (I2C_STS - 0xE010 0304)

The I2C_STS register provides status information on the TX and RX blocks as well as the current state of the external buses. Individual bits are enabled as interrupts by the I2C_CTL register and routed to the I2C_USB_INT bit in USBIntSt.

Table 193. I2C status register (I2C_STS - address 0xE010 0304) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:12 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 11 | TFE | | Transmit FIFO Empty. TFE is set when the TX FIFO is empty and is cleared when the TX FIFO contains valid data. | 1 |
| | | 0 | TX FIFO contains valid data. | |
| | | 1 | TX FIFO is empty | |
| 10 | TFF | | Transmit FIFO Full. TFF is set when the TX FIFO is full and is cleared when the TX FIFO is not full. | 0 |
| | | 0 | TX FIFO is not full. | |
| | | 1 | TX FIFO is full | |

**Table 193. I2C status register (I2C_STS - address 0xE010 0304) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 9 | RFE | | Receive FIFO Empty. RFE is set when the RX FIFO is empty and is cleared when the RX FIFO contains valid data. | 1 |
| | | 0 | RX FIFO contains data. | |
| | | 1 | RX FIFO is empty | |
| 8 | RFF | | Receive FIFO Full (RFF). This bit is set when the RX FIFO is full and cannot accept any more data. It is cleared when the RX FIFO is not full. If a byte arrives when the Receive FIFO is full, the SCL is held low until the CPU reads the RX FIFO and makes room for it. | 0 |
| | | 0 | RX FIFO is not full | |
| | | 1 | RX FIFO is full | |
| 7 | SDA | | The current value of the SDA signal. | - |
| 6 | SCL | | The current value of the SCL signal. | - |
| 5 | Active | | Indicates whether the bus is busy. This bit is set when a START condition has been seen. It is cleared when a STOP condition is seen.. | 0 |
| 4 | DRSI | | Slave Data Request Interrupt. Once a transmission is started, the transmitter must have data to transmit as long as it isn’t followed by a STOP condition or it will hold SCL low until more data is available. The Slave Data Request bit is set when the slave transmitter is data-starved. If the slave TX FIFO is empty and the last byte transmitted was acknowledged, then SCL is held low until the CPU writes another byte to transmit. This bit is cleared when a byte is written to the slave Tx FIFO. | 0 |
| | | 0 | Slave transmitter does not need data. | |
| | | 1 | Slave transmitter needs data. | |
| 3 | DRMI | | Master Data Request Interrupt. Once a transmission is started, the transmitter must have data to transmit as long as it isn’t followed by a stop condition or it will hold SCL low until more data is available. The Master Data Request bit is set when the master transmitter is data-starved. If the master TX FIFO is empty and the last byte did not have a STOP condition flag, then SCL is held low until the CPU writes another byte to transmit. This bit is cleared when a byte is written to the master TX FIFO. | 0 |
| | | 0 | Master transmitter does not need data. | |
| | | 1 | Master transmitter needs data. | |
| 2 | NAI | | No Acknowledge Interrupt. After every byte of data is sent, the transmitter expects an acknowledge from the receiver. This bit is set if the acknowledge is not received. It is cleared when a byte is written to the master TX FIFO. | 0 |
| | | 0 | Last transmission received an acknowledge. | |
| | | 1 | Last transmission did not receive an acknowledge. | |

Table 193. I2C status register (I2C_STS - address 0xE010 0304) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 1 | AFI | | Arbitration Failure Interrupt. When transmitting, if the SDA is low when SDAOUT is high, then this I$^2$C has lost the arbitration to another device on the bus. The Arbitration Failure bit is set when this happens. It is cleared by writing a one to bit 1 of the status register. | 0 |
| | | 0 | No arbitration failure on last transmission. | |
| | | 1 | Arbitration failure occurred on last transmission. | |
| 0 | TDI | | Transaction Done Interrupt. This flag is set if a transaction completes successfully. It is cleared by writing a one to bit 0 of the status register. It is unaffected by slave transactions. | 0 |
| | | 0 | Transaction has not completed. | |
| | | 1 | Transaction completed. | |

## 6.12 I2C Control Register (I2C_CTL - 0xE010 0308)

The I2C_CTL register is used to enable interrupts and reset the I$^2$C state machine. Enabled interrupts cause the USB_I2C_INT interrupt output line to be asserted when set.

Table 194. I2C Control register (I2C_CTL - address 0xE010 0308) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:9 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 8 | SRST | | Soft reset. This is only needed in unusual circumstances. If a device issues a start condition without issuing a stop condition. A system timer may be used to reset the I$^2$C if the bus remains busy longer than the time-out period. On a soft reset, the Tx and Rx FIFOs are flushed, I2C_STS register is cleared, and all internal state machines are reset to appear idle. The I2C_CLKHI, I2C_CLKLO and I2C_CTL (except Soft Reset Bit) are NOT modified by a soft reset. | 0 |
| | | 0 | See the text. | |
| | | 1 | Reset the I$^2$C to idle state. Self clearing. | |
| 7 | TFFIE | | Transmit FIFO Not Full Interrupt Enable. This enables the Transmit FIFO Not Full interrupt to indicate that the more data can be written to the transmit FIFO. Note that this is not full. It is intended help the CPU to write to the I$^2$C block only when there is room in the FIFO and do this without polling the status register. | 0 |
| | | 0 | Disable the TFFI. | |
| | | 1 | Enable the TFFI. | |
| 6 | RFDAIE | | Receive Data Available Interrupt Enable. This enables the DAI interrupt to indicate that data is available in the receive FIFO (i.e. not empty). | 0 |
| | | 0 | Disable the DAI. | |
| | | 1 | Enable the DAI. | |
| 5 | REFIE | | Receive FIFO Full Interrupt Enable. This enables the Receive FIFO Full interrupt to indicate that the receive FIFO cannot accept any more data. | 0 |
| | | 0 | Disable the RFFI. | |
| | | 1 | Enable the RFFI. | |

**Table 194. I2C Control register (I2C_CTL - address 0xE010 0308) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 4 | DRSIE | | Slave Transmitter Data Request Interrupt Enable. This enables the DRSI interrupt which signals that the slave transmitter has run out of data and the last byte was acknowledged, so the SCL line is being held low. | 0 |
| | | 0 | Disable the DRSI interrupt. | |
| | | 1 | Enable the DRSI interrupt. | |
| 3 | DRMIE | | Master Transmitter Data Request Interrupt Enable. This enables the DRMI interrupt which signals that the master transmitter has run out of data, has not issued a STOP, and is holding the SCL line low. | 0 |
| | | 0 | Disable the DRMI interrupt. | |
| | | 1 | Enable the DRMI interrupt. | |
| 2 | NAIE | | Transmitter No Acknowledge Interrupt Enable. This enables the NAI interrupt signalling that transmitted byte was not acknowledged. | 0 |
| | | 0 | Disable the NAI. | |
| | | 1 | Enable the NAI. | |
| 1 | AFIE | | Transmitter Arbitration Failure Interrupt Enable. This enables the AFI interrupt which is asserted during transmission when trying to set SDA high, but the bus is driven low by another device. | 0 |
| | | 0 | Disable the AFI. | |
| | | 1 | Enable the AFI. | |
| 0 | TDIE | | Transmit Done Interrupt Enable. This enables the TDI interrupt signalling that this I$^2$C issued a STOP condition. | 0 |
| | | 0 | Disable the TDI interrupt. | |
| | | 1 | Enable the TDI interrupt. | |

## 6.13 I2C Clock High Register (I2C_CLKHI - 0xE010 030C)

The CLK register holds a terminal count for counting 48 MHz clock cycles to create the high period of the slower I$^2$C serial clock, SCL.

**Table 195. I2C_CLKHI register (I2C_CLKHI - address 0xE010 030C) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | CDHI | Clock divisor high. This value is the number of 48 MHz clocks the serial clock (SCL) will be high. | 0xB9 |

## 6.14 I2C Clock Low Register (I2C_CLKLO - 0xE010 0310)

The CLK register holds a terminal count for counting 48 MHz clock cycles to create the low period of the slower I$^2$C serial clock, SCL.

**Table 196. I2C_CLKLO register (I2C_CLKLO - address 0xE010 0310) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:0 | CDLO | Clock divisor low. This value is the number of 48 MHz clocks the serial clock (SCL) will be low. | 0xB9 |

### 6.15 Interrupt handling

The interrupts set in the OTGIntSt register are set and cleared during HNP switching. All OTG related interrupts are routed separately to the VIC.

I2C related interrupts are set in the I2C_STS register and routed, if enabled by I2C_CTL, to the USB_I2C_INT bit.

For more details on the interrupts created by device controller, see the USB device chapter. For interrupts created by the host controllers, see the OHCI specification.

**Remark:** During the HNP switching between host and device with the OTG stack active, an action may raise several levels of interrupts. It is advised to let the OTG stack initiate any actions based on interrupts and ignore device and host level interrupts. This means that during HNP switching, the OTG stack provides the communication to the host and device controllers.

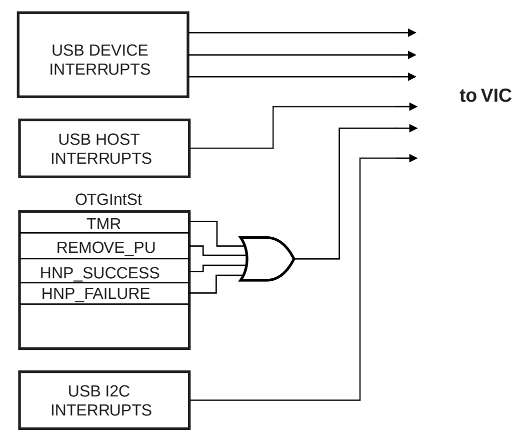


Fig 52. USB OTG interrupt handling

## 7. HNP support

This section describes the hardware support for the Host Negotiation Protocol (HNP) provided by the OTG controller.

When two dual-role OTG devices are connected to each other, the plug inserted into the mini-AB receptacle determines the default role of each device. The device with the mini-A plug inserted becomes the default Host (A-device), and the device with the mini-B plug inserted becomes the default Peripheral (B-device).

Once connected, the default Host (A-device) and the default Peripheral (B-device) can switch Host and Peripheral roles using HNP.

The context of the OTG controller operation is shown in Figure 15–53. Each controller (Host, Device, or OTG) communicates with its software stack through a set of status and control registers and interrupts. In addition, the OTG software stack communicates with the external OTG transceiver through the I2C interface and the external transceiver interrupt signal.

The OTG software stack is responsible for implementing the HNP state machines as described in the On-The-Go Supplement to the USB 2.0 Specification.

The OTG controller hardware provides support for some of the state transitions in the HNP state machines as described in the following subsections.

The USB state machines, the HNP switching, and the communications between the USB controllers are described in more detail in the following documentation:

- *USB OHCI specification*
- *USB OTG supplement, version 1.2*
- *USB 2.0 specification*
- *ISP1302 datasheet and usermanual*

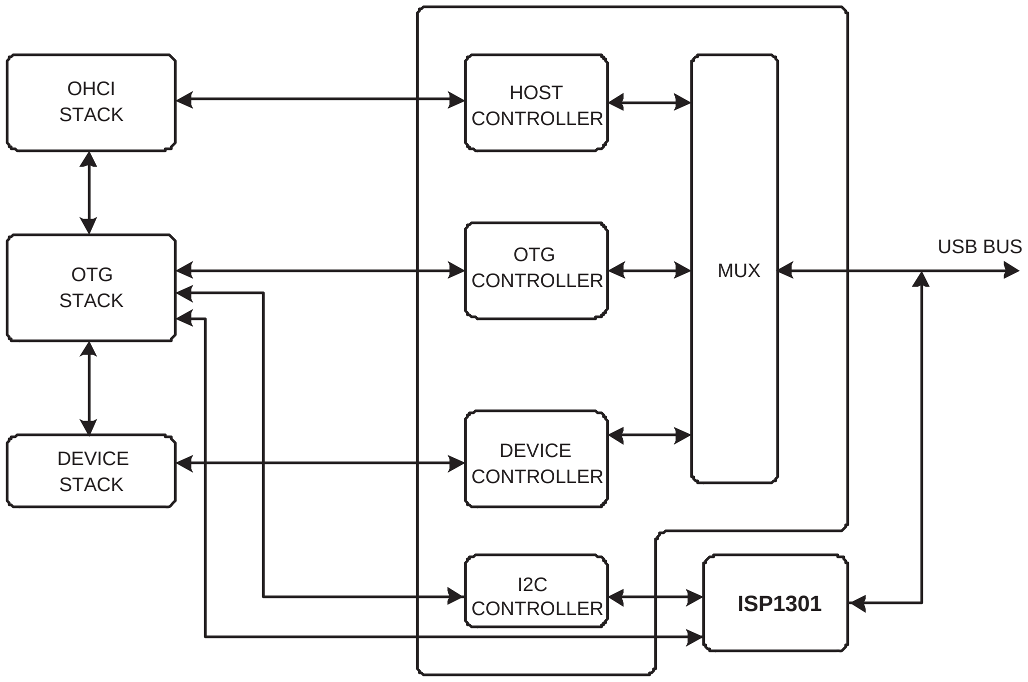


**Fig 53. USB OTG controller with software stack**

## 7.1 B-device: peripheral to host switching

In this case, the default role of the OTG controller is peripheral (B-device), and it switches roles from Peripheral to Host.

The On-The-Go Supplement defines the behavior of a dual-role B-device during HNP using a state machine diagram. The OTG software stack is responsible for implementing all of the states in the Dual-Role B-Device State Diagram.

The OTG controller hardware provides support for the state transitions between the states b_peripheral, b_wait_acon, and b_host in the Dual-Role B-Device state diagram. Setting B_HNP_TRACK in the OTGStCtrl register enables hardware support for the B-device switching from peripheral to host. The hardware actions after setting this bit are shown in Figure 15–54.

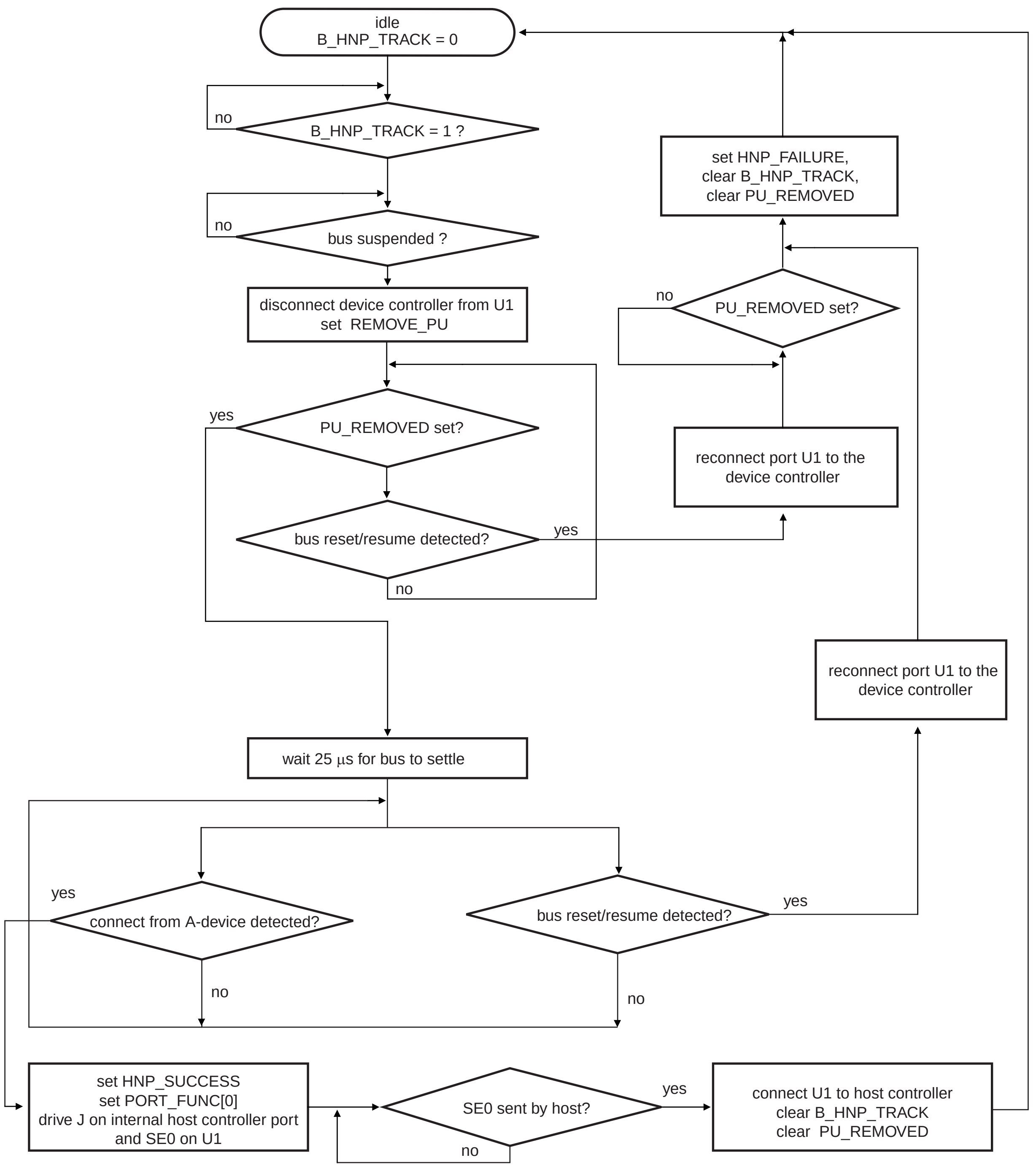

idle
B_HNP_TRACK = 0
no
B_HNP_TRACK = 1 ?
no
bus suspended ?
disconnect device controller from U1
set REMOVE_PU
yes
PU_REMOVED set?
bus reset/resume detected?
yes
no
set HNP_FAILURE,
clear B_HNP_TRACK,
clear PU_REMOVED
no
PU_REMOVED set?
reconnect port U1 to the device controller
reconnect port U1 to the device controller
wait 25 μs for bus to settle
yes
connect from A-device detected?
no
bus reset/resume detected?
yes
no
set HNP_SUCCESS
set PORT_FUNC[0]
drive J on internal host controller port and SE0 on U1
SE0 sent by host?
yes
no
connect U1 to host controller
clear B_HNP_TRACK
clear PU_REMOVED

**Fig 54. Hardware support for B-device switching from peripheral state to host state**

Figure 15–55 shows the actions that the OTG software stack should take in response to the hardware actions setting REMOVE_PU, HNP_SUCCESS, AND HNP_FAILURE. The relationship of the software actions to the Dual-Role B-Device states is also shown. B-device states are in bold font with a circle around them.

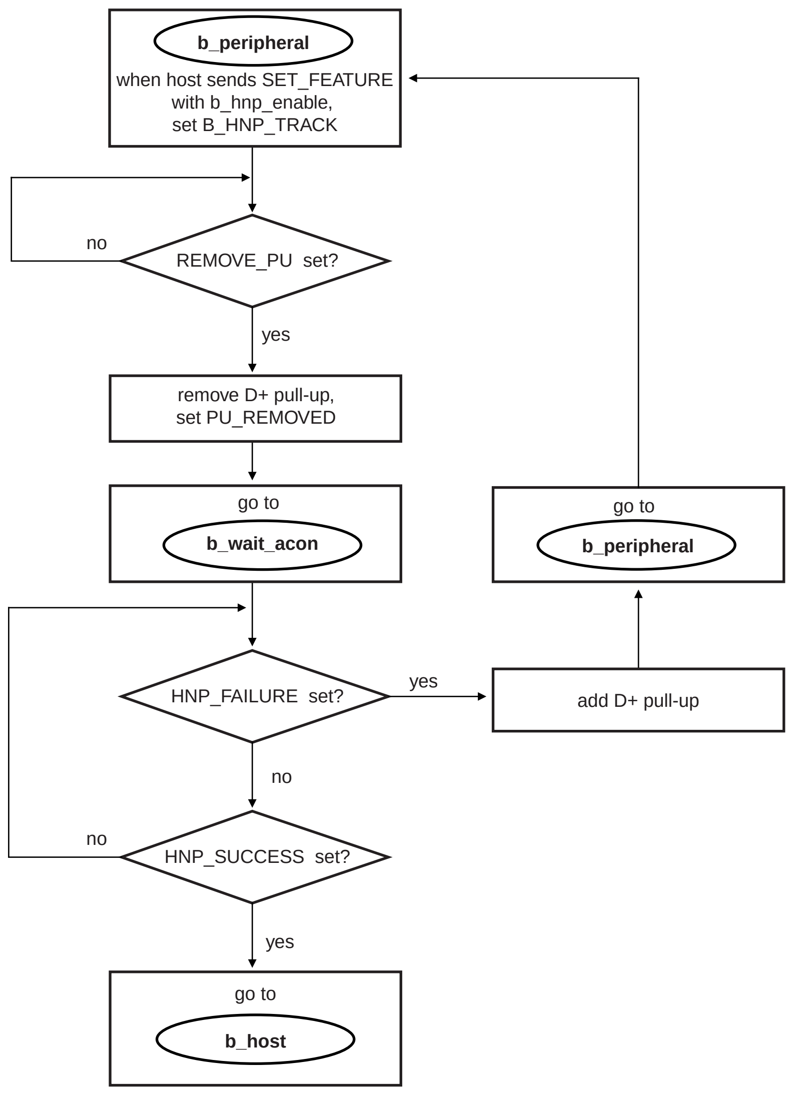


**Fig 55. State transitions implemented in software during B-device switching from peripheral to host**

Note that only the subset of B-device HNP states and state transitions supported by hardware are shown. Software is responsible for implementing all of the HNP states.

Figure 15–55 may appear to imply that the interrupt bits such as REMOVE_PU should be polled, but this is not necessary if the corresponding interrupt is enabled.

Following are code examples that show how the actions in Figure 15–55 are accomplished. The examples assume that ISP1301 is being used as the external OTG transceiver.

### Remove D+ pull-up

```
/* Remove D+ pull-up through ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x007; // Send OTG Control (Clear) register address
OTG_I2C_TX = 0x201; // Clear DP_PULLUP bit, send STOP condition
```

```
/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;
```

**Add D+ pull-up**

```
/* Add D+ pull-up through ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x006; // Send OTG Control (Set) register address
OTG_I2C_TX = 0x201; // Set DP_PULLUP bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;
```

## 7.2 A-device: host to peripheral HNP switching

In this case, the role of the OTG controller is host (A-device), and the A-device switches roles from host to peripheral.

The On-The-Go Supplement defines the behavior of a dual-role A-device during HNP using a state machine diagram. The OTG software stack is responsible for implementing all of the states in the Dual-Role A-Device State Diagram.

The OTG controller hardware provides support for the state transitions between a_host, a_suspend, a_wait_vfall, and a_peripheral in the Dual-Role A-Device state diagram. Setting A_HNP_TRACK in the OTGStCtrl register enables hardware support for switching the A-device from the host state to the device state. The hardware actions after setting this bit are shown in Figure 15–56.

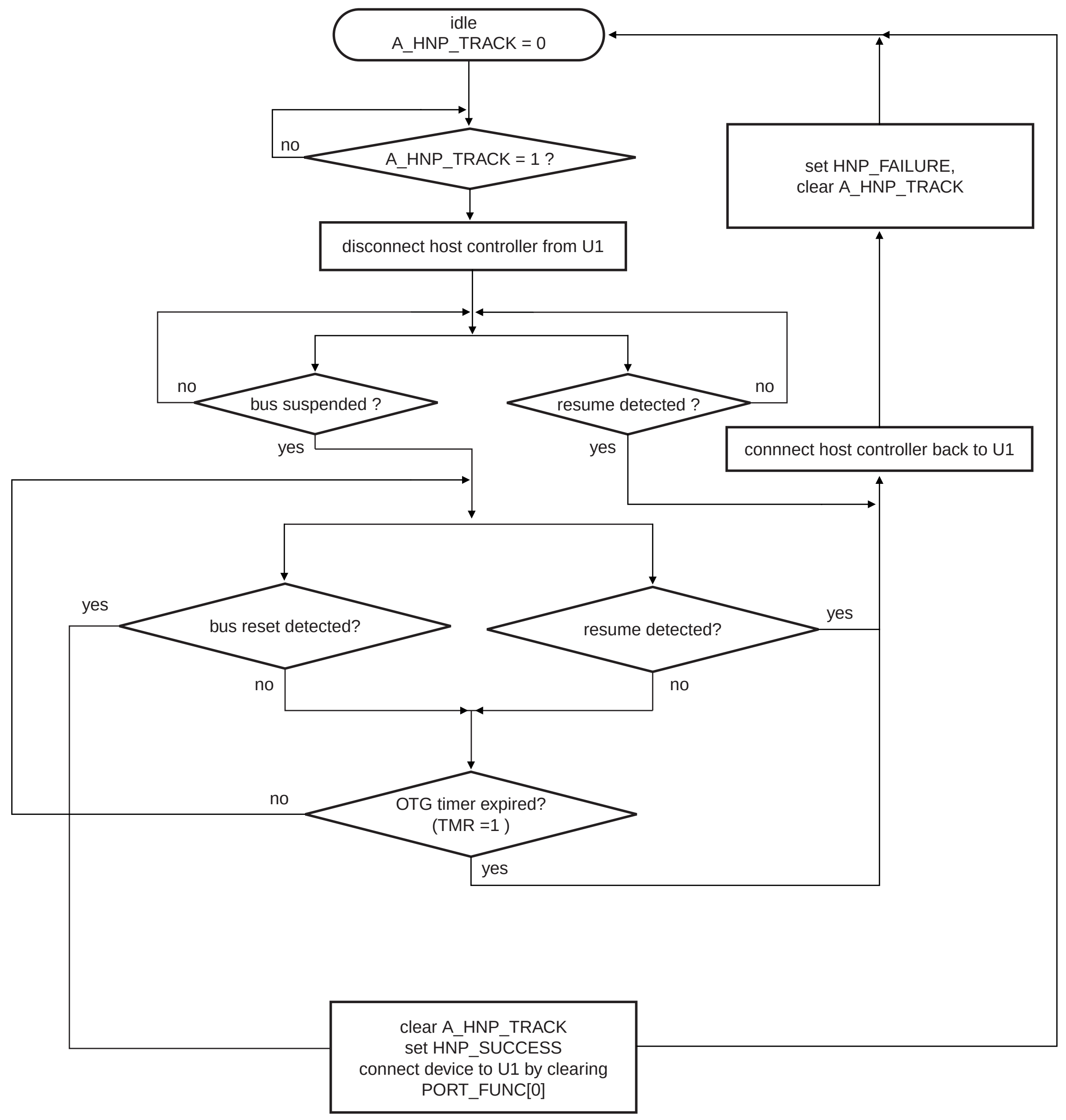


**Fig 56. Hardware support for A-device switching from host state to peripheral state**

Figure 15–57 shows the actions that the OTG software stack should take in response to the hardware actions setting TMR, HNP_SUCCESS, and HNP_FAILURE. The relationship of the software actions to the Dual-Role A-Device states is also shown. A-device states are shown in bold font with a circle around them.

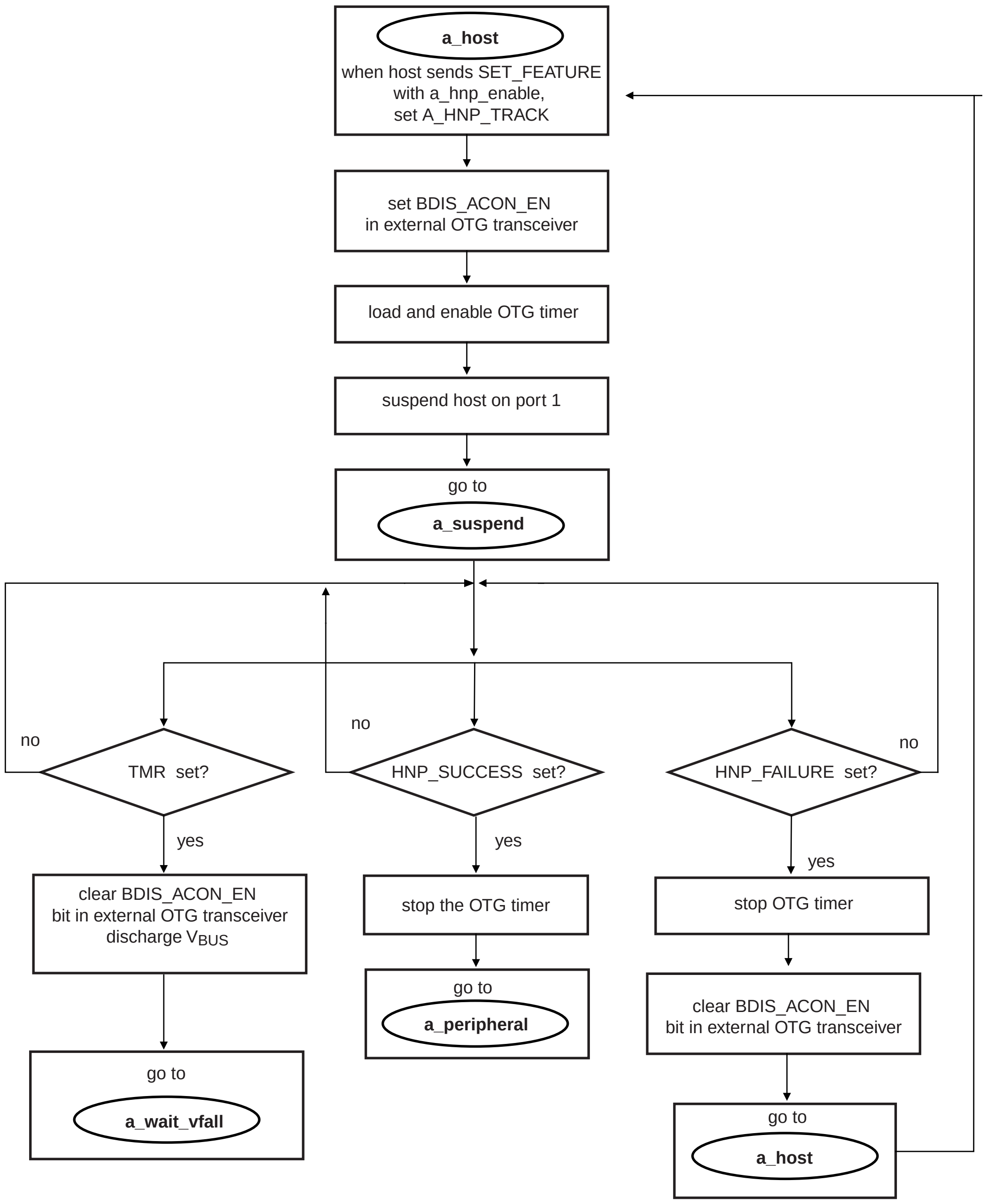

a_host
when host sends SET_FEATURE with a_hnp_enable, set A_HNP_TRACK
set BDIS_ACON_EN in external OTG transceiver
load and enable OTG timer
suspend host on port 1
go to
a_suspend
no
TMR set?
yes
clear BDIS_ACON_EN bit in external OTG transceiver discharge V_BUS
go to
a_wait_vfall
no
HNP_SUCCESS set?
yes
stop the OTG timer
go to
a_peripheral
no
HNP_FAILURE set?
yes
stop OTG timer
clear BDIS_ACON_EN bit in external OTG transceiver
go to
a_host

**Fig 57. State transitions implemented in software during A-device switching from host to peripheral**

Note that only the subset of A-device HNP states and state transitions supported by hardware are shown. Software is responsible for implementing all of the HNP states.

Figure 15–57 may appear to imply that the interrupt bits such as TMR should be polled, but this is not necessary if the corresponding interrupt is enabled.

Following are code examples that show how the actions in Figure 15–57 are accomplished. The examples assume that ISP1301 is being used as the external OTG transceiver.

### Set BDIS_ACON_EN in external OTG transceiver

```
/* Set BDIS_ACON_EN in ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x004; // Send Mode Control 1 (Set) register address
OTG_I2C_TX = 0x210; // Set BDIS_ACON_EN bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;
```

### Clear BDIS_ACON_EN in external OTG transceiver

```
/* Set BDIS_ACON_EN in ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x005; // Send Mode Control 1 (Clear) register address
OTG_I2C_TX = 0x210; // Clear BDIS_ACON_EN bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;
```

### Discharge $V_{BUS}$

```
/* Clear the VBUS_DRV bit in ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x007; // Send OTG Control (Clear) register address
OTG_I2C_TX = 0x220; // Clear VBUS_DRV bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;

/* Set the VBUS_DISCHRG bit in ISP1301 */
OTG_I2C_TX = 0x15A; // Send ISP1301 address, R/W=0
OTG_I2C_TX = 0x006; // Send OTG Control (Set) register address
OTG_I2C_TX = 0x240; // Set VBUS_DISCHRG bit, send STOP condition

/* Wait for TDI to be set */
while (!(OTG_I2C_STS & TDI));

/* Clear TDI */
OTG_I2C_STS = TDI;
```

### Load and enable OTG timer

```
/* The following assumes that the OTG timer has previously been  */

/* configured for a time scale of 1 ms (TMR_SCALE = “10”)         */
/* and monoshot mode (TMR_MODE = 0)                               */

/* Load the timeout value to implement the a_aidl_bdis_tmr timer */
/*     the minimum value is 200 ms                                */
OTG_TIMER = 200;

/* Enable the timer */
OTG_STAT_CTRL |= TMR_EN;
```

### Stop OTG timer

```
/* Disable the timer - causes TMR_CNT to be reset to 0  */
OTG_STAT_CTRL &= ~TMR_EN;

/* Clear TMR interrupt */
OTG_INT_CLR = TMR;
```

### Suspend host on port 1

```
/* Write to PortSuspendStatus bit to suspend host port 1 -                */
/* this example demonstrates the low-level action software needs to take. */
/* The host stack code where this is done will be somewhat more involved. */
HC_RH_PORT_STAT1 = PSS;
```

## 8. Clocking and power management

The OTG controller clocking is shown in Figure 15–58. Note that the host controller is not implemented on the LPC2927 and LPC2929.

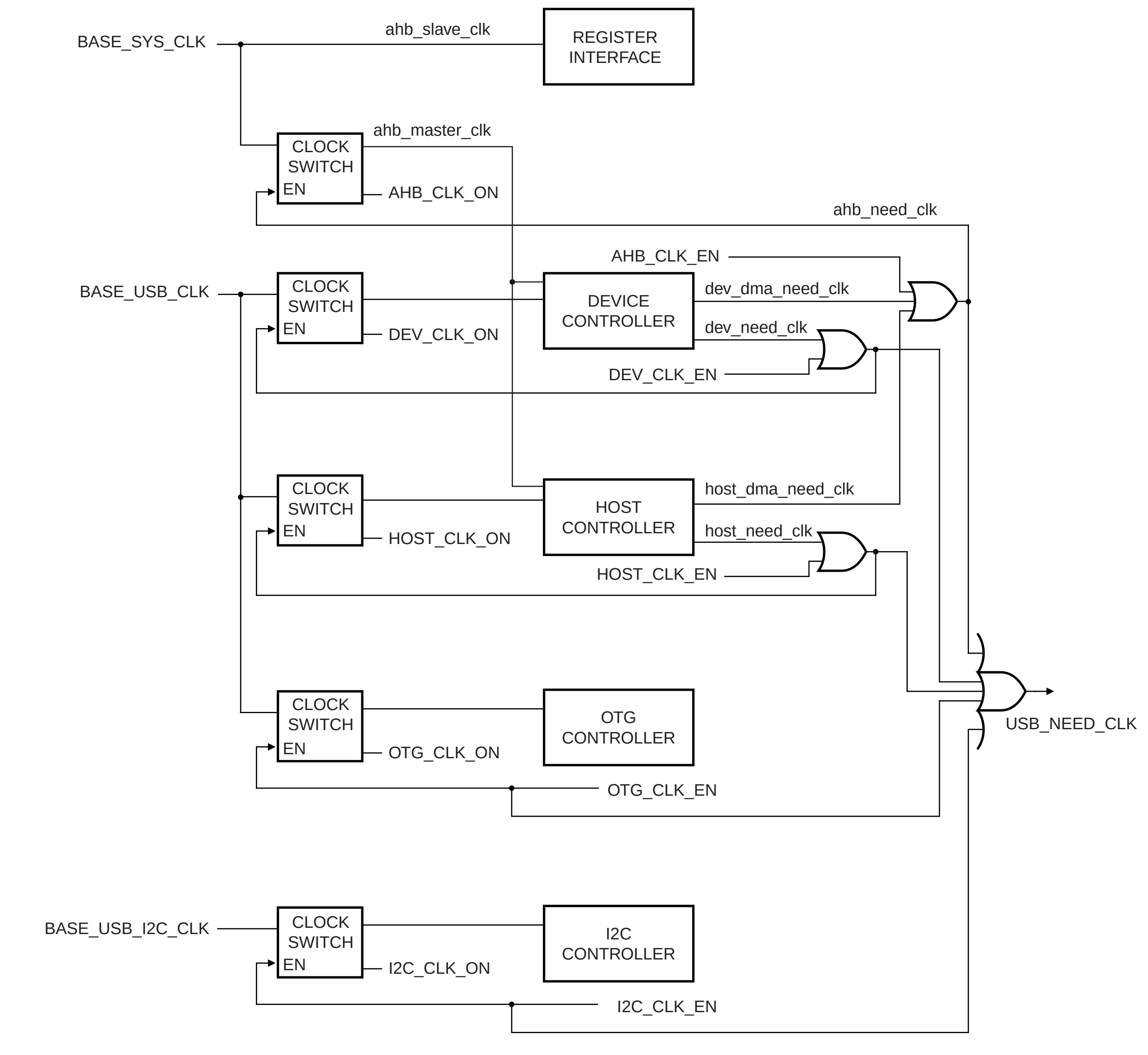


**Fig 58. Clocking and power control**

A clock switch controls each clock with the exception of ahb_slave_clk. When the enable of the clock switch is asserted, its clock output is turned on and its CLK_ON output is asserted. The CLK_ON signals are observable in the OTGClkSt register.

To conserve power, the clocks to the Device, host, OTG, and I2C controllers can be disabled when not in use by disabling the clocks in the CGU1 (see Table 3–15).

When software wishes to access registers in one of the controllers, it should first ensure that the respective controller's 48 MHz clock is enabled by setting its CLK_EN bit in the OTGClkCtrl register and then poll the corresponding CLK_ON bit in OTGClkSt until set. Once set, the controller's clock will remain enabled until CLK_EN is cleared by software. Accessing the register of a controller when its 48 MHz clock is not enabled will result in a data abort exception.

## 8.1 Device clock request signals

The Device controller has two clock request signals, dev_need_clk and dev_dma_need_clk. When asserted, these signals turn on the device's 48 MHz clock and ahb_master_clk respectively.

The dev_need_clk signal is asserted while the device is not in the suspend state, or if the device is in the suspend state and activity is detected on the USB bus. The dev_need_clk signal is de-asserted if a disconnect is detected (CON bit is cleared in the SIE Get Device Status register – Section 13–11.7). This signal allows DEV_CLK_EN to be cleared during normal operation when software does not need to access the Device controller registers – the Device will continue to function normally and automatically shut off its clock when it is suspended or disconnected.

The dev_dma_need_clk signal is asserted on any Device controller DMA access to memory. Once asserted, it remains active for 2 ms (2 frames), to help assure that DMA throughput is not affected by any latency associated with re-enabling ahb_master_clk. 2 ms after the last DMA access, dev_dma_need_clk is de-asserted to help conserve power. This signal allows AHB_CLK_EN to be cleared during normal operation.

## 8.2 Host clock request signals

The Host controller has two clock request signals, host_need_clk and host_dma_need_clk. When asserted, these signals turn on the host's 48 MHz clock and ahb_master_clk respectively.

The host_need_clk signal is asserted while the Host controller functional state is not UsbSuspend, or if the functional state is UsbSuspend and resume signaling or a disconnect is detected on the USB bus. This signal allows HOST_CLK_EN to be cleared during normal operation when software does not need to access the Host controller registers – the Host will continue to function normally and automatically shut off its clock when it goes into the UsbSuspend state.

The host_dma_need_clk signal is asserted on any Host controller DMA access to memory. Once asserted, it remains active for 2 ms (2 frames), to help assure that DMA throughput is not affected by any latency associated with re-enabling ahb_master_clk. 2 ms after the last DMA access, host_dma_need_clk is de-asserted to help conserve power. This signal allows AHB_CLK_EN to be cleared during normal operation.

## 8.3 Power-down mode support

Before Power-down mode can be entered when USBWAKE is set, USB_NEED_CLK must be de-asserted. This is accomplished by clearing all of the CLK_EN bits in OTGClkCtrl and putting the Host controller into the UsbSuspend functional state. If it is necessary to wait for either of the dma_need_clk signals or the dev_need_clk to be de-asserted, the status of USB_NEED_CLK can be polled in the USBIntSt register to determine when they have all been de-asserted.

## 9. USB OTG controller initialization

The LPC29xx OTG device controller initialization includes the following steps:

1. Enable the USB device block through the PMU.
2. Configure and enable the USB PLL in the CGU1, see Table 3–15.
3. Enable the desired controller clocks by setting their respective CLK_EN bits in the USBClkCtrl register. Poll the corresponding CLK_ON bits in the USBClkSt register until they are set.
4. Enable the desired USB pin functions by writing to the corresponding port control registers, see Table 6–58.
5. Follow the appropriate steps in Section 13–12 “USB device controller initialization” to initialize the device controller.

# UM10316

# Chapter 16: LPC29xx General Purpose Input/Output (GPIO)

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. Available ports depend on the pin configuration for each part.

Table 197. GPIO ports available

| Part number | GPIO port 0 | GPIO port 1 | GPIO port 2 | GPIO port 3 | GPIO port 4 | GPIO port 5 | GPIO port 5/ USB |
|---|---|---|---|---|---|---|---|
| LPC2917/19/01 | P0[31:0] | P1[31:0] | P2[27:0] | P3[15:0] | - | - | - |
| LPC2921/23/25 | P0[31:0] | P1[27:0] | - | - | - | - | P5[19:18] |
| LPC2926/27/29 | P0[31:0] | P1[27:0] | P2[27:0] | P3[15:0] | - | - | P5[19:18] |
| LPC2930 | P0[31:0] | P1[27:0] | P2[27:0] | P3[15:0] | P4[23:0] | P5[15:0] | P5[19:16] |
| LPC2939 | P0[31:0] | P1[27:0] | P2[27:0] | P3[15:0] | P4[23:0] | P5[15:0] | P5[19:16] |

## 2. GPIO functional description

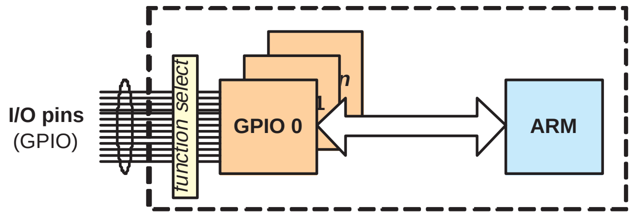


Fig 59. Schematic representation of the GPIO

Each General-Purpose I/O block GPIO provides control over up to 32 port pins. The data direction (in/out) and output level of each port pin can be programmed individually.

If a port pin is to be used it must first be routed to an I/O pin so that it is available externally. This part of the configuration is done via the SCU. See Section 6–3.1 for information on mapping of GPIO port pins to I/O pins. GPIO port pinning can be found in Ref. 31–1.

A number of points should be noted in regard to SCU mapping of GPIO pins:

- If an input port is not mapped through the SCU to an external I/O pin it is assigned a logical 0.
- If an output port is not mapped through the SCU to an external I/O pin it is left dangling; i.e. not connected.

The GPIO pins can also be used in an open-drain configuration. In this configuration, multiple devices can communicate on one signal line in any direction (e.g. bi-directionally).

The signal line is normally pulled up to a HIGH voltage level (logic 1) by an external resistor. Each of the devices connected to the signal line can either drive the signal line to a LOW voltage level (logic 0) or stay at high impedance (open-drain). If none of the devices drives the signal line to a LOW voltage level the signal line is pulled-up by the resistor (logic 1).

Devices in high-impedance can also read the value of the signal line to detect a logic 0 or logic 1. This allows communication in multiple directions.

The open-drain configuration is achieved by:

- Initially:
  - Configuring the pin direction as input (high impedance/open drain).
  - Setting the pin output to a LOW voltage level (logic 0).
- Configuring the pin direction as output to drive a LOW voltage level (logic 0).
- Configuring the pin direction as input to provide an open drain. In this case the other devices and external resistor determine the voltage level. The actual level (logic 0 or logic 1) can be read from the GPIO pin.

## 3. Register overview

**Table 198. Register overview: GPIO (base address: 0xE004 A000 (GPIO0), 0xE004 B000 (GPIO1), 0xE004 C000 (GPIO2), 0xE004 D000 (GPIO3), 0xE004 E000 (GPIO4), 0xE004 F000 (GPIO5))**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| PINS | R | 0x0 | Port input register | - | Table 16–199 |
| OR | R/W | 0x4 | Port output register | 0x0000 0000 | Table 16–200 |
| DR | R/W | 0x8 | Port direction register | 0x0000 0000 | Table 16–201 |

### 3.1 GPIO port input register

The port input register is used to reflect the synchronized input level on each I/O pin individually. In the case of writing to the port input register, the contents are written into the port output register.

Table 16–199 shows the bit assignment of the PINS register. Bits for for unavailable ports are reserved (see Table 16–197), do not modify, and read as logic 0.

**Table 199. PINS register bit description (PINS0 to 5, addresses 0xE004 A000 (GPIO0), 0xE004 B000 (GPIO1), 0xE004 C000 (GPIO2), 0xE004 D000 (GPIO3), 0xE004 E000 (GPIO4), 0xE004 F000 (GPIO5))**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | PINS[31] | R/W | 1 | Pn[31] input pin is HIGH |
| | | | 0 | Pn[31] input pin is LOW |
| 30:1 | ... | ... | ... | ... |
| 0 | PINS[0] | R/W | 1 | Pn[0] input pin is HIGH |
| | | | 0 | Pn[0] input pin is LOW |

## 3.2 GPIO port output register

The port output register is used to define the output level on each I/O pin individually if this pin has been configured as an output by the port direction register. If the port input register is written to the port output register is written to as well.

Table 16–200 shows the bit assignment of the OR register. Bits for for unavailable ports are reserved (see Table 16–197), do not modify, and read as logic 0.

**Table 200. OR register bit description (OR0 to 5, addresses 0xE004 A004 (GPIO0), 0xE004 B004 (GPIO1), 0xE004 C004 (GPIO2), 0xE004 D004 (GPIO3), 0xE004 E004 (GPIO4), 0xE004 F004 (GPIO5))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | OR[31] | R/W | 1 | If configured as an output, pin Pn[31] is driven HIGH |
| | | | 0* | If configured as an output, pin Pn[31] is driven LOW |
| 30:1 | ... | ... | ... | ... |
| 0 | OR[0] | R/W | 1 | If configured as an output, pin Pn[0] is driven HIGH |
| | | | 0* | If configured as an output, pin Pn[0] is driven LOW |

## 3.3 GPIO port direction register

The port direction register is used to individually control each I/O pin output-driver enable. If the port is configured as input, see Table 6–59 to configure the appropriate pad type.

Table 16–201 shows the bit assignment of the DR register. Bits for for unavailable ports are reserved (see Table 16–197), do not modify, and read as logic 0.

**Table 201. DR register bit description (DR0 to 5, addresses 0xE004 A008 (GPIO0), 0xE004 B008 (GPIO1), 0xE004 C008 (GPIO2), 0xE004 D008 (GPIO3), 0xE004 E008 (GPIO4), 0xE004 F008 (GPIO5))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | DR[31] | R/W | 1 | Pin Pn[31] is configured as an output |
| | | | 0* | Pin Pn[31] is configured as an input |
| 30:1 | ... | ... | ... | ... |
| 0 | DR[0] | R/W | 1 | Pin Pn[0] is configured as an output |
| | | | 0* | Pin Pn[0] is configured as an input |

# UM10316

# Chapter 17: LPC29xx timer 0/1/2/3

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. Note that capture pins CAP0 and CAP1 on timer 1 are not pinned out for LPC2926/27/29 and LPC2921/23/25.

## 2. Timer functional description

The timers can be used to measure the time between events. An interrupt can be generated:

- When a predetermined period has elapsed (match functionality: see section Section 17–4)
- On an external trigger (capture functionality: see section Section 17–4.1)

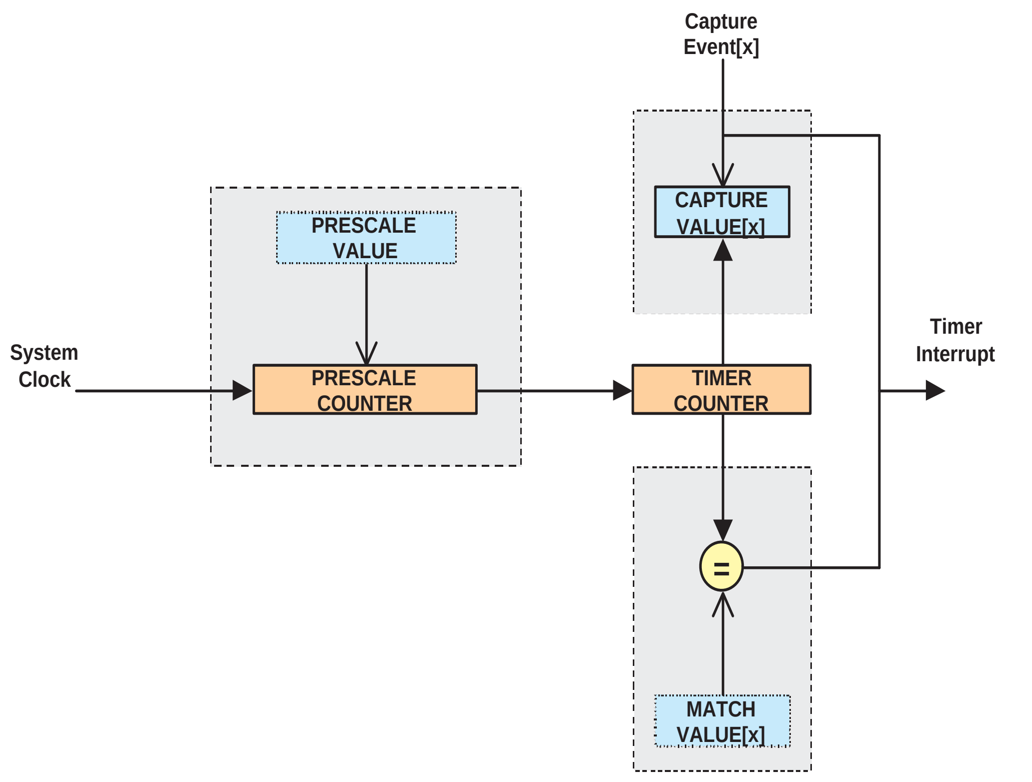


**Fig 60. Timer architecture**

The timer runs at a frequency derived from the input system clock by dividing it by the prescale value. The prescale value is programmed by writing to the PR register.

## 3. Timer counter and interrupt timing

Each timer consists of a prescale counter (PR register) and a timer counter (TC register). The prescale counter is incremented at every cycle of the system clock. As soon as the prescale counter matches the prescale value contained in the PV register it is reset to 0 and the timer counter is incremented. Both events occur at the next system clock cycle, so effectively the timer counter is incremented at every prescale-value+1 cycle of the system clock.

When the timer counter equals a match value (MRx registers) the timer performs the configured match action (MCR register). For a reset on match the timer counter is reset at the next prescaled clock (see Figure 17–61): for a stop-on-match the prescale and timer counters stop immediately (see Figure 17–62).

If interrupts are enabled and an interrupt condition occurs (match value reached or capture event received) the timer generates an interrupt. This interrupt is generated at the next system clock cycle (see Figure 17–61 and Figure 17–62).

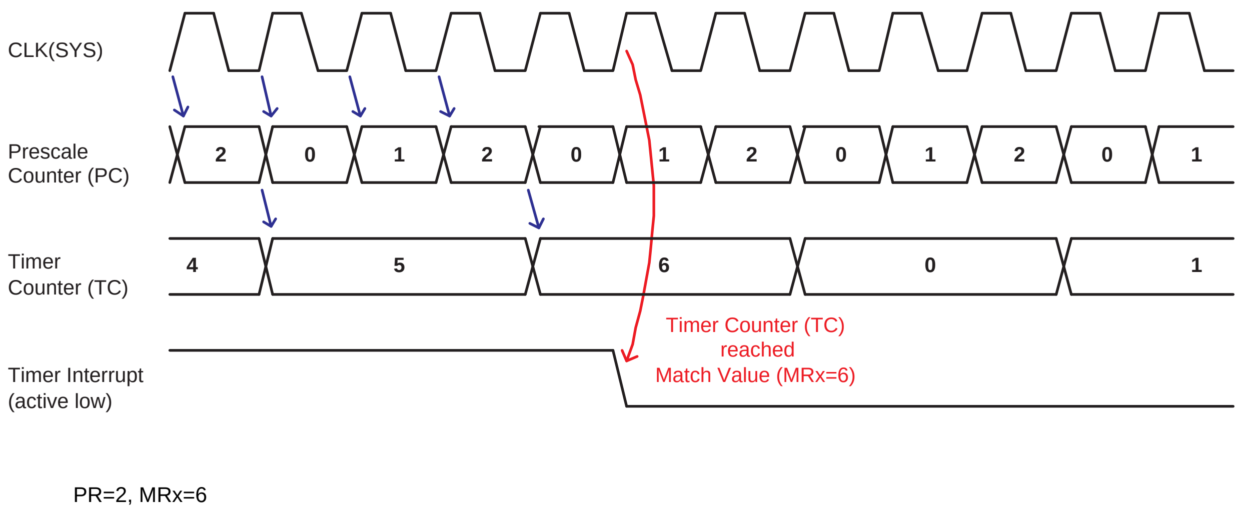


Fig 61. Reset-on-match timing

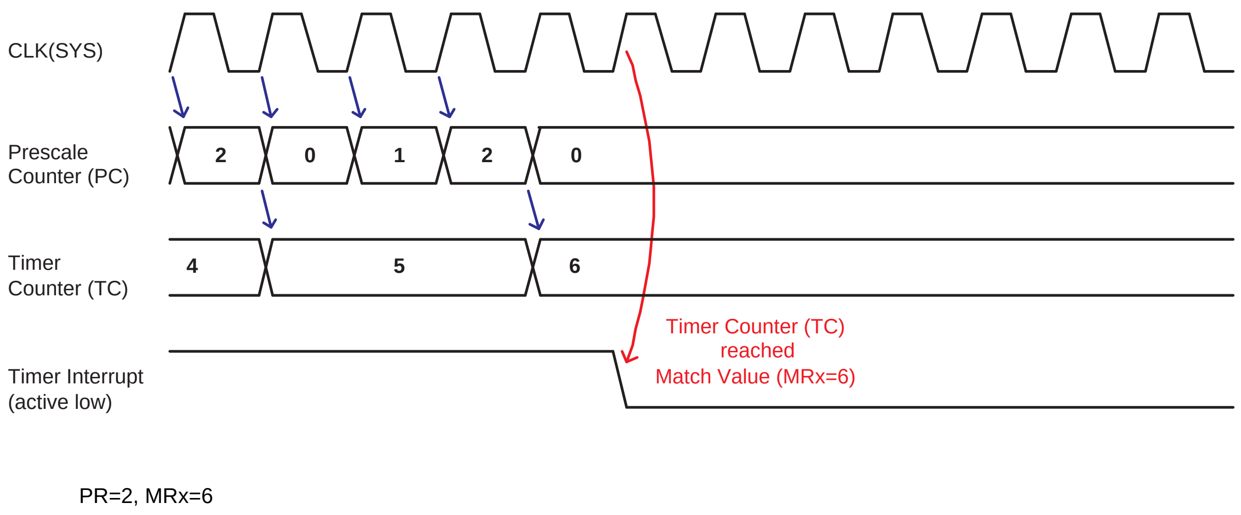

CLK(SYS)
Prescale Counter (PC)
2
0
1
2
0
Timer Counter (TC)
4
5
6
Timer Interrupt (active low)
Timer Counter (TC) reached Match Value (MRx=6)
PR=2, MRx=6

**Fig 62. Stop-on-match timing**

## 4. Timer match functionality

The timer block contains four match circuits, each of which can be programmed with an individual match value and a specific action-on-match. Once the counter value matches one of the programmed match values in the MR# register one or more of the following actions can occur (selected by programming the MCR register):

- Reset the counter and prescaler
- Stop the counter
- Generate an interrupt
- Generate an external notification (in this case, on a match the external match pins go to the setting selected via the EMR register).

### 4.1 Timer capture functionality

The timer block contains four capture circuits. The capture functionality allows measuring the time of an external event. Depending on configuration, a rising or a falling edge of the input can cause a capture event. Following an event the capture register is loaded with the Timer Counter value and (if enabled) an interrupt is generated.

The trigger for the capture and whether an interrupt should be generated on match is configured using the CCR register. The captured value is then available in the Capture register (CR#).

### 4.2 Timer interrupt handling

Once the interrupt is generated its status can be accessed and cleared using the IR register. See Section 17–4 and Section 17–4.1 for details of how to set up interrupt generation.

## 5. Register overview

Table 202. Timer register overview (base address: 0xE004 1000 (timer 0), 0xE004 2000 (timer 1), 0xE004 3000 (timer 2), 0xE004 4000 (timer 3), 0xE00C 0000 (MSCSS timer 0, 0xE00C1000 (MSCSS timer 1))

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| TCR | R/W | 0x000 | Timer control register | 0x0 | see Table 17–203 |
| TC | R/W | 0x004 | Timer counter value | 0x0000 0000 | see Table 20–251 |
| PR | R/W | 0x008 | Prescale register | 0x0000 0000 | see Table 20–252 |
| MCR | R/W | 0x00C | Match-control register | 0x000 | see Table 17–206 |
| EMR | R/W | 0x010 | External-match register | 0x000 | see Table 17–207 |
| MR0 | R/W | 0x014 | Match register 0 | 0x0000 0000 | see Table 17–208 |
| MR1 | R/W | 0x018 | Match register 1 | 0x0000 0000 | see Table 17–208 |
| MR2 | R/W | 0x01C | Match register 2 | 0x0000 0000 | see Table 17–208 |
| MR3 | R/W | 0x020 | Match register 3 | 0x0000 0000 | see Table 17–208 |
| CCR | R/W | 0x024 | Capture control register | 0x000 | see Table 17–209 |
| CR0 | R | 0x028 | Capture register 0 | 0x0000 0000 | see Table 17–210 |
| CR1 | R | 0x02C | Capture register 1 | 0x0000 0000 | see Table 17–210 |
| CR2 | R | 0x030 | Capture register 2 | 0x0000 0000 | see Table 17–210 |
| CR3 | R | 0x034 | Capture register 3 | 0x0000 0000 | see Table 17–210 |
| - | R | 0xFD4 | Reserved | 0x0000 00C80 | - |
| INT_CLR_ENABLE | W | 0xFD8 | Interrupt clear-enable register | - | see Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Interrupt set-enable register | - | see Table 10–92 |
| INT_STATUS | R | 0xFE0 | Interrupt status register | 0x0000 0000 | see Table 10–93 |
| INT_ENABLE | R | 0xFE4 | Interrupt enable register | 0x0000 0000 | see Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | Interrupt clear-status register | - | see Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | Interrupt set-status register | - | see Table 10–96 |
| - | R | 0xFFC | Reserved | 0x3012 2400 | - |

### 5.1 Timer control register (TCR)

The TCR is used to control the operation of the timer counter. The counting process starts on CLK_TMRx once the COUNTER_ENABLE bit is set. The process can be reset by setting the COUNTER_RESET bit. The Timer_Counter and Prescale_Counter remain in the reset state as long as the COUNTER_RESET bit is active. The counting process is suspended when the PAUSE_ENABLE bit is set and the pause pin is high.

**Table 203. TCR register description (addresses: 0xE004 1000 (timer 0), 0xE004 2000 (timer 1), 0xE004 3000 (timer 2), 0xE004 4000 (timer 3), 0xE00C 0000 (MSCSS timer 0), 0xE00C 1000 (MSCSS timer 1))**

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved; do not modify, read as logic 0 |
| 2 | PAUSE_ENABLE | R/W | | Enables the pause feature of the timer. If this bit is set the timer and prescale counters will be stopped when a logic HIGH is asserted on timer pin PAUSE [1] |
| | | | 0* | |
| 1 | COUNTER_RESET | R/W | | Reset timer and prescale counter. If this bit is set the counters remain reset until it is cleared again |
| | | | 0* | |
| 0 | COUNTER_ENABLE | R/W | | Enable timer and prescale counter. If this bit is set the counters are running |
| | | | 0* | |

[1] Only for MSCSS Timer 0 and MSCSS Timer 1. For all other timers this bit is reserved: do not modify, read as logic 0.

## 5.2 Timer counter

The timer counter represents the timer-count value, which is incremented every prescale cycle. Depending on the prescale register value and the period of CLK_TMRx, this means that the contents of the register can change very rapidly.

**Table 204. TC register description (addresses: 0xE004 1004 (timer 0), 0xE004 2004 (timer 1), 0xE004 3004 (timer 2), 0xE004 4004 (timer 3), 0xE00C 0004 (MSCSS timer 0), 0xE00C 1004 (MSCSS timer 1)**

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | TC[31:0] | R/W | | Timer counter. It is advisable not to access this register, which may change very rapidly |
| | | | 0000 00 00h* | |

## 5.3 Timer prescale register

The timer prescale register determines the number of clock cycles used as a prescale value for the timer counter clock. When the Prescale_Register value is not equal to zero the internal prescale counter first counts the number of CLK_TMRx cycles as defined in this register plus one, then increments the TC_value.

Updates to the prescale register PR are only possible when the timer and prescale counters are disabled, see bit COUNTER_ENABLE in the TCR register. It is advisable to reset the timer counters once a new prescale value has been programmed. Writes to this register are ignored when the timer counters are enabled (i.e. bit COUNTER_ENABLE in the TCR register is set).

Table 205. PR register bit description (addresses: 0xE004 1008 (timer 0), 0xE004 2008 (timer 1), 0xE004 3008 (timer 2), 0xE004 4008 (timer 3), 0xE00C 0008 (MSCSS timer 0), 0xE00C 1008 (MSCSS timer 1))

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | PR[31:0] | R/W | | Prescale register. This specifies the maximum value for the prescale counter. The timer counter (TC) increments after ‘PR+1’ CLK_TMRx cycles are counted. |
| | | | 0000 00 00h* | |

## 5.4 Timer match-control register

Each MCR can be configured through the match control register to stop both the timer counter and prescale counter. This maintains their value at the time of the match to restart the timer counter at logic 0, and allows the counters to continue counting and/or generate an interrupt when their contents match those of the timer counter. A stop-on-match has higher priority than a reset-on-match.

An interrupt is generated if one of the match registers matches the contents of the timer counter and the interrupt has been enabled through the interrupt-enable control register.

The match control register is used to control what operations are performed when one of the match registers matches the timer counter.

Table 206. MCR register bit description (addresses: 0xE004 100C (timer 0), 0xE004 200C (timer 1), 0xE004 300C (timer 2), 0xE004 400C (timer 3), 0xE00C 000C (MSCSS timer 0), 0xE00C 100C (MSCSS timer 1))

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 | STOP_3 | R/W | 1 | Stop on match MR3 and TC. When logic 1 the timer and prescale counter stop counting if MR3 matches TC |
| | | | 0* | |
| 6 | RESET_3 | R/W | 1 | Reset on match MR3 and TC. When logic 1 the timer counter is reset if MR3 matches TC |
| | | | 0* | |
| 5 | STOP_2 | R/W | 1 | Stop on match MR2 and TC. When logic 1 the timer and prescale counter stop counting if MR2 matches TC |
| | | | 0* | |
| 4 | RESET_2 | R/W | 1 | Reset on match MR2 and TC. When logic 1 the timer counter is reset if MR2 matches TC |
| | | | 0* | |
| 3 | STOP_1 | R/W | 1 | Stop on match MR1 and TC. When logic 1 the timer and prescale counter stop counting if MR1 matches TC |
| | | | 0* | |

**Table 206. MCR register bit description (addresses: 0xE004 100C (timer 0), 0xE004 200C (timer 1), 0xE004 300C (timer 2), 0xE004 400C (timer 3), 0xE00C 000C (MSCSS timer 0), 0xE00C 100C (MSCSS timer 1))** *…continued*

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 2 | RESET_1 | R/W | 1 | Reset on match MR1 and TC. When logic 1 the timer counter is reset if MR1 matches TC |
| | | | 0* | |
| 1 | STOP_0 | R/W | 1 | Stop on match MR0 and TC. When logic 1 the timer and prescale counter stop counting if MR0 matches TC |
| | | | 0* | |
| 0 | RESET_0 | R/W | 1 | Reset on match MR0 and TC. When logic 1 the timer counter is reset if MR0 matches TC |
| | | | 0* | |

## 5.5 Timer external-match register

The EMR provides both control and status of the external match pins. The external match flags and the match outputs can either toggle, go to logic 0, go to logic 1 or maintain state when the contents of the match register are equal to the contents of the timer counter. Note that the match output is set to a specific level on writing the CTRL bits.

**Table 207. EMR register bit description (addresses: 0xE004 1010 (timer 0), 0xE004 2010 (timer 1), 0xE004 3010 (timer 2), 0xE004 4010 (timer 3), 0xE00C 0010 (MSCSS timer 0), 0xE00C 1010 (MSCSS timer 1))**

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 and 10 | CTRL_3[1:0] | R/W | | External match control 3 |
| | | | 00* | Do nothing |
| | | | 01 | Set logic 0 |
| | | | 10 | Set logic 1 |
| | | | 11 | Toggle |
| 9 and 8 | CTRL_2[1:0] | R/W | | External match control 2 |
| | | | 00* | Do nothing |
| | | | 01 | Set logic 0 |
| | | | 10 | Set logic 1 |
| | | | 11 | Toggle |
| 7 and 6 | CTRL_1[1:0] | R/W | | External match control 1 |
| | | | 00* | Do nothing |
| | | | 01 | Set logic 0 |
| | | | 10 | Set logic 1 |
| | | | 11 | Toggle |

**Table 207. EMR register bit description (addresses: 0xE004 1010 (timer 0), 0xE004 2010 (timer 1), 0xE004 3010 (timer 2), 0xE004 4010 (timer 3), 0xE00C 0010 (MSCSS timer 0), 0xE00C 1010 (MSCSS timer 1))** *…continued*

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 5 and 4 | CTRL_0[1:0] | R/W | | External match control 0 |
| | | | 00* | Do nothing |
| | | | 01 | Set logic 0 |
| | | | 10 | Set logic 1 |
| | | | 11 | Toggle |
| 3 | EMR_3 | R | 0 | Current value of the Match 3 pin |
| 2 | EMR_2 | R | 0 | Current value of the Match 2 pin |
| 1 | EMR_1 | R | 0 | Current value of the Match 1 pin |
| 0 | EMR_0 | R | 0 | Current value of the Match 0 pin |

## 5.6 Timer match register

The MR determines the timer-counter match value. Four match registers are available per timer.

**Table 208. MR0-3 register bit description ((addresses: 0xE004 1014-20 (timer 0), 0xE004 2014-20 (timer 1), 0xE004 3014-20 (timer 2), 0xE004 4014-20 (timer 3), 0xE00C 0014-20 (MSCSS timer 0), 0xE00C1014-20 (MSCSS timer 1))**

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | MR[31:0] | R/W | | Match register. This specifies the match value for the timer counter |
| | | | 0000 00 00h* | |

## 5.7 Timer capture-control register

The CCR controls when one of the four possible capture registers is loaded with the value in the timer counter, and whether an interrupt is generated when the capture occurs.

A rising edge is detected if the sequence logic 0 followed by logic 1 occurs: a falling edge is detected if logic 1 followed by logic 0 occurs. The capture control register maintains two bits for each of the counter registers to enable sequence detection for each of the capture registers. If the enabled sequence is detected the timer counter value is loaded into the capture register. If it has been enabled through the interrupt-enable control register an interrupt is then generated. Setting both the rising and falling bits at the same time is a valid configuration.

A reset clears the CCR register.

Table 209. CCR register bit description (addresses: 0xE004 1024 (timer 0), 0xE004 2024 (timer 1), 0xE004 3024 (timer 2), 0xE004 4024 (timer 3), 0xE00C 0024 (MSCSS timer 0), 0xE00C 1024 (MSCSS timer 1))
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 | FALL_3 | R/W | 1 | Capture on capture input 3 falling. When logic 1, a sequence of logic 1 followed by logic 0 from capture input 3 causes CR3 to be loaded with the contents of TC |
| | | | 0* | |
| 6 | RISE_3 | R/W | 1 | Capture on capture input 3 rising. When logic 1, a sequence of logic 0 followed by logic 1 from capture input 3 causes CR3 to be loaded with the contents of TC |
| | | | 0* | |
| 5 | FALL_2 | R/W | 1 | Capture on capture input 2 falling. When logic 1, a sequence of logic 1 followed by logic 0 from capture input 2 causes CR2 to be loaded with the contents of TC |
| | | | 0* | |
| 4 | RISE_2 | R/W | 1 | Capture on capture input 2 rising. When logic 1, a sequence of logic 0 followed by logic 1 from capture input 2 causes CR2 to be loaded with the contents of TC |
| | | | 0* | |
| 3 | FALL_1 | R/W | 1 | Capture on capture input 1 falling. When logic 1, a sequence of logic 1 followed by logic 0 from capture input 1 causes CR1 to be loaded with the contents of TC |
| | | | 0* | |
| 2 | RISE_1 | R/W | 1 | Capture on capture input 1 rising. When logic 1, a sequence of logic 0 followed by logic 1 from capture input 1 causes CR1 to be loaded with the contents of TC |
| | | | 0* | |
| 1 | FALL_0 | R/W | 1 | Capture on capture input 0 falling. When logic 1, a sequence of logic 1 followed by logic 0 from capture input 0 causes CR0 to be loaded with the contents of TC |
| | | | 0* | |
| 0 | RISE_0 | R/W | 1 | Capture on capture input 0 rising. When logic 1, a sequence of logic 0 followed by logic 1 from capture input 0 causes CR0 to be loaded with the contents of TC |
| | | | 0* | |

## 5.8 Timer capture register

The CR is loaded with the timer-counter value when there is an event on the relevant capture input. Four capture registers are available per timer.

**Table 210. CR0-3 register bit description (addresses: 0xE004 1028-34 (timer 0), 0xE004 2028-34 (timer 1), 0xE004 3028-34 (timer 2), 0xE004 4028-34 (timer 3), 0xE00C 0028-34 (MSCSS timer 0), 0xE00C 1028-34 (MSCSS timer 1))**

** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | CR[31:0] | R | | Capture register. This reflects the timer-counter captured value after a capture event |
| | | | 0000 00 00h* | |

## 5.9 Timer interrupt bit description

Table 17–211 gives the interrupts for the timer. The first column gives the bit number i in the interrupt registers. For a general explanation of the interrupt concept and a description of the registers see Section 10–5.

**Table 211. Timer interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 8 | unused | Unused |
| 7 | C3 | Capture 3 event |
| 6 | C2 | Capture 2 event |
| 5 | C1 | Capture 1 event |
| 4 | C0 | Capture 0 event |
| 3 | M3 | Match 3 event |
| 2 | M2 | Match 2 event |
| 1 | M1 | Match 1 event |
| 0 | M0 | Match 0 event |

# UM10316

# Chapter 18: LPC29xx SPI0/1/2

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

The LPC29xx contains three Serial Peripheral Interface (SPI) modules to enable synchronous serial communication with slave or master peripherals that have either Motorola SPI or Texas Instruments synchronous serial interfaces.

The key features are:

- Master or slave operation
- Supports up to four slaves in sequential multi-slave operation
- Programmable clock bit rate and prescale based on SPI source clock (BASE_SPI_CLK), independent of system clock
- Separate transmit and receive FIFO memory buffers; each 16 bits wide by 32 locations deep
- Programmable choice of interface operation: Motorola SPI or Texas Instruments synchronous serial interfaces
- Programmable data-frame size from four to16 bits
- Independent masking of transmit FIFO, receive FIFO and receive-overrun interrupts
- Serial clock rate master mode: fserial_clk $\leq f_{CLK\_SPI}/2$
- Serial clock rate slave mode: fserial_clk = $f_{CLK\_SPI}/4$
- Internal loop-back test mode

### 2.1 SPI functional description

The SPImodule performs serial-to-parallel conversion on data received from a peripheral device. The transmit and receive paths are buffered with FIFO memories (16 bits wide x 32 words deep). Serial data is transmitted on SPI_TxD and received on SPI_RxD.

#### 2.1.1 Modes of operation

The SPI module can operate in:

- Master mode:
  - Normal transmission mode
  - Sequential-slave mode
- Slave mode

Normal transmission mode

In normal transmission mode software intervention is needed every time a new slave needs to be addressed. Also some interrupt handling is required.

In normal transmission mode software programs the settings of the SPI module, writes data to the transmit FIFO and then enables the SPI module. The SPI module transmits until all data has been sent, or until it gets disabled with data still unsent. When data needs to be transmitted to another slave software has to re-program the settings of the SPI module, write new data and enable the SPI module again.

**Remark:** When reprogramming any of its settings the SPI module needs to be disabled first, then enabled again after changing the settings. Transmit data can also be added when the SPI module is still enabled: disabling is not necessary in this case.

Sequential-slave mode

This mode reduces software intervention and interrupt load.

In this mode it is possible to sequentially transmit data to different slaves without having to reprogram the SPI module between transfers. The purpose of this is to minimize interrupts, software intervention and bus traffic. This mode is only applicable when the SPI module is in master mode.

In the example in Figure 18–63 the SPI module supports addressing of four slaves, all of which are sent data in sequential-slave mode. Three elements are transferred to slave 1, two to slave 2, three to slave 3 and finally one to slave 4, after which the SPI module disables itself. When it gets enabled again the same data is transmitted to the four slaves.

Before entering this mode the transmit data needs to be present in the transmit FIFO. No data may be added after entering sequential-slave mode. When the data to be transferred needs to be changed the transmit FIFO needs to be flushed and sequential-slave mode has to be left and entered again to take over the new data present in the transmit FIFO. This is necessary because the FIFO contents are saved as a side-effect of entering sequential-slave mode from normal transmission mode. The data in the transmit FIFO will be saved to allow transmitting it repeatedly without the need to refill the FIFO with the same data.

All programming of the settings necessary to adapt to all slaves has to be done before enabling (starting the transfer) the SPI module in sequential-slave mode. Once a transfer has started these settings cannot be changed until the SPI module has finished the transfer and is automatically disabled again. The use of only one slave in sequential-slave mode is possible.

Once a sequential-slave mode transfer has started it will complete even if the SPI module is disabled before the transfers are over. When a transfer is finished the SPI module disables itself and request a sequential-slave mode ready interrupt.

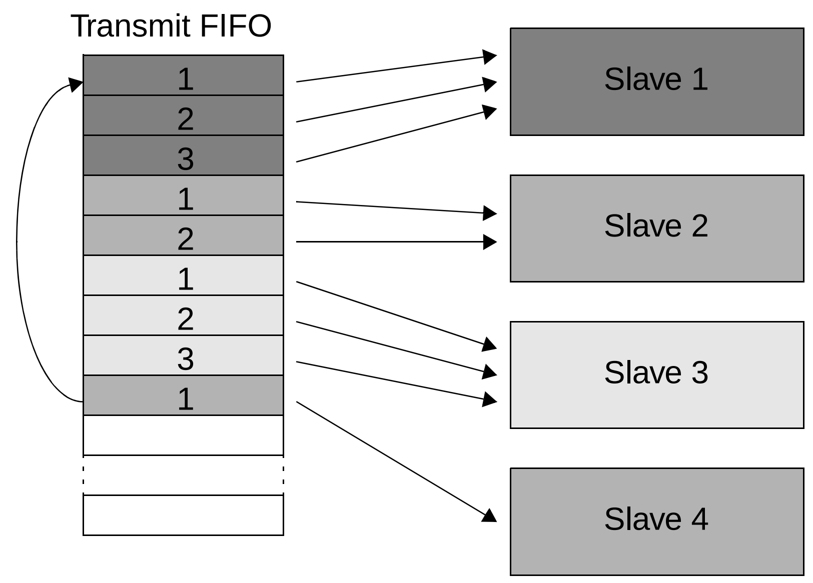


**Fig 63. Sequential-slave mode: example**

It is possible to temporarily suspend or skip one or more of the slaves in a transfer. To do this the data in the transmit FIFO does not need to be flushed: during the transfer it is skipped and nothing happens on the serial interface for the exact time that would have been used by transferring to the skipped slave. In the receive FIFO dummy zero-filled words are written, their number being equal to the number of words that would have been received by the suspended slave. When suspending slaves it is important to keep the corresponding SLVn_SETTINGS. The NUMBER_WORDS field is necessary to skip the data for this slave and the other settings are needed to create the delay of the suspended transfer on the serial interface. Suspending a slave does not change anything in the duration of a sequential-slave transfer.

A slave can also be completely disabled. In this case the transmit FIFO may not hold any data for this slave, which means the transmit FIFO may need to be flushed and reprogrammed. The SLVn_SETTINGS for a disabled slave are ignored.

### 2.1.2 Slave mode

The SPI module can be used in slave mode by setting the MS_MODE bit in the SPI_CONFIG register. The settings of the slave can be programmed in the SLV0_SETTINGS registers that would correspond to slave 0 (offsets 02h4 and 028h). Only slave 0 can be enabled by writing 01h to the SLV_ENABLE register and setting the update_enable bit in the SPI_CONFIG register. A slave can only be programmed to be in normal transmission mode.

## 3. Register overview

Table 212. Register overview: SPI (base address: 0xE004 7000 (SPI0), 0xE004 8000 (SPI1), 0xE004 9000 (SPI2))

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| SPI_CONFIG | R/W | 0x000 | Configuration register | 0x0001 0000 | see Table 18–213 |
| SLV_ENABLE | R/W | 0x004 | Slave-enable register | 0x0000 0000 | see Table 18–214 |
| TX_FIFO_FLUSH | W | 0x008 | Tx FIFO flush register | - | see Table 18–215 |
| FIFO_DATA | R/W | 0x00C | FIFO data register | 0x0000 0000 | see Table 18–216 |
| RX_FIFO_POP | W | 0x010 | Rx FIFO pop register | 0x010 | see Table 18–217 |
| RX_FIFO_READMODE | R/W | 0x014 | Rx FIFO read-mode selection register | 0x0000 0000 | see Table 18–218 |
| DMA_SETTINGS | R/W | 0x018 | DMA settings and enable register | 0x0000 0000 | - |
| STATUS | R | 0x01C | Status register | 0x0000 0005 | see Table 18–220 |
| SLV0_SETTINGS1 | R/W | 0x024 | Slave-settings register 1 for slave 0 | 0x0000 0020 | see Table 18–221 |
| SLV0_SETTINGS2 | R/W | 0x028 | Slave-settings register 2 for slave 0 | 0x0000 0000 | see Table 18–222 |
| SLV1_SETTINGS1 | R/W | 0x02C | Slave-settings register 1 for slave 1 | 0x0000 0020 | see Table 18–221 |
| SLV1_SETTINGS2 | R/W | 0x030 | Slave-settings register 2 for slave 1 | 0x0000 0000 | see Table 18–222 |
| SLV2_SETTINGS1 | R/W | 0x034 | Slave-settings register 1 for slave 2 | 0x0000 0020 | see Table 18–221 |
| SLV2_SETTINGS2 | R/W | 0x038 | Slave-settings register 2 for slave 2 | 0x0000 0000 | see Table 18–222 |
| SLV3_SETTINGS1 | R/W | 0x03C | Slave-settings register 1 for slave 3 | 0x0000 0020 | see Table 18–221 |
| SLV3_SETTINGS2 | R/W | 0x040 | Slave-settings register 2 for slave 3 | 0x0000 0000 | see Table 18–222 |
| INT_THRESHOLD | R/W | 0xFD4 | Tx/Rx FIFO threshold interrupt levels | 0x0000 0000 | see Table 18–223 |
| INT_CLR_ENABLE | W | 0xFD8 | Interrupt clear-enable register | - | see Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Interrupt set-enable register | - | see Table 10–92 |
| INT_STATUS | R | 0xFE0 | Interrupt status register | 0x0000 0000 | see Table 10–93 |
| INT_ENABLE | R | 0xFE4 | interrupt enable register | 0x0000 0000 | see Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | Interrupt clear-status register | - | see Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | Interrupt set-status register | - | see Table 10–96 |
| - | - | 0xFFC | Reserved | 0x3409 3600 | |

### 3.1 SPI configuration register

The SPI configuration register configures SPI operation mode.

Table 213. SPI_CONFIG register bit description (SPI_CONFIG0/1/2, addresses: 0xE004 7000 (SPI0), 0xE004 8000 (SPI1), 0xE004 9000 (SPI2))

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | INTER_SLAVE_DLY | R/W | | The minimum delay between two transfers to different slaves on the serial interface (measured in clock cycles of BASE_SPI_CLK)<br>The minimum value is 1. |
| | | | 0001h* | |
| 15 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

Table 213. SPI_CONFIG register bit description (SPI_CONFIG0/1/2, addresses: 0xE004 7000 (SPI0), 0xE004 8000 (SPI1), 0xE004 9000 (SPI2)) *…continued*
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 7 | UPDATE_ENABLE | R/W | | Update enable bit<br>This must be set by software when the SLV_ENABLE register has been programmed. It will be automatically cleared when the new value is in use.<br>In sequential-slave mode the newly programmed value will be used when the pending sequential-slave transfer finishes.<br>In normal transmission mode the newly programmed value will be used right away (after a clock-domain synchronization delay) |
| | | | 1 | The newly programmed value in the SLV_ENABLE register is not used for transmission yet. As soon as the value is used this bit is cleared automatically. |
| | | | 0* | The current value in the SLV_ENABLE register is used for transmission. A new value may be programmed. As soon as update enable is cleared again the new value will be used for transmission |
| 6 | SOFTWARE_RESET | R/W | | Software reset bit. |
| | | | 1 | Writing 1 to this bit resets the SPI module completely. This bit is self-clearing |
| | | | 0* | |
| 5 | TIMER_TRIGGER | R/W | | Timer trigger-block bit<br>When set the trigger pulses received from a timer (outside the SPI) enable the SPI module; otherwise they are ignored.<br>NOTE: the SPI module can only be enabled by the timer when in sequential-slave mode, otherwise the trigger pulses are ignored. |
| | | | | Timer2 Match Outputs:<br>Tmr2, Match0 --> SP10, trigger in<br>Tmr2, Match1 --> SP11, trigger in<br>Tmr2, Match2 --> SP12, trigger in |
| | | | 1 | Trigger pulses enable SPI module |
| | | | 0* | Trigger pulses are ignored |
| 4 | SLAVE_DISABLE | R/W | | Slave-output disable (only relevant in slave mode)<br>When multiple slaves are connected to a single chip-select signal for broadcasting of a message by a master, only one slave may drive data on its transmit-data line since all slave transmit-data lines are tied together to the single master. |
| | | | 1 | Slave cannot drive its transmit-data output |
| | | | 0* | Slave can drive its transmit-data output |

**Table 213. SPI_CONFIG register bit description (SPI_CONFIG0/1/2, addresses: 0xE004 7000 (SPI0), 0xE004 8000 (SPI1), 0xE004 9000 (SPI2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 3 | TRANSMIT_MODE | R/W | | Transmit mode |
| | | | 1 | Sequential-slave mode |
| | | | 0* | Normal mode |
| 2 | LOOPBACK_MODE | R/W | | Loopback-mode bit<br>Note: when the RX FIFO width is smaller than the TX FIFO width the most significant bits of the transmitted data will be lost in loopback mode. |
| | | | 1 | Transmit data is internally looped-back and received |
| | | | 0* | Normal serial interface operation |
| 1 | MS_MODE | R/W | | Master/slave mode |
| | | | 1 | Slave mode |
| | | | 0* | Master mode |
| 0 | SPI_ENABLE | R/W | | SPI enable bit<br>Slave mode:<br>If the SPI module is not enabled it will not accept data from a master or send data to a master.<br>Master mode:<br>If there is data present in the transmit FIFO the SPI module will start transmitting. This bit will also be set when the SPI module receives a non-blocked enable trigger from the external timer in sequential-slave mode.<br>In sequential-slave mode or when using the external trigger this bit is self-clearing. |
| | | | 1 | SPI enable |
| | | | 0* | SPI disable |

## 3.2 SPI slave-enable register

The slave-enable register controls which slaves are enabled.

**Table 214. SLV_ENABLE register bit description (SLV_ENABLE0/1/2, addresses: 0xE004 7004 (SPI0), 0xE004 8004 (SPI1), 0xE004 9004 (SPI2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 6 and 7 | SLV_ENABLE_3 | R/W | | Slave enable slave 3[1] |
| | | | 00* | The slave is disabled |
| | | | 01 | The slave is enabled |
| | | | 10 | Not supported |
| | | | 11 | The slave is suspended |

**Table 214. SLV_ENABLE register bit description (SLV_ENABLE0/1/2, addresses: 0xE004 7004 (SPI0), 0xE004 8004 (SPI1), 0xE004 9004 (SPI2))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 4 and 5 | SLV_ENABLE_2 | R/W | | Slave enable slave 2[1] |
| | | | 00* | The slave is disabled |
| | | | 01 | The slave is enabled |
| | | | 10 | Not supported |
| | | | 11 | The slave is suspended |
| 3 and 2 | SLV_ENABLE_1 | R/W | | Slave enable slave 1[1] |
| | | | 00* | The slave is disabled |
| | | | 01 | The slave is enabled |
| | | | 10 | Not supported |
| | | | 11 | The slave is suspended |
| 1 and 0 | SLV_ENABLE_0 | R/W | | Slave enable slave 0[1] |
| | | | 00* | The slave is disabled |
| | | | 01 | The slave is enabled |
| | | | 10 | Not supported |
| | | | 11 | The slave is suspended |

[1] In normal transmission mode only one slave may be enabled and the others should be disabled: in sequential-slave mode more than one slave may be enabled. Slaves can also be suspended, which means they will be skipped during the transfer. This is used to avoid sending data to a slave while there is data in the transmit FIFO for that slave, thus skipping data in the transmit FIFO.

## 3.3 SPI transmit-FIFO flush register

The transmit-FIFO flush register forces transmission of the transmit FIFO contents.

**Table 215. TX_FIFO_FLUSH register bit description (TX_FIFO_FLUSH0/1/2: addresses 0xE004 7008 (SPI0), 0xE004 8008 (SPI1), 0xE004 9008 (SPI2))**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | TX_FIFO_FLUSH | W | 1 | Flush transmit FIFO<br>In sequential-slave mode the transmit FIFO keeps its data by default. This means that the FIFO needs to be flushed before changing its contents. |

## 3.4 SPI FIFO data register

The FIFO data register is used to write to the transmit FIFO or read from the receive FIFO.

**Table 216. FIFO_DATA register bit description (FIFO_DATA0/1/2: addresses 0xE004 700C (SPI0), 0xE004 800C (SPI1), 0xE004 900C (SPI2))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | FIFO_DATA | R/W | 0000h* | This register is used to access the FIFOs: Writing data puts new data in the transmit FIFO. Reading data reads a word from the receive FIFO[1]. |

[1] The RX_FIFO_READMODE register can change the effect of reading this register.

## 3.5 SPI receive FIFO POP register

The receive-FIFO POP register is used in RX FIFO PROTECT mode (see Section 18–3.6) to pop the first element from the receive FIFO.

**Table 217. RX_FIFO_POP register bit description (FIFO_POP0/1/2: addresses 0xE004 7010 (SPI0), 0xE004 8010 (SPI1), 0xE004 9010 (SPI2))**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | RX_FIFO_POP | W | 1 | Pops the first element from the receive FIFO. This is necessary in RX FIFO PROTECT mode because reading the FIFO_DATA register will not cause the receive FIFO pointer to be updated. This is to protect the receive FIFO against losing data because of speculative reads. |

## 3.6 SPI receive-FIFO read-mode register

The receive-FIFO read-mode register configures the SPI RX FIFO read mode.

Table 218. RX_FIFO_READMODE register bit description (RX_FIFO_READMODE0/1/2: addresses 0xE004 7014 (SPI0), 0xE004 8014 (SPI1), 0xE004 9014 (SPI2))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | RX_FIFO_PROTECT | R/W | | Receive-FIFO protect-mode bit |
| | | | 1 | Enables the receive-FIFO protect mode to protect the receive-FIFO contents from speculative read actions<br>A read of the FIFO_DATA register will return the data from the FIFO, but will not update the FIFO's read pointer. Speculative reads of the FIFO_DATA register will thus not cause data loss from the receive FIFO. After every read of data the RX FIFO POP register needs to be written to remove the read element from the FIFO and to point to the next element. |
| | | | 0* | Disables receive-FIFO protect mode<br>An explicit pop of the receive FIFO is no longer needed. Reading the FIFO_DATA register will also update the receive FIFO's read pointer. |

## 3.7 SPI DMA settings register

The DMA settings register enables the DMA transfer for the receive and transmit lines and the defines the burst mode.

Table 219. DMA_SETTINGS register bit description (DMA_SETTINGS0/1/2: addresses 0xE004 7018 (SPI0), 0xE004 8018 (SPI1), 0xE004 9018 (SPI2))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7:5 | TX_DMA_BURST | R/W | 000* - 111 | Defines when the SPI will request a Tx burst DMA transfer. The DMA burst will be requested when the transmit FIFO has this number of free spaces (= room to hold one element):<br>000 : 1 free space<br>001 : 4 free spaces<br>010 : 8 free spaces<br>011 : 16 free spaces<br>100 : 32 free spaces<br>101 : 64 free spaces<br>110 : 128 free spaces<br>111 : 256 free spaces |

**Table 219. DMA_SETTINGS register bit description (DMA_SETTINGS0/1/2: addresses 0xE004 7018 (SPI0), 0xE004 8018 (SPI1), 0xE004 9018 (SPI2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 4:2 | RX_DMA_BURST | R/W | 000* - 111 | Defines when the SPI will request a Rx burst DMA transfer. The DMA burst will be requested when the receive FIFO contains this number of received data elements:<br>000 : 1 element<br>001 : 4 elements<br>010 : 8 elements<br>011 : 16 elements<br>100 : 32 elements<br>101 : 64 elements<br>110 : 128 elements<br>111 : 256 elements |
| 1 | TX_DMA_ENABLE | R/W | | Tx DMA enable bit |
| | | | 1 | DMA enabled |
| | | | 0* | DMa disabled |
| 0 | RX_DMA-ENABLE | R/W | | Rx DMA enable bit |
| | | | 1 | DMA enabled |
| | | | 0* | DMA disabled |

## 3.8 SPI status register (Status)

The status register summarizes the status of the SPI module.

**Table 220. SPI status-register bit description (STATUS0/1/2, addresses: 0xE004 701C (SPI0), 0xE004 801C (SPI1), 0xE004 901C (SPI2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 | SMS_MODE_BUSY | R | | Sequential-slave mode busy flag |
| | | | 1 | SPI is currently transmitting in sequential-slave mode. Once all data to all slaves has been sent this bit will be cleared |
| | | | 0* | SPI is not in sequential-slave mode or not busy transmitting in this mode |
| 4 | SPI_BUSY | R | | SPI busy flag |
| | | | 1 | SPI is currently transmitting/receiving or the transmit FIFO is not empty |
| | | | 0* | SPI is idle |
| 3 | RX_FIFO_FULL | R | | Receive FIFO full bit |
| | | | 1 | Receive FIFO full |
| | | | 0* | Receive FIFO not full |
| 2 | RX_FIFO_EMPTY | R | | Receive FIFO empty bit |
| | | | 1* | Receive FIFO empty |
| | | | 0 | Receive FIFO not empty |

**Table 220. SPI status-register bit description (STATUS0/1/2, addresses: 0xE004 701C (SPI0), 0xE004 801C (SPI1), 0xE004 901C (SPI2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 | TX_FIFO_FULL | R | | Transmit FIFO full bit |
| | | | 1 | Transmit FIFO full |
| | | | 0* | Transmit FIFO not full |
| 0 | TX_FIFO_EMPTY | R | | Transmit FIFO empty bit |
| | | | 1* | Transmit FIFO empty |
| | | | 0 | Transmit FIFO not empty |

## 3.9 SPI slave-settings 1 register

The 1st slave-settings register configures the serial clock rate, the number of words and the inter-frame delay for each slave of the SPI module.

**Table 221. SLVn_SETTINGS1 register bit description (SLV0/1/2_SETTINGS1, addresses: 0xE004 7024/2C/34/3C (SPI0), 0xE004 8024/2C/34/3C (SPI1), 0xE004 9024/2C/34/3C (SPI2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | INTER_TRANSFER_DLY | R/W | | The delay between transfers to this slave, measured in serial clock cycles. This delay is a minimum of 0 serial clock cycles[1] |
| | | | 00h* | |
| 23 to 16 | NUMBER_WORDS | R/W | | Number of words to send in sequential-slave mode. After this number of words has been transmitted to the slave the master will start transmitting to the next slave. If sequential-slave mode is disabled this field is not used (minus 1 encoded)[1]. |
| | | | 00h* | |
| 15 to 8 | CLK_DIVISOR2 | R/W | | Serial clock-rate divisor 2[2]: A value from 2 to 254 (lsb bit is hard-coded 0) |
| | | | 02h* | |
| 7 to 0 | CLK_DIVISOR1 | R/W | | Serial clock-rate divisor 1[2]: A value from 0 to 255 |
| | | | 00h* | |

[1] This register is only relevant in master mode, and each individual slave has its own parameters.

[2] The serial-clock frequency is derived from BASE_SPI_CLK (CLK_SPI) using the values programmed in the CLK_DIVISOR1 and CLK_DIVISOR2 fields:

$$fserialclk = \frac{f(CLK\_SPI)}{clkdivisor2 \times (1 + clkdivisor1)}$$

## 3.10 SPI slave-settings 2 register

The SPI second slave-settings register configures several other parameters for each slave of the SPI module.

**Remark:** Some bits in this register are only relevant in master mode, and each individual slave has its own register with parameters.

Table 18–222 shows the bit assignment of the SLVn_SETTINGS2 register.

**Table 222. SLVn_SETTINGS2 register bit description (SLV0/1/2_SETTINGS2, addresses: 0xE004 7028/30/38/40 (SPI0), 0xE004 8028/30/38/40 (SPI1), 0xE004 9028/30/38/40 (SPI2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 17 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 16 to 9 | PRE_POST_CS_DLY | R/W | | Programmable delay that occurs twice in a transfer. This delay is present (i) between assertion of the chip-select and transfer (sampling) of the first data bit AND (ii) between transfer of the last data bit and de-assertion of chip-select. The minimum delay is one SPI serial clock cycle. This register is minus-one encoded (0 gives a one-cycle delay). This field is only relevant in master mode. |
| | | | 0* | |
| 8 | CS_VALUE | R/W | | Chip-select value between back-to-back transfers selection bit. The period in which the chip-select has this value is programmed in the inter_transfer_dly field of the SLVn_SETTINGS1 register This field is only relevant in master mode. |
| | | | 1 | Chip-select has a steady-state HIGH value between transfers |
| | | | 0* | Chip-select has a steady-state LOW value between transfers |
| 7 | TRANSFER_FORMAT | R/W | | Format of transfer |
| | | | 1 | Texas Instruments synchronous serial format |
| | | | 0* | Motorola SPI format |
| 6 | SPO | R/W | | Serial clock polarity (only used if Motorola SPI mode is selected) |
| | | | 1 | The serial clock has a steady-state HIGH value between transfers |
| | | | 0* | The serial clock has a steady-state LOW value between transfers |

**Table 222. SLVn_SETTINGS2 register bit description (SLV0/1/2_SETTINGS2, addresses: 0xE004 7028/30/38/40 (SPI0), 0xE004 8028/30/38/40 (SPI1), 0xE004 9028/30/38/40 (SPI2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 5 | SPH | R/W | | Serial clock phase (only used if Motorola SPI mode is selected). Determines which edges of the serial clock data is captured on during transfers. |
| | | | 1 | First data bit is captured on the second clock-edge transition of a new transfer |
| | | | 0* | First data bit is captured on the first clock-edge transition of a new transfer |
| 4 to 0 | WORDSIZE | R/W | | Word size of transfers to this slave[1] (minus 1 encoded) |
| | | | | Motorola SPI mode: |
| | | | 0 0111h | 8 bits |
| | | | 0 1111h | 16 bits |
| | | | | Texas Instruments synchronous serial mode: |
| | | | 0 0011h | 4 bits |
| | | | 0 0111h | 8 bits |
| | | | 0 1111h | 16 bits |
| | | | 0 0000h* | |

[1] Tx: If WORDSIZE < Tx FIFO width (16 bits) only the LSBs are transmitted. Rx: In case WORDSIZE < Rx FIFO (16 bits) the MSBs of the data stored in the Rx FIFO are zero.

## 3.11 SPI FIFO interrupt threshold register

The interrupt threshold register configures the FIFO levels at which an interrupt request is generated to service the FIFOs.

Table 18–223 shows the bit assignment of the INT_THRESHOLD register.

Table 223. INT_THRESHOLD register bit description (INT_THRESHOLD, addresses: 0xE004 7FD4 (SPI0), 0xE004 8FD4 (SPI1), 0xE004 9FD4 (SPI2))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 8 | TX_THRESHOLD | R/W | | A transmit threshold-level interrupt is requested when the transmit FIFO contains less than this number of elements. When the value is higher than the FIFO size the behavior of the threshold interrupt is undefined. |
| | | | 00h* | |
| 7 to 0 | RX_THRESHOLD | R/W | | A receive threshold-level interrupt is requested when the receive FIFO contains more than this number of elements. When the value is higher than the FIFO size the behavior of the threshold interrupt is undefined. |
| | | | 00h* | |

## 3.12 SPI interrupt bit description

Table 18–224 gives the interrupts for the Serial Peripheral Interface. The first column gives the bit number in the interrupt registers. For an overview of the interrupt registers see Table 18–212. For a general explanation of the interrupt concept and a description of the registers see Section 10–5.

Table 224. SPI interrupt sources

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 5 | unused | Unused |
| 4 | SMS | Sequential-slave mode ready |
| 3 | TX | Transmit threshold level |
| 2 | RX | Receive threshold level |
| 1 | TO | Receive time-out |
| 0 | OV | Receive overrun |

# UM10316

# Chapter 19: LPC29xx Universal Asynchronous Receiver/Transmitter (UART)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. The modem control features are pinned out on the LPC2939/30 only.

## 2. 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.
- Fractional divider for baud rate control.
- Autobaud capabilities and mechanism that enables software flow control implementation.
- RS-485/EIA-485, 9-bit mode support. Direction control provided by RTS pin (LPC2930/39). Use a GPIO pin if RTS not pinned out.
- DMA support.
- Standard modem interface signals included (CTS, DCD, DTS, DTR, RI, RTS) for both UARTs on LPC2930 and LPC2939.

## 3. Pin description

**Table 225. UART0/1 Pin description**

| Pin | Type | Description |
|---|---|---|
| UART0 RXD, UART1 RXD | Input | **Serial Input.** Serial receive data. |
| UART0 TXD, UART1 TXD | Output | **Serial Output.** Serial transmit data. |
| **Modem interface pins (LPC2930 and LPC2939 only)** | | |
| UART0/1 CTS | Input | **Clear To Send.** Active low signal indicates if the external modem is ready to accept transmitted data via TXD from the UART. In normal operation of the modem interface (MCR[4] = 0), the complement value of this signal is stored in UnMSR[4]. State change information is stored in MSR[0] and is a source for a priority level 4 interrupt, if enabled (IER[3] = 1).<br>Only CTS is also used in auto-cts mode to control the UART transmitter.<br>Clear to send. CTS is an asynchronous, active low modem status signal. Its condition can be checked by reading bit 4 (CTS) of the modem status register. Bit 0 (DCTS) of the Modem Status Register (MSR) indicates that CTS1 has changed states since the last read from the MSR. If the modem status interrupt is enabled when CTS1 changes levels and the auto-cts mode is not enabled, an interrupt is generated. CTS is also used in the auto-cts mode to control the transmitter. |

**Table 225. UART0/1 Pin description**

| Pin | Type | Description |
|---|---|---|
| UART0/1 DCD | Input | **Data Carrier Detect.** Active low signal indicates if the external modem has established a communication link with the UART and data may be exchanged. In normal operation of the modem interface (MCR[4]=0), the complement value of this signal is stored in MSR[7]. State change information is stored in MSR3 and is a source for a priority level 4 interrupt, if enabled (IER[3] = 1). |
| UART0/1 DSR | Input | **Data Set Ready.** Active low signal indicates if the external modem is ready to establish a communications link with the UART. In normal operation of the modem interface (MCR[4] = 0), the complement value of this signal is stored in MSR[5]. State change information is stored in MSR[1] and is a source for a priority level 4 interrupt, if enabled (IER[3] = 1). |
| UART0/1 DTR | Output | Data Terminal Ready. Active low signal indicates that the UART is ready to establish connection with external modem. The complement value of this signal is stored in MCR[0].<br>The DTR pin can also be used as an RS-485/EIA-485 output enable signal. |
| UART0/1 RI | Input | **Ring Indicator.** Active low signal indicates that a telephone ringing signal has been detected by the modem. In normal operation of the modem interface (MCR[4] = 0), the complement value of this signal is stored in UnMSR[6]. State change information is stored in MSR[2] and is a source for a priority level 4 interrupt, if enabled (IER[3] = 1). |
| UART0/1 RTS | Output | **Request To Send.** Active low signal indicates that the UART would like to transmit data to the external modem. The complement value of this signal is stored in MCR[1].<br>Only in the auto-rts mode uses RTS to control the transmitter FIFO threshold logic.<br>Request to send. RTS is an active low signal informing the modem or data set that the UART is ready to receive data. RTS is set to the active (low) level by setting the RTS modem control register bit and is set to the inactive (high) level either as a result of a system reset or during loop-back mode operations or by clearing bit 1 (RTS) of the MCR. In the auto-rts mode, RTS is controlled by the transmitter FIFO threshold logic.<br>The RTS pin can also be used as an RS-485/EIA-485 output enable signal. |
| UART0/1 UOUT1/2 | Output | User designated output. OUT1/2 are set to LOW by setting the respective bits in the MCR register. OUT1/2 are set to HIGH on reset or by clearing the OUT1/2 bits in the MCR register. |

## 4. Register overview

Table 226. Register overview: UART0/1 (base address 0xE004 5000 (UART0) and 0xE004 6000 (UART1))

| Name | Access | Address offset | Description | Reset value[1] | Remark |
|---|---|---|---|---|---|
| RBR | RO | 0x00 | Receiver Buffer Register | NA | DLAB=0[2] |
| THR | WO | 0x00 | Transmit Holding Register | NA | DLAB=0[2] |
| DLL | R/W | 0x00 | Divisor Latch LSB | 0x01 | DLAB=1[2] |
| DLM | R/W | 0x04 | Divisor Latch MSB | 0x00 | DLAB=1[2] |
| IER | R/W | 0x04 | Interrupt Enable Register | 0x00 | DLAB=0[2] |
| IIR | RO | 0x08 | Interrupt ID Register | 0x01 | |
| FCR | WO | 0x08 | FIFO Control Register | 0x00 | |
| LCR | R/W | 0x0C | Line Control Register | 0x00 | |
| MCR | R/W | 0x10 | Modem Control Register | 0x00 | |
| LSR | RO | 0x14 | Line Status Register | 0x60 | |
| MSR | RO | 0x18 | Modem Status Register | 0x00 | |
| SCR | R/W | 0x1C | Scratch Pad Register | 0x00 | |
| ACR | R/W | 0x20 | Auto-baud Control Register | 0x00 | |
| FDR | R/W | 0x28 | Fractional Divider Register | 0x10 | |
| TER | R/W | 0x30 | Transmit Enable Register | 0x80 | |
| RS485CTRL | R/W | 0x4C | RS-485 Control | 0x00 | |
| ADRMATCH | R/W | 0x50 | RS-485 address match | 0x00 | |
| RS485DLY | R/W | 0x54 | RS-485/ EIA-485 direction control delay | 0x00 | |

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

[2] The Divisor Latch Access Bit (DLAB) is contained in UnLCR bit 7 and enables access to the Divisor Latches.

### 4.1 UARTn Receiver Buffer Register

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

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

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

**Table 227. UARTn Receiver Buffer Register (U0RBR - address 0xE004 5000, U1RBR - 0xE004 6000 when DLAB = 0, Read Only) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:8 | - | Reserved | NA |
| 7:0 | RBR | The UARTn Receiver Buffer Register contains the oldest received byte in the UARTn Rx FIFO. | Undefined |

## 4.2 UARTn Transmit Holding Register

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

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

**Table 228. UARTn Transmit Holding Register (U0THR - address 0xE004 5000, U1THR - 0xE004 6000 when DLAB = 0, Write Only) bit description**

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

## 4.3 UARTn Divisor Latch Registers

The UARTn Divisor Latch is part of the UARTn Baud Rate Generator and holds the value used to divide the APB clock (BASE_UART_CLK) in order to produce the baud rate clock, which must be 16× the desired baud rate. The UnDLL and UnDLM registers together form a 16 bit divisor where UnDLL contains the lower 8 bits of the divisor and UnDLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed. The Divisor Latch Access Bit (DLAB) in UnLCR must be one in order to access the UARTn Divisor Latches.

**Table 229. UARTn Divisor Latch LSB Register (U0DLL - address 0xE004 5000, U1DLL - 0xE004 6000 when DLAB = 1) bit description**

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

**Table 230. UARTn Divisor Latch MSB Register (U0DLM - address 0xE004 5004, U1DLM - 0xE004 6004 when DLAB = 1) bit description**

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

## 4.4 UARTn Interrupt Enable Register

The UnIER is used to enable the three UARTn interrupt sources.

**Table 231. UARTn Interrupt Enable Register (U0IER - address 0xE004 5004, U1IER - 0xE004 6004 when DLAB = 0) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:10 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 9 | ABTOIntEn | | enables the auto-baud time-out interrupt. | 0 |
| | | 0 | Disable Auto-baud Time-out Interrupt. | |
| | | 1 | Enable Auto-baud Time-out Interrupt. | |
| 8 | ABEOIntEn | | enables the end of auto-baud interrupt. | 0 |
| | | 0 | Disable End of Auto-baud Interrupt. | |
| | | 1 | Enable End of Auto-baud Interrupt. | |
| 7:3 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | RX Line Status Interrupt Enable | | enables the UARTn RX line status interrupts. The status of this interrupt can be read from UnLSR[4:1]. | 0 |
| | | 0 | Disable the RX line status interrupts. | |
| | | 1 | Enable the RX line status interrupts. | |
| 1 | THRE Interrupt Enable | | enables the THRE interrupt for UARTn. The status of this can be read from UnLSR[5]. | 0 |
| | | 0 | Disable the THRE interrupts. | |
| | | 1 | Enable the THRE interrupts. | |
| 0 | RBR Interrupt Enable | | enables the Receive Data Available interrupt for UARTn. It also controls the Character Receive Time-out interrupt. | 0 |
| | | 0 | Disable the RDA interrupts. | |
| | | 1 | Enable the RDA interrupts. | |

## 4.5 UARTn Interrupt Identification Register

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

**Table 232. UARTn Interrupt Identification Register (U0IIR - address 0xE004 5008, U1IIR - 0xE004 6008, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:10 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 9 | ABTOInt | | Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled. | 0 |
| 8 | ABEOInt | | End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled. | 0 |
| 7:6 | FIFO Enable | | These bits are equivalent to UnFCR[0]. | 0 |

Table 232. UARTn Interrupt Identification Register (U0IIR - address 0xE004 5008, U1IIR - 0xE004 6008, Read Only) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 5:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 3:1 | IntId | | Interrupt identification. UnIER[3:1] identifies an interrupt corresponding to the UARTn Rx FIFO. All other combinations of UnIER[3:1] not listed above are reserved (000,100,101,111). | 0 |
| | | 011 | 1 - Receive Line Status (RLS). | |
| | | 010 | 2a - Receive Data Available (RDA). | |
| | | 110 | 2b - Character Time-out Indicator (CTI). | |
| | | 001 | 3 - THRE Interrupt | |
| 0 | IntStatus | | Interrupt status. Note that U1IIR[0] is active low. The pending interrupt can be determined by evaluating UnIIR[3:1]. | 1 |
| | | 0 | At least one interrupt is pending. | |
| | | 1 | No interrupt is pending. | |

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

If the IntStatus bit is 1 no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in Table 19–233. Given the status of UnIIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The UnIIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

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

The UARTn RDA interrupt (UnIIR[3:1] = 010) shares the second level priority with the CTI interrupt (UnIIR[3:1] = 110). The RDA is activated when the UARTn Rx FIFO reaches the trigger level defined in UnFCR[7:6] and is reset when the UARTn Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

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

Table 233. UARTn 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] | UnLSR Read[2] |
| 0100 | Second | RX Data Available | Rx data available or trigger level reached in FIFO (UnFCR0=1) | UnRBR Read[3] or UARTn FIFO drops below trigger level |
| 1100 | Second | Character Time-out indication | Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).<br>The exact time will be:<br>[(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs | UnRBR Read[3] |
| 0010 | Third | THRE | THRE[2] | UnIIR Read (if source of interrupt) or THR write[4] |

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

[2] For details see Section 19–4.9 “UARTn Line Status Register”

[3] For details see Section 19–4.1 “UARTn Receiver Buffer Register”

[4] For details see Section 19–4.5 “UARTn Interrupt Identification Register” and Section 19–4.2 “UARTn Transmit Holding Register”

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

## 4.6 UARTn FIFO Control Register

The UnFCR controls the operation of the UARTn Rx and TX FIFOs.

**Table 234. UARTn FIFO Control Register (U0FCR - address 0xE004 5008, U1FCR - 0xE004 6008, Write Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | - | Reserved | NA |
| 7:6 | RX Trigger Level | | These two bits determine how many receiver UARTn FIFO characters must be written before an interrupt is activated. | 0 |
| | | 00 | Trigger level 0 (1 character or 0x01) | |
| | | 01 | Trigger level 1 (4 characters or 0x04) | |
| | | 10 | Trigger level 2 (8 characters or 0x08) | |
| | | 11 | Trigger level 3 (14 characters or 0x0E) | |
| 5:4 | - | 0 | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 3 | DMA mode | 1 | When in FIFO mode multiple-character transfers are performed until the transmitter FIFO is filled or the receiver FIFO is empty. The receiver direct-memory access becomes active when the receive-FIFO trigger level is reached or a character time-out occurs | 0 |
| | | 0 | Only single-character transfers are done as default in 450 mode | |
| 2 | TX FIFO Reset | 0 | No impact on either of UARTn FIFOs. | 0 |
| | | 1 | Writing a logic 1 to UnFCR[2] will clear all bytes in UARTn TX FIFO and reset the pointer logic. This bit is self-clearing. | |
| 1 | RX FIFO Reset | 0 | No impact on either of UARTn FIFOs. | 0 |
| | | 1 | Writing a logic 1 to UnFCR[1] will clear all bytes in UARTn Rx FIFO and reset the pointer logic. This bit is self-clearing. | |
| 0 | FIFO Enable | 0 | UARTn FIFOs are disabled. Must not be used in the application. | 0 |
| | | 1 | Active high enable for both UARTn Rx and TX FIFOs and UnFCR[7:1] access. **This bit must be set for proper UARTn operation.** Any transition on this bit will automatically clear the UARTn FIFOs. | |

## 4.7 UARTn Line Control Register

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

**Table 235. UARTn Line Control Register (U0LCR - address 0xE004 500C, U1LCR - 0xE004 600C) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | - | Reserved | NA |
| 7 | Divisor Latch Access Bit (DLAB) | 0 | Disable access to Divisor Latches. | 0 |
| | | 1 | Enable access to Divisor Latches. | |

**Table 235. UARTn Line Control Register (U0LCR - address 0xE004 500C, U1LCR - 0xE004 600C) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 6 | Break Control | 0 | Disable break transmission. | 0 |
| | | 1 | Enable break transmission. Output pin UART0 TXD is forced to logic 0 when UnLCR[6] is active high. | |
| 5:4 | Parity Select | 00 | Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd. | 0 |
| | | 01 | Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even. | |
| | | 10 | Forced "1" stick parity. | |
| | | 11 | Forced "0" stick parity. | |
| 3 | Parity Enable | 0 | Disable parity generation and checking. | 0 |
| | | 1 | Enable parity generation and checking. | |
| 2 | Stop Bit Select | 0 | 1 stop bit. | 0 |
| | | 1 | 2 stop bits (1.5 if UnLCR[1:0]=00). | |
| 1:0 | Word Length Select | 00 | 5 bit character length | 0 |
| | | 01 | 6 bit character length | |
| | | 10 | 7 bit character length | |
| | | 11 | 8 bit character length | |

## 4.8 UART0/1 Modem Control Register

The U0/1MCR enables the modem loopback mode and controls the modem output signals.

**Table 236: UART0/1 Modem Control Register (U0MCR - address 0xE004 5010, U1MCR - 0xE004 6010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:8 | - | - | Reserved | NA |
| 7 | CTSen | 0 | Disable auto-cts flow control. | 0 |
| | | 1 | Enable auto-cts flow control. | |
| 6 | RTSen | 0 | Disable auto-rts flow control. | 0 |
| | | 1 | Enable auto-rts flow control. | |
| 5 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |

**Table 236: UART0/1 Modem Control Register (U0MCR - address 0xE004 5010, U1MCR - 0xE004 6010) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 4 | LoopEn | | The modem loopback mode provides a mechanism to perform diagnostic loopback testing. Serial data from the transmitter is connected internally to serial input of the receiver. Input pin, RXD1, has no effect on loopback and output pin, 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 MSR will be driven by the lower four bits of the MCR 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 U1MCR. | 0 |
| | | 0 | Disable modem loopback mode. | |
| | | 1 | Enable modem loopback mode. | |
| 3 | OUT2 | | Inverse control for the UxOUT2 pin. | 0 |
| | | 0 | OUT2 pin is HIGH (inactive). | |
| | | 1 | OUT2 pin is LOW (active). | |
| 2 | OUT1 | | Inverse control for the UxOUT1 pin. | 0 |
| | | 0 | OUT1 pin is HIGH (inactive). | |
| | | 1 | OUT1 pin is LOW (active). | |
| 1 | RTS | | Source for modem output pin RTS. This bit reads as 0 when modem loopback mode is active. | 0 |
| 0 | DTR | | Source for modem output pin, DTR. This bit reads as 0 when modem loopback mode is active. | 0 |

### 4.8.1 Auto-flow control

If auto-RTS mode is enabled the UART0/1‘s receiver FIFO hardware controls the RTS output of the UART. If the auto-CTS mode is enabled the UART0/1‘s TSR hardware will only start transmitting if the CTS input signal is asserted.

#### 19.4.8.2 Auto-RTS

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

When the receiver FIFO level reaches the programmed trigger level, 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 RTS until after it has begun sending the additional byte. RTS is automatically reasserted (to a low value) once the receiver FIFO has reached the previous trigger level. The reassertion of RTS1 signals to the sending UART to continue transmitting data.

If Auto-RTS mode is disabled, the RTSen bit controls the RTS output of the UART. If Auto-RTS mode is enabled, hardware controls the RTS output, and the actual value of 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 has trigger level in FCR set to 0x2 then if Auto-RTS is enabled the UART will deassert the RTS output as soon as the receive FIFO contains 8 bytes. The RTS output will be reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

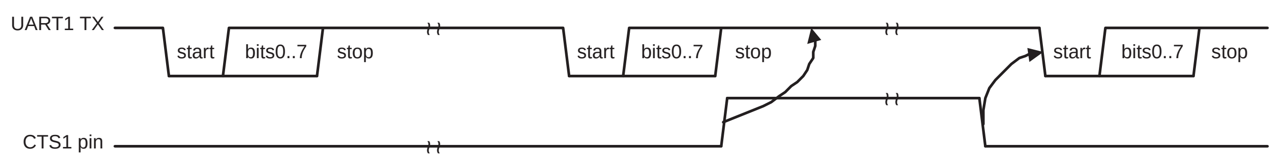


**Fig 64. Auto-RTS Functional Timing**

### 19.4.8.3 Auto-CTS

The Auto-CTS function is enabled by setting the CTSen bit. If Auto-CTS is enabled the transmitter circuitry in the TSR module checks CTS input before sending the next data byte. When CTS is active (low), the transmitter sends the next byte. To stop the transmitter from sending the following byte, 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 CTS signal does not trigger a modem status interrupt unless the CTS Interrupt Enable bit is set, Delta CTS bit in the MSR will be set though. Table 19–237 lists the conditions for generating a Modem Status interrupt.

**Table 237: Modem status interrupt generation**

| Enable Modem Status Interrupt (IER[3]) | CTSen (MCR[7]) | CTS Interrupt Enable (IER[7]) | Delta CTS (MSR[0]) | Delta DCD or Trailing Edge RI or Delta DSR (MSR[3] or MSR[2] or MSR[1]) | Modem Status Interrupt |
|---|---|---|---|---|---|
| 0 | x | x | x | x | No |
| 1 | 0 | x | 0 | 0 | No |
| 1 | 0 | x | 1 | x | Yes |
| 1 | 0 | x | x | 1 | Yes |
| 1 | 1 | 0 | x | 0 | No |
| 1 | 1 | 0 | x | 1 | Yes |
| 1 | 1 | 1 | 0 | 0 | No |
| 1 | 1 | 1 | 1 | x | Yes |
| 1 | 1 | 1 | x | 1 | Yes |

The auto-CTS function reduces interrupts to the host system. When flow control is enabled, a CTS1 state change does not trigger host interrupts because the device automatically controls its own transmitter. Without Auto-CTS, the transmitter sends any data present in the transmit FIFO and a receiver overrun error can result. Figure 19–65 illustrates the Auto-CTS functional timing.

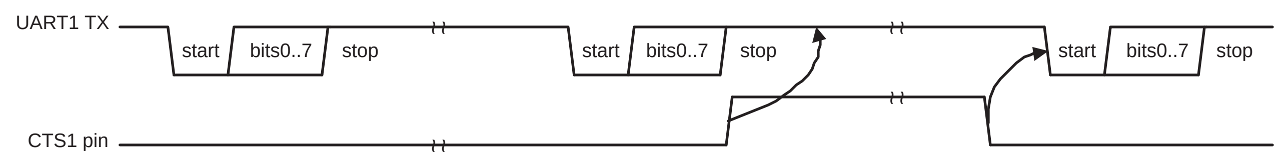


**Fig 65. Auto-CTS Functional Timing**

While starting transmission of the initial character the CTS1 signal is asserted. Transmission will stall as soon as the pending transmission has completed. The UART will continue transmitting a 1 bit as long as CTS1 is deasserted (high). As soon as CTS1 gets deasserted transmission resumes and a start bit is sent followed by the data bits of the next character.

## 4.9 UARTn Line Status Register

The UnLSR is a read-only register that provides status information on the UARTn TX and RX blocks.

**Table 238. UARTn Line Status Register (U0LSR - address 0xE004 5014, U1LSR - 0xE004 6014, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31: 8 | - | - | Reserved | - |
| 7 | Error in RX FIFO (RXFE) | | UnLSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the UnRBR. This bit is cleared when the UnLSR register is read and there are no subsequent errors in the UARTn FIFO. | 0 |
| | | 0 | UnRBR contains no UARTn RX errors or UnFCR[0]=0. | |
| | | 1 | UARTn RBR contains at least one UARTn RX error. | |
| 6 | Transmitter Empty (TEMT) | | TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when either the UnTSR or the UnTHR contain valid data. | 1 |
| | | 0 | UnTHR and/or the UnTSR contains valid data. | |
| | | 1 | UnTHR and the UnTSR are empty. | |
| 5 | Transmitter Holding Register Empty (THRE)) | | THRE is set immediately upon detection of an empty UARTn THR and is cleared on a UnTHR write. | 1 |
| | | 0 | UnTHR contains valid data. | |
| | | 1 | UnTHR is empty. | |
| 4 | Break Interrupt (BI) | | When RXDn is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all 1's). An UnLSR read clears this status bit. The time of break detection is dependent on UnFCR[0]. **Note:** The break interrupt is associated with the character at the top of the UARTn RBR FIFO. | 0 |
| | | 0 | Break interrupt status is inactive. | |
| | | 1 | Break interrupt status is active. | |

**Table 238. UARTn Line Status Register (U0LSR - address 0xE004 5014, U1LSR - 0xE004 6014, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 3 | Framing Error (FE) | | When the stop bit of a received character is a logic 0, a framing error occurs. An UnLSR read clears UnLSR[3]. The time of the framing error detection is dependent on UnFCR0. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error. **Note:** A framing error is associated with the character at the top of the UARTn RBR FIFO. | 0 |
| | | 0 | Framing error status is inactive. | |
| | | 1 | Framing error status is active. | |
| 2 | Parity Error (PE) | | When the parity bit of a received character is in the wrong state, a parity error occurs. An UnLSR read clears UnLSR[2]. Time of parity error detection is dependent on UnFCR[0]. **Note:** A parity error is associated with the character at the top of the UARTn RBR FIFO. | 0 |
| | | 0 | Parity error status is inactive. | |
| | | 1 | Parity error status is active. | |
| 1 | Overrun Error (OE) | | The overrun error condition is set as soon as it occurs. An UnLSR read clears UnLSR1. UnLSR1 is set when UARTn RSR has a new character assembled and the UARTn RBR FIFO is full. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost. | 0 |
| | | 0 | Overrun error status is inactive. | |
| | | 1 | Overrun error status is active. | |
| 0 | Receiver Data Ready (RDR) | | UnLSR0 is set when the UnRBR holds an unread character and is cleared when the UARTn RBR FIFO is empty. | 0 |
| | | 0 | UnRBR is empty. | |
| | | 1 | UnRBR contains valid data. | |

## 4.10 UART0/1 Modem Status Register

The MSR is a read-only register that provides status information on the modem input signals. MSR[3:0] is cleared on MSR read. Note that modem signals have no direct affect on UART1 operation, they facilitate software implementation of modem signal operations.

**Table 239: UARTn Modem Status Register (U0MSR - address 0xE004 5018, U1MSR - 0xE004 6018, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | - | Reserved | NA |
| 7 | DCD | | Data Carrier Detect State. Complement of input DCD. This bit is connected to MCR[3] in modem loopback mode. | 0 |
| 6 | RI | | Ring Indicator State. Complement of input RI. This bit is connected to MCR[2] in modem loopback mode. | 0 |

Table 239: UARTn Modem Status Register (U0MSR - address 0xE004 5018, U1MSR - 0xE004 6018, Read Only) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 5 | DSR | | Data Set Ready State. Complement of input signal DSR. This bit is connected to MCR[0] in modem loopback mode. | 0 |
| 4 | CTS | | Clear To Send State. Complement of input signal CTS. This bit is connected to MCR[1] in modem loopback mode. | 0 |
| 3 | Delta DCD | | Set upon state change of input DCD. Cleared on an MSR read. | 0 |
| | | 0 | No change detected on modem input, DCD. | |
| | | 1 | State change detected on modem input, DCD. | |
| 2 | Trailing Edge RI | | Set upon low to high transition of input RI. Cleared on an MSR read. | 0 |
| | | 0 | No change detected on modem input, RI. | |
| | | 1 | Low-to-high transition detected on RI. | |
| 1 | Delta DSR | | Set upon state change of input DSR. Cleared on an MSR read. | 0 |
| | | 0 | No change detected on modem input, DSR. | |
| | | 1 | State change detected on modem input, DSR. | |
| 0 | Delta CTS | | Set upon state change of input CTS. Cleared on an MSR read. | 0 |
| | | 0 | No change detected on modem input, CTS. | |
| | | 1 | State change detected on modem input, CTS. | |

## 4.11 UARTn Scratch Pad Register

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

Table 240. UARTn Scratch Pad Register (U0SCR - address 0xE004 501C, U1SCR - 0xE004 601C) bit description

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

## 4.12 UARTn Auto-baud Control Register

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

**Table 241. UARTn Auto-baud Control Register (U0ACR - 0xE004 5020, U1ACR - 0xE004 6020) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:10 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |
| 9 | ABTOIntClr | | Auto-baud time-out interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the UnIIR. Writing a 0 has no impact. | 0 |
| 8 | ABEOIntClr | | End of auto-baud interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the UnIIR. Writing a 0 has no impact. | 0 |
| 7:3 | - | NA | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | 0 |
| 2 | AutoRestart | | Restart bit. | 0 |
| | | 0 | No restart. | |
| | | 1 | Restart in case of time-out (counter restarts at next UART0 Rx falling edge) | 0 |
| 1 | Mode | | Auto-baud mode select bit. | 0 |
| | | 0 | Mode 0. | |
| | | 1 | Mode 1. | |
| 0 | Start | | This bit is automatically cleared after auto-baud completion. | 0 |
| | | 0 | Auto-baud stop (auto-baud is not running). | |
| | | 1 | Auto-baud start (auto-baud is running).Auto-baud run bit. This bit is automatically cleared after auto-baud completion. | |

### 4.12.1 Auto-baud

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

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

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

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

The auto-baud function can generate two interrupts.

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

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

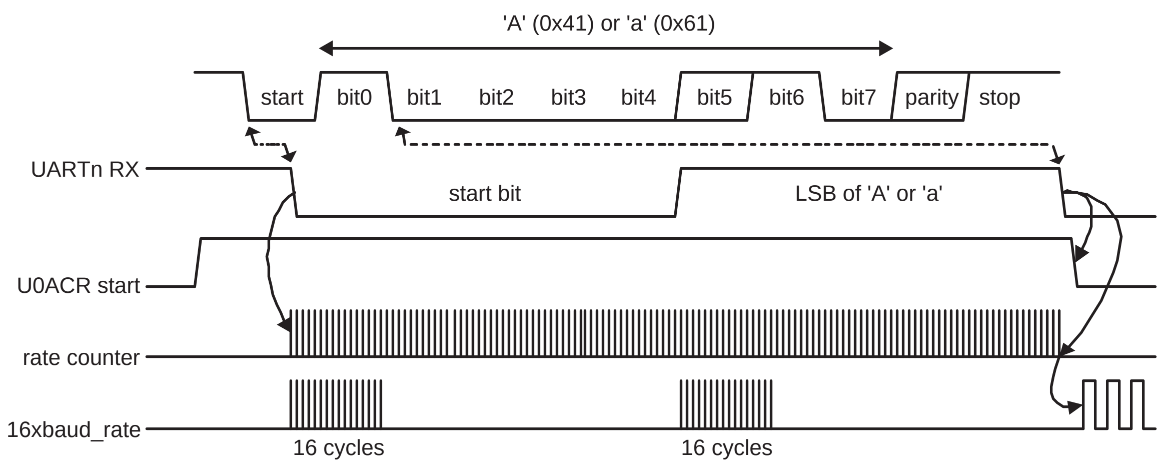
The fractional baud-rate generator must be disabled (DIVADDVAL = 0) during auto-baud. Also, when auto-baud is used, any write to UnDLM and UnDLL registers should be done before UnACR register write. The minimum and the maximum baudrates supported by UARTn are function of BASE_UART_CLK (UARTCLK), number of data bits, stop bits and parity bits.

$$ratemin = \frac{2 \times UARTCLK}{16 \times 2^{15}} \leq UARTn_{baudrate} \leq \frac{UARTCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratem \tag{1}$$

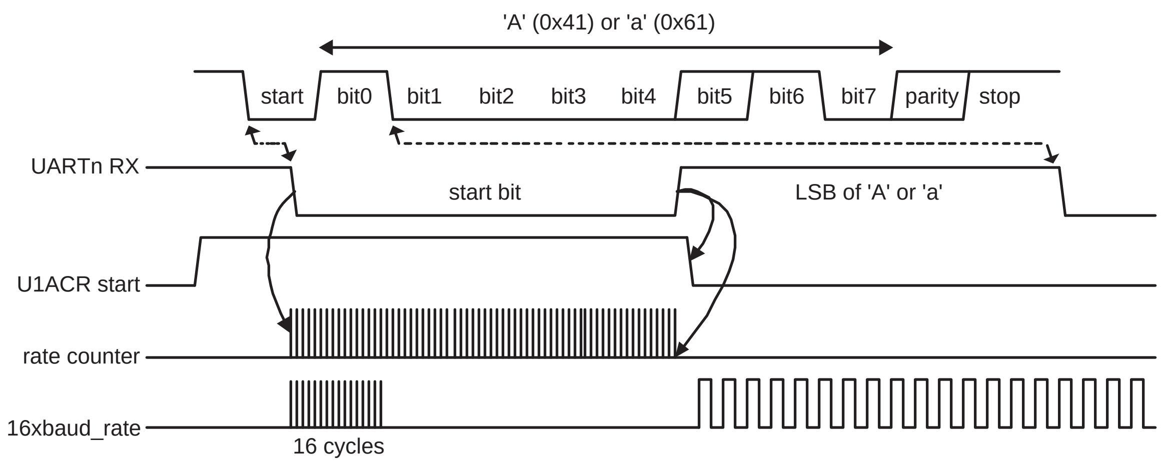
## 4.12.2 Auto-baud modes

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

1. On UnACR Start bit setting, the baud-rate measurement counter is reset and the UARTn UnRSR is reset. The UnRSR baud rate is switch to the highest rate.
2. A falling edge on UARTn Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting BASE_UART_CLK cycles.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the UARTn input clock, guaranteeing the start bit is stored in the UnRSR.
4. During the receipt of the start bit (and the character LSB for mode = 0) the rate counter will continue incrementing with the pre-scaled UARTn input clock (BASE_UART_CLK).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UARTn Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UARTn Rx pin.
6. The rate counter is loaded into UnDLM/UnDLL and the baud-rate will be switched to normal operation. After setting the UnDLM/UnDLL the end of auto-baud interrupt UnIIR ABEOInt will be set, if enabled. The UnRSR will now continue receiving the remaining bits of the ”A/a" character.



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



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

**Fig 66. Autobaud a) mode 0 and b) mode 1 waveform**

## 4.13 UARTn Fractional Divider Register

The UART Fractional Divider Register (U0/1FDR) 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 242. UARTn Fractional Divider Register (U0FDR - address 0xE004 5028, U1FDR - 0xE004 6028) bit description

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

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

The UART baudrate can be calculated as (n = 0/1):

$$UARTn_{baudrate} = \frac{UARTCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)} \quad (2)$$

Where UARTCLK is the peripheral clock (BASE_UART_CLK), U0/1DLM and U0/1DLL are the standard UART0/1 baud rate divider registers, and DIVADDVAL and MULVAL are UART0/1 fractional baudrate generator specific parameters.

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

1. 1 ≤ MULVAL ≤ 15
2. 0 ≤ DIVADDVAL ≤ 14
3. DIVADDVAL<MULVAL

The value of the U0/1FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the U0/1FDR 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.

### 4.13.1 Baudrate calculation

UART can operate with or without using the Fractional Divider. In real-life applications it is likely that the desired baudrate 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 baudrate with a relative error of less than 1.1% from the desired one.

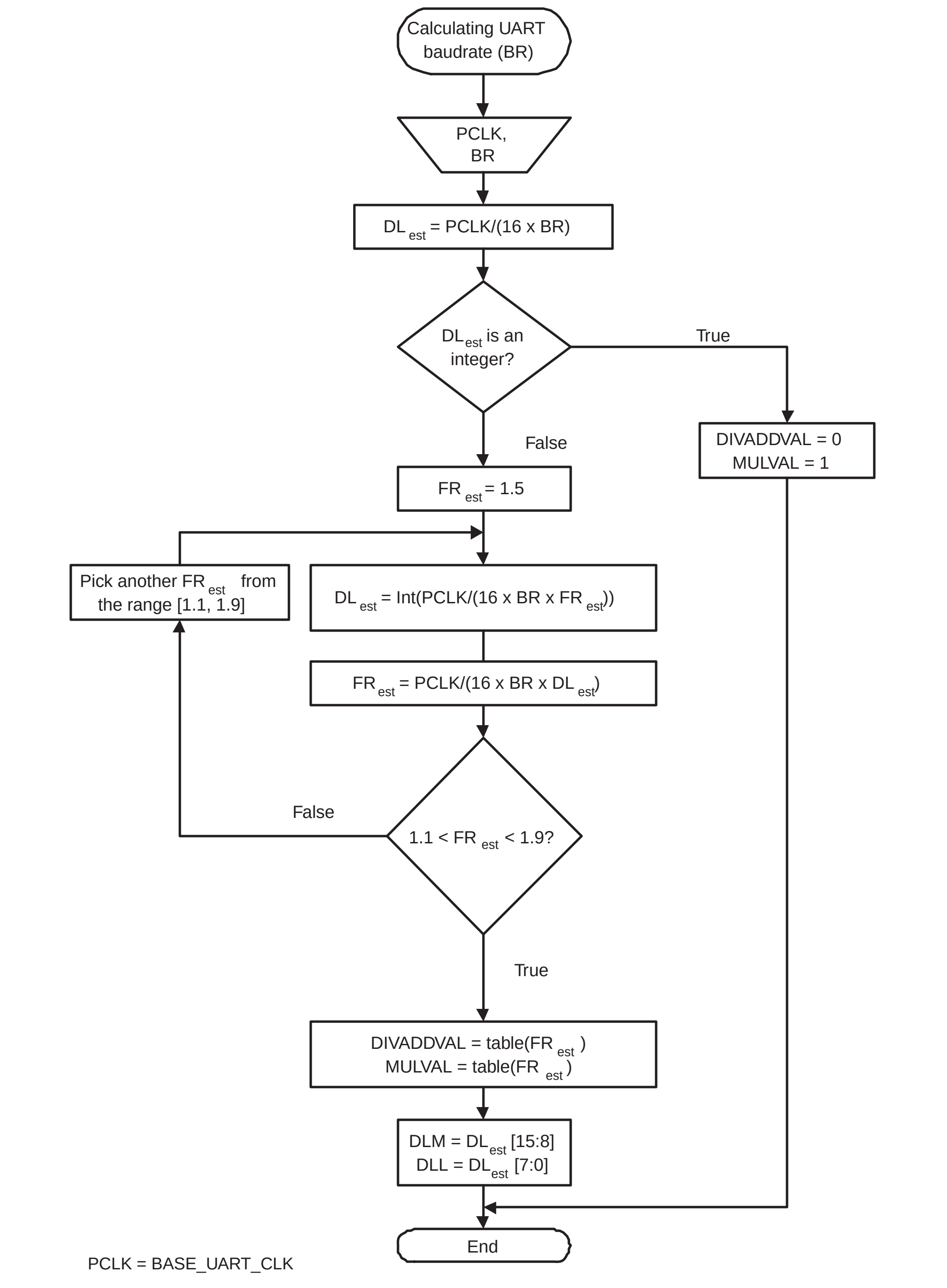

Calculating UART baudrate (BR)
PCLK, BR
$DL_{est}$ = PCLK/(16 x BR)
$DL_{est}$ is an integer?
True
DIVADDVAL = 0
MULVAL = 1
False
$FR_{est}$ = 1.5
Pick another $FR_{est}$ from the range [1.1, 1.9]
$DL_{est}$ = Int(PCLK/(16 x BR x $FR_{est}$))
$FR_{est}$ = PCLK/(16 x BR x $DL_{est}$)
False
1.1 < $FR_{est}$ < 1.9?
True
DIVADDVAL = table($FR_{est}$ )
MULVAL = table($FR_{est}$ )
DLM = $DL_{est}$ [15:8]
DLL = $DL_{est}$ [7:0]
End
PCLK = BASE_UART_CLK

**Fig 67. Algorithm for setting UART dividers**

Table 243. 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 |

#### 4.13.1.1 Example 1: BASE_UART_CLK = 14.7456 MHz, BR = 9600

According to the the provided algorithm $DL_{est}$ = BASE_UART_CLK/(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.

#### 4.13.1.2 Example 2: BASE_UART_CLK = 12 MHz, BR = 115200

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

The closest value for FRest = 1.628 in the look-up Table 19–243 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 19–2 UART's is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

### 4.14 UARTn Transmit Enable Register

The UnTER register enables implementation of software flow control. When TXEn=1, UARTn transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UARTn transmission will stop.

Table 19–244 describes how to use TXEn bit in order to achieve software flow control.

**Table 244. UARTn Transmit Enable Register (U0TER - address 0xE004 5030, U1TER - 0xE004 6030) bit description**

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

## 4.15 UARTn RS485 Control register

The UnRS485CTRL register controls the configuration of the UART 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.

**Table 245. UARTn RS485 Control register(U0RS485CTRL - address 0xE004 504C, U1RS485CTRL - address 0xE004 604C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:3 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | AADEN | 0 | Auto Address Detect (AAD) is disabled. | 0 |
| | | 1 | Auto Address Detect (AAD) is enabled. | |
| 1 | RXDIS | 0 | The receiver is enabled. | 0 |
| | | 1 | The receiver is disabled. | |
| 0 | NMMEN | 0 | RS-485 Normal Multidrop Mode (NMM) is disabled. | 0 |
| | | 1 | RS-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. | |

## 4.16 UARTn RS485 Address Match register

Table 246. UARTn RS485 Address Match register (U0RS485ADRMATCH - address 0xE004 5050, U1RS485ADRMATCH - address 0xE004 6050) bit description

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

## 4.17 UARTn RS-485 Delay value register

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

Table 247. UART1 RS-485 Delay value register (U0RS485DLY - address 0xE004 5054, U1RS485DLY - address 0xE004 6054) bit description

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:8 | - | Reserved | NA |
| 7:0 | DLY | Contains the direction control RTS delay value. This register works in conjunction with an 8-bit counter. | 0x00 |

## 4.18 RS-485 modes of operation

### RS-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 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 partiy bit, and the receiver will be automatically enabled (RS485CTRL bit 1 will be cleared by hardware). The receiver will also generate n Rx Data Ready Interrupt.

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

### RS-485/EIA-485 auto direction control

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

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

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 RTS  with the exception of loopback mode.

**Remark:** On parts without modem pins a GPIO pin can be used to implement direction control.

### 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 . This delay time can be programmed in the 8-bit RS485DLY register. The delay time is in periods of the baud clock. Any delay time from 0 to 255 bit times may be programmed.

### RS485/EIA-485 output inversion

The polarity of the direction control signal on the RTS pins can be reversed by programming bit 5 in the U1RS485CTRL register. When this bit is set, the direction control pin will be driven to logic 1 when the transmitter has data waiting to be sent. The direction control pin will be driven to logic 0 after the last bit of data has been transmitted.

## 5. Architecture

The architecture of the UARTs 0 and 1 is shown below in the block diagram.

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

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

The UARTn transmitter block, UnTX, accepts data written by the CPU or host and buffers the data in the UARTn TX Holding Register FIFO (UnTHR). The UARTn TX Shift Register (UnTSR) reads the data stored in the UnTHR and assembles the data to transmit via the serial output pin, TXDn.

The UARTn Baud Rate Generator block, UnBRG, generates the timing enables used by the UARTn TX block. The UnBRG clock input source is the APB clock (BASE_UART_CLK). The main clock is divided down per the divisor specified in the UnDLL and UnDLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

The interrupt interface contains registers UnIER and UnIIR. The interrupt interface receives several one clock wide enables from the UnTX and UnRX blocks.

Status information from the UnTX and UnRX is stored in the UnLSR. Control information for the UnTX and UnRX is stored in the UnLCR.

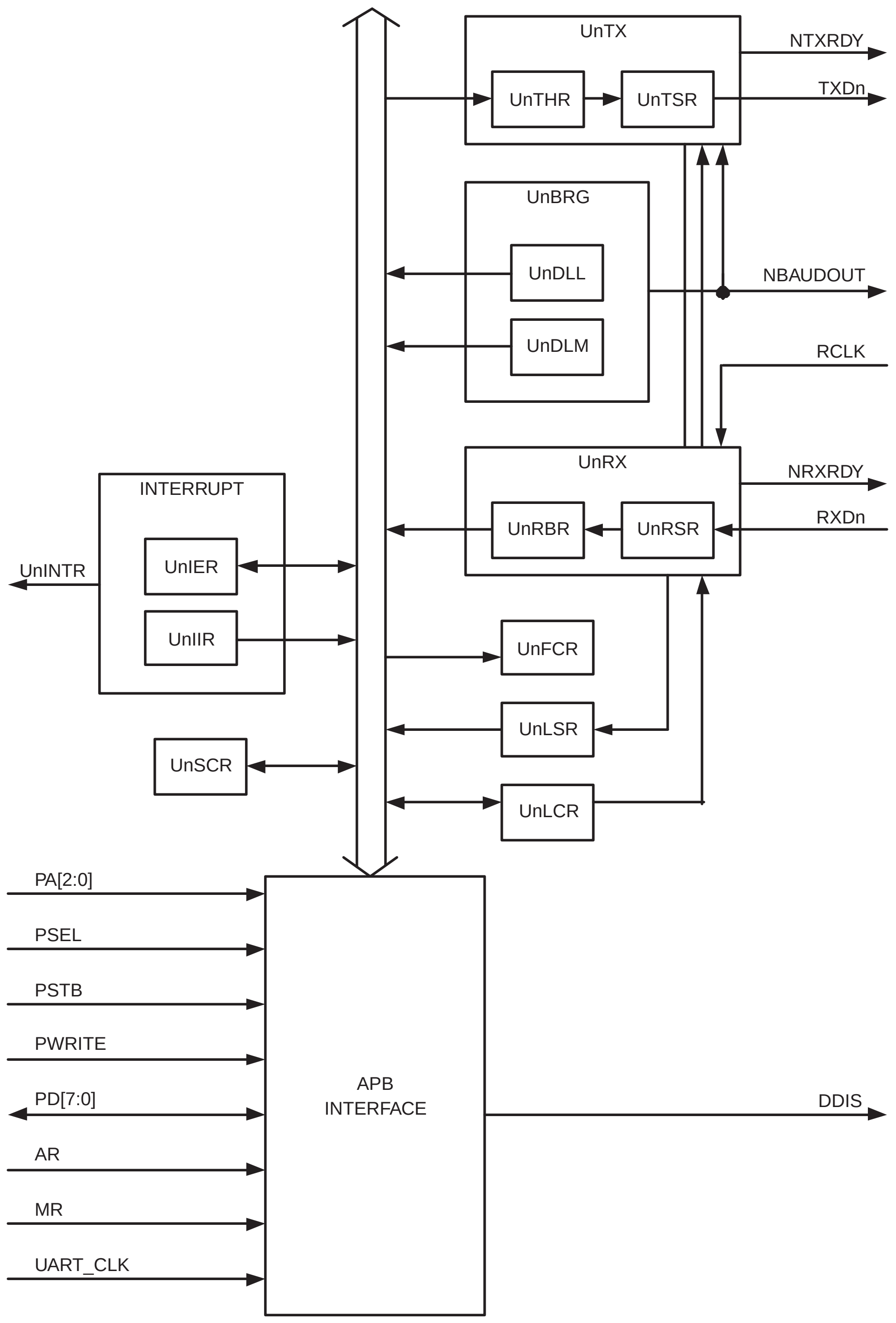


**Fig 68. UART0/1 block diagram**

# UM10316

# Chapter 20: LPC29xx WatchDog Timer (WDT)

Rev. 3 — 19 October 2010 **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

The purpose of the Watchdog timer is to reset the ARM9 processor within a reasonable amount of time if the processor enters an error state. The Watchdog generates a system reset if the user program fails to trigger it correctly within a predetermined amount of time.

The Watchdog is programmed with a time-out value and then periodically restarted. When the Watchdog times out it generates a reset through the RGU.

To generate Watchdog interrupts in Watchdog debug mode the interrupt has to be enabled via the interrupt-enable register. A Watchdog-overflow interrupt can be cleared by writing to the clear-interrupt register.

Another way to prevent resets during debug mode is via the pause feature of the Watchdog timer. The Watchdog is stalled when the ARM9 is in debug mode and the PAUSE_ENABLE bit in the Watchdog timer control register is set.

The Watchdog reset output is fed to the Reset Generator Unit (RGU). The RGU contains a reset-source register to identify the source when the device has gone through a reset. See Section 4–3.

A Watchdog reset is equal to an external reset: the program counter will start from 0x0000 0000 and registers are cleared. The Reset Generation Unit contains a reset source register to determine the reset source when the device has gone through a reset. See Section 4–3.

## 3. Watchdog programming example

The Watchdog should be set up for normal or debug mode as follows:

**Table 248. Watchdog programming steps**

| Step | Normal mode | Debug mode |
|---|---|---|
| 1 | Read from Watchdog key register (0x038). Returns value (0x251D8950). | Read from Watchdog key register (0x038). Returns value (0x251D8950). |
| 2 | Write 0x251D8950 (key) to Watchdog timeout register (0x03C).<br>It is now unlocked. | Write 0x251D8951 (key xor wd_rst_dis) to Watchdog debug register (0x040).<br>Reset generation is now disabled. |
| 3 | Write time-out value (e.g.0x0000FFFF) to Watchdog timeout register .<br>This indicates time-out reset at 65,536 clock cycles. It is now locked again | Write 0x251D8950 (key) to Watchdog timeout register (0x03C).<br>It is now unlocked. |

Table 248. Watchdog programming steps

| Step | Normal mode | Debug mode |
|---|---|---|
| 4 | Write 0x251D8951 (key exor counter_enable) to the Watchdog Timer Control register. The timer is now started | Write time-out value (e.g.0x0000 FFFF) to the Watchdog time-out register. This indicates time-out reset at 65,536 clock cycles. It is now locked again. |
| 5 | Write 0x251D8950 (key) to the Watchdog key register (0x038) at periodical intervals to restart Timer_Counter. Write before time-out occurs! | Write 0x251D8951 (key exor counter_enable) to the Watchdog timer control register. The timer is now started |
| 6 | - | Write 0x251D8950 (key) to the Watchdog key register (0x038) at periodical intervals to restart Timer_Counter. Write before time-out occurs! |

## 4. Register overview

Table 249. Register overview: WDT (base address 0xE004 0000)

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| WTCR | R/W | 0x000 | Timer control register | 0x0 | see Table 20–250 |
| TC | R/W | 0x004 | Timer counter value | 0x0000 0000 | see Table 20–251 |
| PR | R/W | 0x008 | Prescale register | 0x0000 0000 | see Table 20–252 |
| WD_KEY | R/W | 0x038 | Watchdog key register | 0x251D 8950 | see Table 20–253 |
| WD_TIMEOUT | R/W | 0x03C | Watchdog time-out register | 0xFFFF FFFF | see Table 20–254 |
| WD_DEBUG | R/W | 0x040 | Watchdog debug register | 0x0000 0000 | see Table 20–255 |
| - | R | 0xFD4 | Reserved | 0x0000 00C8 | |
| INT_CLR_ENABLE | W | 0xFD8 | Interrupt clear-enable register | 0x0000 0101 | see Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Interrupt set-enable register | - | see Table 10–92 |
| INT_STATUS | R | 0xFE0 | Interrupt status register | 0x0000 0000 | see Table 10–93 |
| INT_ENABLE | R | 0xFE4 | interrupt enable register | 0x0000 0000 | see Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | Interrupt clear-status register | - | see Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | Interrupt set-status register | - | see Table 10–96 |
| - | R | 0xFFC | Reserved | 0x3012 2900 | |

### 4.1 Watchdog timer-control register

The WTCR is used to control the operation of the timer counter. The Watchdog key - as stored in the Watchdog Key register - is used to prevent unintentional control. This key must be XOR-ed with the two control bits so that it is only possible to start the timer by writing 0x251D 8951. All other values are ignored. Resetting the timer (e.g. just before entering power-down mode) is only possible by writing 0x251D 8952. The counting process starts on CLK_SAFE once the COUNTER_ENABLE bit is set. The process can be reset by setting the COUNTER_RESET bit. The TC and TR remain in the reset state for as long as the COUNTER_RESET bit is active.

**Table 250. WTCR register bit description (WTCR, address: 0xE004 0000)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | WD_KEY | R/W | 0x0000 0000* | Protection key, see above. Writes to the WTCR register are ignored if a value other than the Watchdog key is written to this field, read as logic 0 |
| 2 | PAUSE_ENABLE | R/W | 1 | Enables the pause feature of the Watchdog timer. If this bit is set the counters (timer and prescale counter) will be stopped when the ARM processor is in debug mode (connected to ARM9_DBGACK) |
| | | | 0* | |
| 1 | COUNTER_RESET | R/W | 1 | Reset timer and prescale counter. If this bit is set the counters remain reset until it is cleared again |
| | | | 0* | |
| 0 | COUNTER_ENABLE | R/W | 1 | Enable timer and prescale counter. If this bit is set the counters are running |
| | | | 0* | |

## 4.2 Watchdog timer counter

The TC represents the timer-count value which is incremented every prescale cycle. Depending on the prescale register value and the period of CLK_SAFE the contents of this register can change very rapidly.

Writes to the timer counter register are disabled. Furthermore the timer counter is reset when the Watchdog keyword is written to the WD_KEY register. The timer counter stops counting on Watchdog_Time_Out match.

**Table 251. TC register bit description (TC, address: 0xE004 0004)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | TC[31:0] | R | 0x0000 0000* | Watchdog timer counter. It is advisable not to access this register, which may change very rapidly |

## 4.3 Watchdog prescale register

The prescale register determines the number of clock cycles as a prescale value for the Watchdog timer counter. When the value is not equal to zero the internal prescale counter first counts the number of CLK_SAFE cycles as defined in this register plus one, then increments the TC_value.

Updates to the prescale register are only possible when the timer and prescale counters are disabled, see bit COUNTER_ENABLE in the TCR register. It is advisable to reset the timer counters once a new prescale value has been programmed. Writes to this register are ignored when the timer counters are enabled (bit COUNTER_ENABLE in the TCR register is set).

**Table 252. PR register bit descritpion (PR, address: 0xE004 0008)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | PR[31:0] | R/W | 0x0000 0000* | Prescale register. This specifies the maximum value for the prescale counter. The TC increments after ‘PR+1’ CLK_SAFE cycles have been counted |

## 4.4 Watchdog timer key register

The Watchdog timer key register contains a protection code to be used when accessing the other Watchdog timer registers to prevent accidental alteration of these registers. The value is hard-wired and can only be read, not modified. Writing the key value to this register restarts the Timer_Counter, but writing other values has no effect. The Watchdog timer must be periodically triggered by correct writes to this register in order to prevent generation of a system reset .

**Table 253. WD_KEY register bit description (WD_KEY, address: 0xE004 0038)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | WD_KEY_VAL | R/W | 0x251D 8950* | Key value to be used when accessing Watchdog-timer control register |

## 4.5 Watchdog time-out register

The Watchdog time-out register holds the time-out value for Watchdog reset generation. Timer_Counter counts up to this value and then asserts the Watchdog reset. To prevent this from happening the user must write the key word to the Watchdog_Key register before Timer_Counter reaches the programmed value. To be able to write to this register it must be unlocked first. This is done by first writing to this register the key word as stored in the Watchdog_Key register. Updating the Watchdog_Time_Out register by unlocking and writing is also possible when the Watchdog timer has already been enabled (i.e. the COUNTER_ENABLE bit in the WTCR register is set).

**Table 254. WD_TIMEOUT register bit description (WD_TIMEOUT, address: 0xE004 003C)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | WD_TIMEOUT_VAL | R/W | 0x00FF FFFF* | When the TC matches this value the Watchdog reset will be asserted |

## 4.6 Watchdog debug register

To debug the Watchdog functionality, generation of a system reset when the Watchdog timer counter reaches the Wd_Time_Out value must be prevented. When it is enabled an interrupt can be generated instead. Reset generation on time-out can be blocked by writing a 1 to the Watchdog reset-disable bit Wd_Rst_Dis.
This is only possible when the upper 31 bits of the data written to the Watchdog_Debug register are identical to the Watchdog_Key. The Wd_Rst_Dis bit must be XOR-ed with the Watchdog key. In all other cases writes to this register are ignored.

**Table 255. WD_DEBUG register bit description (WD_DEBUG, address: 0xE004 0040)**
** = reset value*

| Bit | Variable name | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | WD_KEY | R/W | 0x0000 0000* | Protection key, see above. Writes to the WD_DEBUG register are ignored if a value other than the Watchdog key value (WD_KEY_VAL) 0x251D 8950* is written to this field, read as logic 0. |
| 0 | WD_RST_DIS | R/W | 1 | Disables generation of a reset on Watchdog time-out. This feature is used for debug purposes only. |
| | | | 0* | |

## 4.7 Watchdog interrupt bit description

Table 20–256 gives the interrupts for the Watchdog subsystem. The first column gives the bit number in the interrupt registers. For a general explanation of the interrupt concept and a description of the registers see Section 10–5.

**Table 256. Watchdog interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 9 | unused | Unused |
| 8 | WD | Watchdog timer |
| 7 to 0 | unused | Unused |

# UM10316

## Chapter 21: LPC29xx CAN 0/1

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. CAN functional description

Figure 21–69 gives a brief overview of the main blocks in the CAN gateway controller. This consists of two identical CAN controllers working as independent CAN nodes. Incoming CAN messages can be filtered by the acceptance filter before they reach the CAN controller. The acceptance filter fetches information on which message should be filtered from the ID look-up table. The status of all CAN controllers is summarized in the central CAN status registers.

The CAN controller block, acceptance filter block and ID look-up table RAM are described in detail in the following sections.

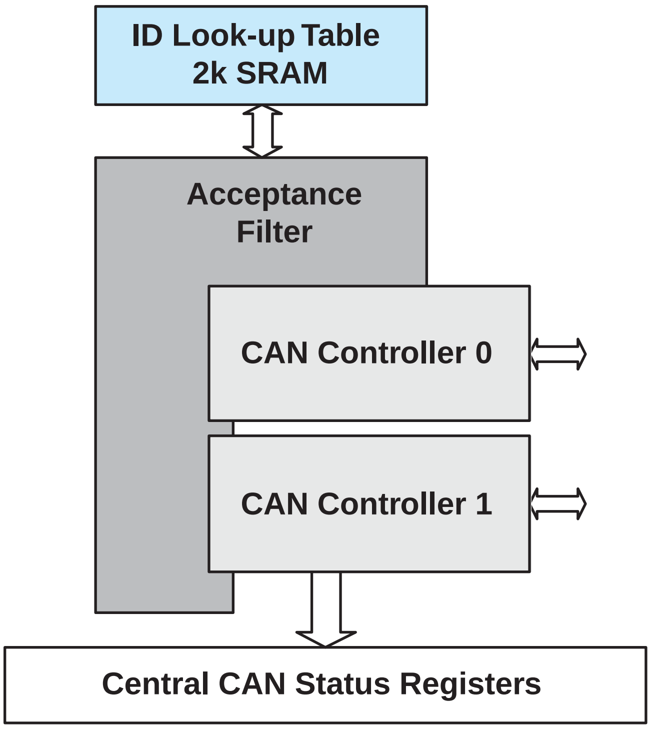


**Fig 69. CAN gateway controller block diagram**

## 3. CAN controller

The CAN controller is a complete serial interface with transmit and receive buffers but without an acceptance filter. Identifier filtering is done for all CAN channels in a separate block, see also Section 21–10.

## 4. CAN bus timing

### 4.1 Baud-rate prescaler

The period of the CAN system clock, $t_{scl}$ , is programmable and determines individual bit timing. The CAN system clock is calculated using the following equation:

$$t_{scl} = \frac{BRP + 1}{fclk(sys)}$$

BRP is the baud-rate prescaler value defined in the bus timing register CCBT.

### 4.2 Synchronization jump width

To compensate for phase shifts between the clock oscillators of different bus controllers, any bus controller must resynchronize on any relevant signal edge of the current transmission. The synchronization jump-width defines the maximum number of clock cycles by which a certain bit period can be shortened or lengthened during one resynchronization:

$$tsjw = tscl(SJW + 1)$$

SJW is the synchronization jump-width value defined in the bus timing register CCBT.

### 4.3 Time segments 1 and 2

Time segments TSEG1 and TSEG2 determine the number of clock cycles per bit-period and the location of the sampling point:

$$tSYNCSEG = 1tscl$$

$$tTSEG1 = tscl(TSEG1 + 1)$$

$$tTSEG2 = tscl(TSEG2 + 1)$$

TSEG1 and TSEG2 are timing-segment 1 and 2 values defined in CCBT. For determination of bit-timing parameters see also Ref. 31–5.

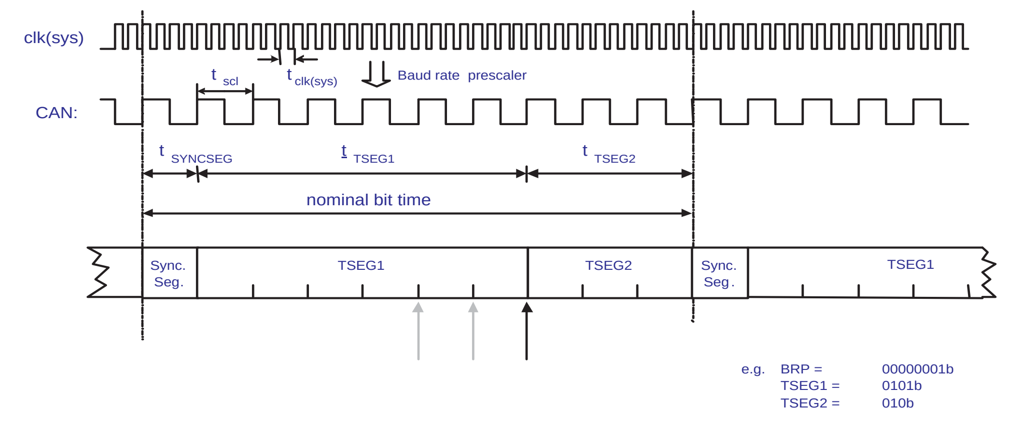


**Fig 70. General structure of a bit-period**

## 5. CAN transmit buffers

The CAN controller contains three transmit buffers. Each of these has a length of four 32-bit words and can store one complete CAN message.

The transmit buffer-status bits TBS3, TBS2, TBS1 in the CAN controller status register CCSTAT signal which of the three transmit buffers is available and ready to be filled with data for the next transmit messages.

### 5.1 Transmit buffer layout

The transmit buffers are located in the address range from CANC Base 030h to 05Ch. The buffer layout is subdivided into message-information, identifier and data registers.

The message info register includes the Tx frame info describing frame format, data length and whether it is a remote or a data frame. In addition, a Tx priority field allows definition of a priority for each transmit buffer (see Section 21–5.2 for more details).

The identifier register contains the message ID. Depending on the chosen frame format, an 11-bit identifier for standard frame format (SFF) or a 29-bit identifier for extended frame format (EFF) then follows.

**Remark:** Unused bits in the ID field have to be defined as 0.

Data registers A and B contain the message data bytes.

The number of data fields used in a message is coded with the data-length code DLC in the message info register. At the start of a remote frame transmission the DLC is not considered because the RTR bit is 1 (= remote).

This forces the number of transmitted/received data bytes to be 0. The DLC must be specified correctly to avoid bus errors, which can occur if two CAN controllers simultaneously start a remote frame transmission with the same identifier. For reasons of compatibility **no** DLC greater than eight should be used. If a value greater than eight is selected, eight bytes are transmitted in the data frame with the DLC specified in the message info register.

### 5.2 Automatic transmit-priority protection

To allow uninterrupted streams of transmit messages, the CAN controller provides automatic transmit-priority detection for all transmit buffers. Depending on the selected transmit priority mode (TPM) in the mode register, internal prioritization is based on the CAN identifier or a user-defined local priority.

If more than one message is enabled for transmission (TR=1 or SRR=1) the internal transmit-message queue is organized so that the transmit buffer with the lowest CAN identifier (ID) or the lowest local priority (TXPRIO) is sent first. The result of this internal scheduling process is taken into account before a new CAN message is sent onto the bus. This is also true for a retransmission caused by a transmission error or lost arbitration.

In cases where the same transmit priority or the same ID is chosen for more than one transmit buffer, the buffer with the lowest number is sent first.

## 6. CAN receive buffer

The CAN Controller has double receive-buffer architecture which allows the CPU to read a received message while the next message is being received and stored in the remaining buffer.

The CAN controller generates a data-overrun condition when both receive buffers are full of messages and have not been released before a new message arrives and passes through the acceptance filter. The data-overrun situation is signaled via the DOS bit in the global status register CCGS and by the data-overrun interrupt DOI (if enabled).

As soon as a received message is read from the receive buffer, the buffer should be released by setting the release-receive buffer bit RRB in the CAN controller mode register CCCMD.

### 6.1 Receive buffer layout

The receive message buffer layout is similar to the transmit message buffer described above. The identifier, frame format, remote-transmission request bit and data-length code have the same meanings as those already described.The only differences are the identifier index IDI and the bypass-mode bit BP in the message info register CCRXBMI.

The identifier index IDI is a 10-bit field in the message info register. It contains the table position (index number) of the ID look-up table for an accepted and received CAN message (see Section 21–3 for more details). Software can use this index number to simplify message transfers from the receive buffer into the standard CPU RAM. The bypass-mode bit BP is a status bit which signals whether or not a current CAN message was received in acceptance-filter bypass mode. The acceptance filter can be put into bypass mode by setting the ACCBP bit in the acceptance-filter mode register CAMODE.

The received data-length code in the message info register represents the received data length.

**Remark:** The CAN protocol specification Ref. 31–5 allows transmission of eight data bytes in conjunction with a data-length code larger than eight. In this case the DLC will not match the number of data bytes. This should be borne in mind when software uses the received DLC information from the message info register CCRXBMI.

## 7. CAN controller self-test

The CAN controller supports two options for self-tests:

- Global self-test: setting the self-reception request bit in normal operating mode
- Local self-test: setting the self-reception request bit in self-test mode

Both self-tests use the self-reception feature of the CAN controller. Along with the self-reception request the transmitted message is also received and stored in the receive buffer, so the acceptance filter must be configured accordingly. As soon as the CAN message is transmitted a transmit and a receive interrupt are generated (if enabled).

### 7.1 Global self-test

A global self-test can (for example) verify the used configuration in a given CAN system. As shown in Figure 21–71, at least one other CAN node which acknowledges each CAN message has to be connected to the CAN bus.

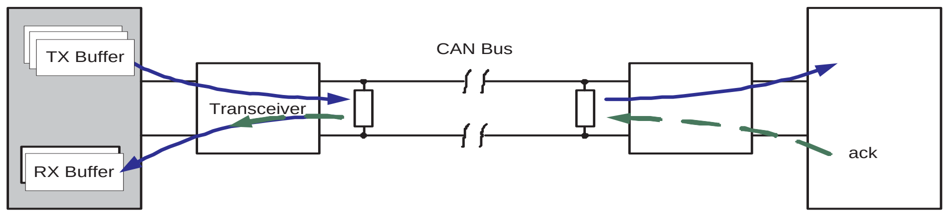


**Fig 71. Global self-test (example high-speed CAN bus)**

Initiating a global self-test is similar to a normal CAN transmission. Transmission of a CAN message is initiated by setting the self-reception request bit SRR in conjunction with the selected message-buffer bits STB3, STB2 and STB1 in the CAN controller command register CCCMD.

### 7.2 Local self-test

Local self-test can be used for single-node tests. In this case an acknowledge from other nodes is not needed. As shown in Figure 21–72, a CAN transceiver with an appropriate CAN bus termination has to be connected.

The CAN controller must be put into self-test mode by setting the STM bit in the CAN controller mode register CCMODE. Setting the STM bit is only possible when the CAN controller is in reset mode.

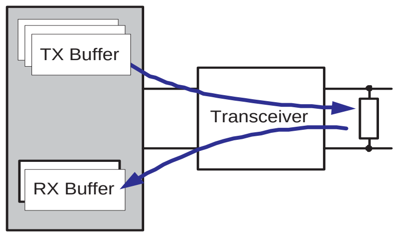


Fig 72. Local self-test (example for high-speed CAN bus)

A message transmission is initiated by setting the self-reception request bit SRR in conjunction with the selected message buffer(s) STB3, STB2 and STB1.

## 8. CAN global acceptance filter

The global acceptance filter provides a look-up for received identifiers - called acceptance filtering in CAN terminology - for all the CAN controllers. It includes a CAN ID look-up table memory in which software maintains one to five sections of identifiers. The CAN ID look-up table memory is 2 kB (512 words, each of 32 bits). It can contain up to 1024 standard frame identifiers (SFFs) or 512 extended frame identifiers (EFFs) or a mixture of both types. Note that the whole CAN ID look-up table memory is only word-accessible. The CAN ID look-up table memory is structured into up to five sections, each of which lists the identifiers of a certain CAN message type (see Table 21–257).

Table 257. CAN ID look-up table memory sections

| Name of Section | Reception method | CAN message frame format | Explicit IDs or group of IDs |
|---|---|---|---|
| Standard Frame Format FullCAN identifier section | stored directly in memory | Standard Frame Format (SFF) | Explicit |
| Standard Frame Format explicit identifier section | buffered | Standard Frame Format (SFF) | Explicit |
| Standard Frame Format group identifier section | buffered | Standard Frame Format (SFF) | Group |
| Extended Frame Format explicit identifier section | buffered | Extended Frame Format (EFF) | Explicit |
| Extended Frame Format group identifier section | buffered | Extended Frame Format (EFF) | Group |

Five start -address registers exist to indicate the boundaries of the different sections within the ID look-up table memory. These registers store the offset for the base address CANAFM (see Section 2–2). The standard frame-format FullCAN identifier section always starts at the offset 00h, with the following sections starting as defined in the start-address registers. The look-up table ends with the FullCAN message object section, which starts at the offset CAEOTA. A non-existent section is indicated by equal values in consecutive start-address registers.

See Figure 21–73 for the structure of the CAN ID look-up table memory sections.

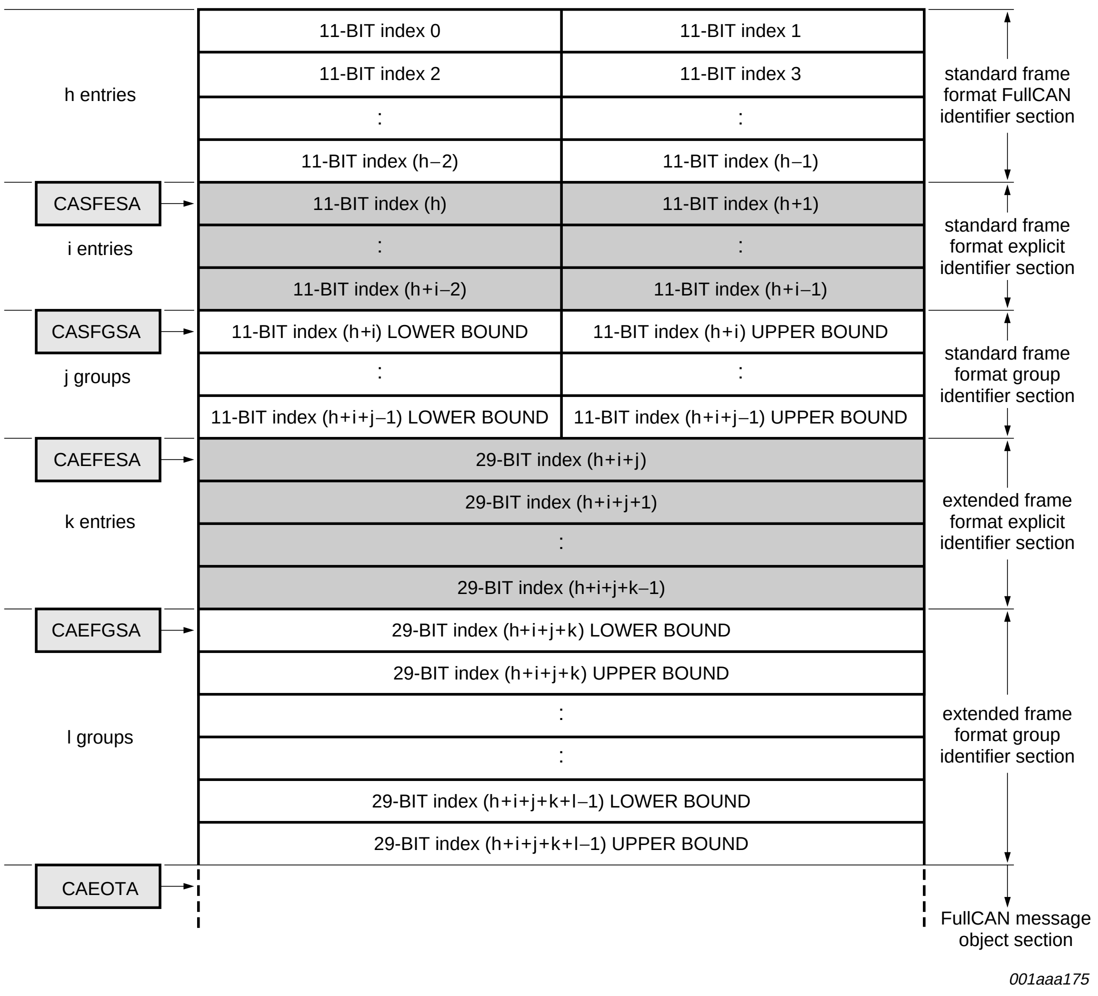

11-BIT index 0
11-BIT index 1
11-BIT index 2
11-BIT index 3
11-BIT index (h–2)
11-BIT index (h–1)
h entries
standard frame format FullCAN identifier section
CASFESA
11-BIT index (h)
11-BIT index (h+1)
i entries
11-BIT index (h+i–2)
11-BIT index (h+i–1)
standard frame format explicit identifier section
CASFGSA
11-BIT index (h+i) LOWER BOUND
11-BIT index (h+i) UPPER BOUND
j groups
11-BIT index (h+i+j–1) LOWER BOUND
11-BIT index (h+i+j–1) UPPER BOUND
standard frame format group identifier section
CAEFESA
29-BIT index (h+i+j)
29-BIT index (h+i+j+1)
k entries
29-BIT index (h+i+j+k–1)
extended frame format explicit identifier section
CAEFGSA
29-BIT index (h+i+j+k) LOWER BOUND
29-BIT index (h+i+j+k) UPPER BOUND
l groups
29-BIT index (h+i+j+k+l–1) LOWER BOUND
29-BIT index (h+i+j+k+l–1) UPPER BOUND
extended frame format group identifier section
CAEOTA
FullCAN message object section
001aaa175

**Fig 73. ID-look-up table memory**

## 8.1 Standard frame-format FullCAN identifier section

If the CAN controller is set into FullCAN mode (EFCAN = 1) the FullCAN identifier section in the look-up table is enabled: otherwise the acceptance filter ignores this section. The entries in the FullCAN identifier section must be arranged in ascending numerical order; one per half-word and two per word (see Figure 21–73).

Since each CAN controller has its own address map, each table entry also contains the number of the CAN controller to which it applies. This section starts at the offset 00h and contains identifiers index 0 to (h-1). The bit allocation is given in Table 21–258.

**Table 258. Standard frame-format FullCAN identifier section**

| Bit | Symbol | Description |
|---|---|---|
| 31 to 29 | SCC | Even index: CAN controller number |
| 28 | MDB | Even index: message disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 27 | - | Not used |
| 26 to 16 | ID[28:18] | Even index: 11-bit CAN 2.0 B identifier |

**Table 258. Standard frame-format FullCAN identifier section** *…continued*

| Bit | Symbol | Description |
|---|---|---|
| 15 to 13 | SCC | Odd index: CAN controller number |
| 12 | MDB | Odd index: message disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 11 | - | Not used |
| 10 to 0 | ID[28:18] | Odd index: 11-bit CAN 2.0 B identifier |

If an incoming message is detected the acceptance filter first tries to find the ID in the FullCAN section, then continues by searching the following sections. In the event of an identifier match during the acceptance filter process, the received FullCAN message object data is moved from the receive buffer of the appropriate CAN controller into the FullCAN message-object section. Table 21–259 shows the detailed layout structure of one FullCAN message stored in the FullCAN message-object section of the look-up table. The base address of a specific message-object data can be calculated by the contents of the CAEOTA and the index i of the ID in the section (see Figure 21–73). Message object data address = CAEOTA + (12 $\times$ i).

**Table 259. FullCAN message-object layout**

| Bit | Symbol | Description |
|---|---|---|
| **Msg_ObjAddr + 0** | | |
| 31 | FF | CAN frame format |
| 30 | RTR | Remote frame request |
| 29 to 26 | - | Not used |
| 25 to 24 | SEM[1:0] | Semaphore bits |
| 23 to 23 | - | Not used |
| 22 to 16 | RXDLC[6:0] | Data-length code |
| 15 to 11 | - | Not used |
| 10 to 0 | ID[28:18] | Identifier bits 28 to 18 |
| **Msg_ObjAddr + 4** | | |
| 31 to 24 | RXDATA4[7:0] | Receive data 4 |
| 23 to 16 | RXDATA3[7:0] | Receive data 3 |
| 15 to 8 | RXDATA2[7:0] | Receive data 2 |
| 7 to 0 | RXDATA1[7:0] | Receive data 1 |
| **Msg_ObjAddr + 8** | | |
| 31 to 24 | RXDATA8[7:0] | Receive data 8 |
| 23 to 16 | RXDATA7[7:0] | Receive data 7 |
| 15 to 8 | RXDATA6[7:0] | Receive data 6 |
| 7 to 0 | RXDATA5[7:0] | Receive data 5 |

Since the FullCAN message-object section of the look-up table can be accessed both by the acceptance filter internal-state machine and by the CPU, there is a method for ensuring that no CPU reads from a FullCAN message-object occurring while the internal state-machine is writing to that object. The acceptance filter uses a three-state semaphore encoded with the two semaphore bits SEM[1:0] for each message object. This mechanism provides the CPU with information about the current state of acceptance filter internal-state machine activity in the FullCAN message-object section.

The semaphore operates in the following manner:

- SEM[1:0] = 01: Acceptance filter is in the process of updating the buffer
- SEM[1:0] = 11: Acceptance Filter has finished updating the buffer
- SEM[1:0] = 00: Either the CPU is in the process of reading from the buffer, or no update since last reading from the buffer

Before writing the first data to a message object SEM[1:0] is set to 01. After having written the last data byte into the message object the acceptance filter internal-state machine will update the semaphore bits by setting SEM[1:0] = 11.

Before reading from a message object, the CPU should read SEM[1:0] to determine the current state of the message object. If SEM[1:0] = 01, the internal state machine is currently active in this message object. If SEM[1:0] = 11, the object is available for reading.

Before the CPU begins reading from the message object it should clear SEM[1:0] = 00, and when the CPU has finished reading it should check SEM[1:0] again. In the case of SEM[1:0] unequal to 00, the message object has been changed during reading, so the contents of the message object should be read out once again. If on the other hand SEM[1:0] = 00 as expected, this means that the valid data has been successfully read by the CPU.

Conditions to activate the FullCAN mode:

- The EFCAN bit in the CAMODE register has to be set
- The start-offset address of the standard frame-format explicit identifier section CASFESA has to be greater than 0
- The available space for the FullCAN message-object section must be large enough to store one object for any FullCAN identifier

## 8.2 Standard frame-format explicit identifier section

The entries of the standard frame-format explicit identifier section must be arranged in ascending numerical order, one per half-word and two per word (see Figure 21–73). Since each CAN controller has its own address map each entry also contains the number of the CAN controller to which it applies.

This section starts with the CASFESA start-address register and contains the identifiers index h to index (h + i – 1). The bit allocation of the first word is given in Table 21–260.

**Table 260. Standard frame-format explicit identifier section**

| Bit | Symbol | Description |
|---|---|---|
| 31 to 29 | SCC | Even index: CAN controller number |
| 28 | MDB | Even index: message disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 27 | - | Not used |
| 26 to 16 | ID[28:18] | Even index: 11-bit CAN 2.0 B identifier |
| 15 to 13 | SCC | Odd index: CAN controller number |

Table 260. Standard frame-format explicit identifier section *…continued*

| Bit | Symbol | Description |
|---|---|---|
| 12 | MDB | Odd index: message disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 11 | - | Not used |
| 10 to 0 | ID[28:18] | Odd index: 11-bit CAN 2.0 B identifier |

By means of the message-disable bits particular CAN identifiers can be turned on and off dynamically from acceptance filtering. When the acceptance filter function is enabled only the message-disable bits in the acceptance-filter look-up table memory can be changed by software. Disabled entries must maintain the ascending sequence of identifiers.

## 8.3 Standard frame-format group identifier section

The table of the standard frame- format group identifier section contains paired upper and lower bounds, one pair per word. These pairs must be arranged in ascending numerical order (see Figure 21–73).

This section starts with the CASFGSA start address register and contains the identifiers index (h + i) lower bound to index (h + i + j – 1) upper bound. The bit allocation of the first word is given in Table 21–261.

Table 261. SFF group identifier section

| Bit | Symbol | Description |
|---|---|---|
| 31 to 29 | SCC | Lower bound: CAN controller number |
| 28 | MDB | Lower bound: message disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 27 | - | Not used |
| 26 to 16 | ID[28:18] | Lower bound: 11-bit CAN 2.0 B identifier |
| 15 to 13 | SCC | Upper bound: CAN controller number |
| 12 | MDB | Upper bound: message-disable bit. Logic 0 is message enabled and logic 1 is message disabled |
| 11 | - | Not used |
| 10 to 0 | ID[28:18] | Upper bound: 11-bit CAN 2.0 B identifier |

By means of the message-disable bits particular CAN identifier groups can be turned on and off dynamically from acceptance filtering. When the acceptance-filter function is enabled only the message-disable bits in the acceptance-filter look-up table memory can be changed by software. Note that in this section the lower bound and upper bound message-disable bit must always have the same value. Disabled entries must maintain the ascending sequence of identifiers.

## 8.4 Extended frame-format explicit identifier section

If extended identifiers (29-bit) are used they can be configured either in this section or in the following section. The table of extended frame-format explicit identifiers must be arranged in ascending numerical order (see Figure 21–73).

This section starts with CAEFESA start-address register and contains the identifiers index (h + i + j) to index (h + i + j + k – 1). The bit allocation of the first word is given in Table 21–262.

Table 262. Extended frame-format explicit identifier section

| Bit | Symbol | Description |
|---|---|---|
| **EFF_GRP_ start address** | | |
| 31 to 29 | SCC | CAN controller number |
| 28 to 0 | ID[28:0] | 29-bit CAN 2.0 B identifier |

## 8.5 Extended frame-format group identifier section

The extended frame-format group identifier section must contain an even number of entries of the same form as in the extended frame-format explicit identifier section (see Figure 21–73). Like the explicit identifier section, the group identifier section must be arranged in ascending numerical order. The upper and lower bounds in the section are implicitly paired as an inclusive group of extended addresses, so that any received address which falls in the inclusive group is accepted and received. Software must maintain the section to consist of such word pairs.

This section starts with CAEFGSA start address register and contains the identifiers index (h + i + j + k) lower bound to index (h + i + j + k + l – 1) upper bound. The bit allocation is given in Table 21–263.

Table 263. Extended frame-format group identifier section

| Bit | Symbol | Description |
|---|---|---|
| **CAEFGSA start address** | | |
| 31 to 29 | SCC | Lower bound: CAN controller number |
| 28 to 0 | ID[28:0] | Lower bound: 29-bit CAN 2.0 B identifier |
| **CAEFGSA start address + 4** | | |
| 31 to 29 | SCC | Upper bound: CAN controller number |
| 28 to 0 | ID[28:0] | Upper bound: 29-bit CAN 2.0 B identifier |

## 8.6 CAN acceptance filter registers

The complete register layout of the CAN acceptance filter is shown in Figure 21–74. Refer to it for resolving register, register-slice and bit names.

## 8.7 CAN acceptance-filter mode register

The ACCBP and ACCOFF bits of the acceptance-filter mode register CAMODE are used for putting the acceptance filter into the bypass- and off-modes respectively. The EFCAN bit of the mode register can be used to activate FullCAN mode for received 11-bit CAN ID messages.

**Acceptance filter off-mode** is typically used during initialization. In this mode an unconditional access to all registers and the look-up table RAM is possible. CAN messages are not accepted in acceptance filter off-mode and are therefore not stored in the receive buffers of active CAN Controllers.

**Acceptance filter bypass-mode** can be used (for example) to change the acceptance filter configuration in a running system by changing identifiers in the ID look-up table memory. Software acceptance filtering has to be used during this reconfiguration.

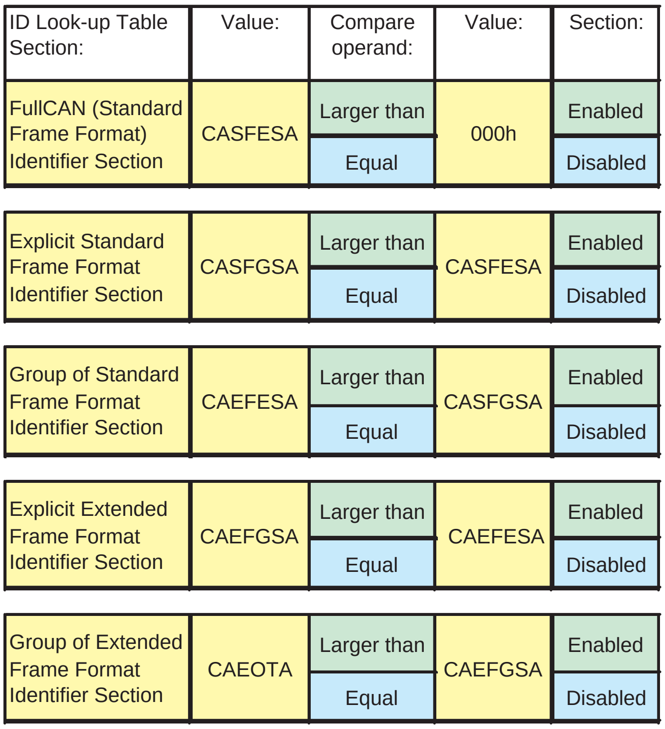
Use the ID ready interrupt IDI and the receive interrupt RI. In this mode all CAN messages are accepted and stored in the receive buffers of active CAN Controllers.

With the activated FullCAN Mode, received FullCAN messages are automatically stored by the acceptance filter in the FullCAN message-object section (see also Section 21–13 for more details).

## 8.8 Section start-registers of the ID look-up table memory

Four 12-bit section configuration registers CASFESA, CASFGSA, CAEFESA and CAEFGSA define the boundaries of the different identifier sections in the ID look-up table memory (see Figure 21–75). The fifth 12-bit section configuration register, the end-of-table address register CAEOTA, defines the end of all identifier sections. The end-of-table address also assigns the start address of the section where FullCAN message objects (if enabled) are stored. See also the example in Section 21–13.

A write-access to all section configuration registers is only possible during acceptance filter off- and bypass-modes. Read-access is allowed in all acceptance filter modes.



| ID Look-up Table Section: | Value: | Compare operand: | Value: | Section: |
|---|---|---|---|---|
| FullCAN (Standard Frame Format) Identifier Section | CASFESA | Larger than | 000h | Enabled |
| | | Equal | | Disabled |
| Explicit Standard Frame Format Identifier Section | CASFGSA | Larger than | CASFESA | Enabled |
| | | Equal | | Disabled |
| Group of Standard Frame Format Identifier Section | CAEFESA | Larger than | CASFGSA | Enabled |
| | | Equal | | Disabled |
| Explicit Extended Frame Format Identifier Section | CAEFGSA | Larger than | CAEFESA | Enabled |
| | | Equal | | Disabled |
| Group of Extended Frame Format Identifier Section | CAEOTA | Larger than | CAEFGSA | Enabled |
| | | Equal | | Disabled |

**Fig 74. Section configuration register settings**

## 8.9 CAN ID look-up table memory

The CAN identifier look-up table memory can contain explicit CAN identifiers and groups of identifiers for standard and extended CAN frame formats. These are listed as a table by source CAN channel (SCC) in ascending order, together with a CAN identifier in each section.

Each CAN identifier is linked to an ID Index number (see also Figure 21–75 and Figure 21–85). For a CAN identifier match, the matching ID index is stored in the identifier index IDI of the message info register CCRXBMI for the appropriate CAN controller (see Section 21–6.1 for more details).

## 8.10 CAN acceptance-filter search algorithm

The identifier-screening process of the acceptance filter starts in the following order:

1. FullCAN (standard frame-format) identifier section
2. Explicit standard frame-format identifier section
3. Group of standard frame-format identifier section
4. Explicit extended frame-format identifier section
5. Group of extended frame-format identifier section

**Remark:** Only activated sections take part in the screening process.

In cases where equal message identifiers of the same frame format are defined in more than one section, the first match ends the screening process for this identifier. For example, if the same source CAN channel in conjunction with the identifier is defined in the FullCAN, explicit standard frame-format and group of standard frame-format identifier sections, screening will finish with the match in the FullCAN section.

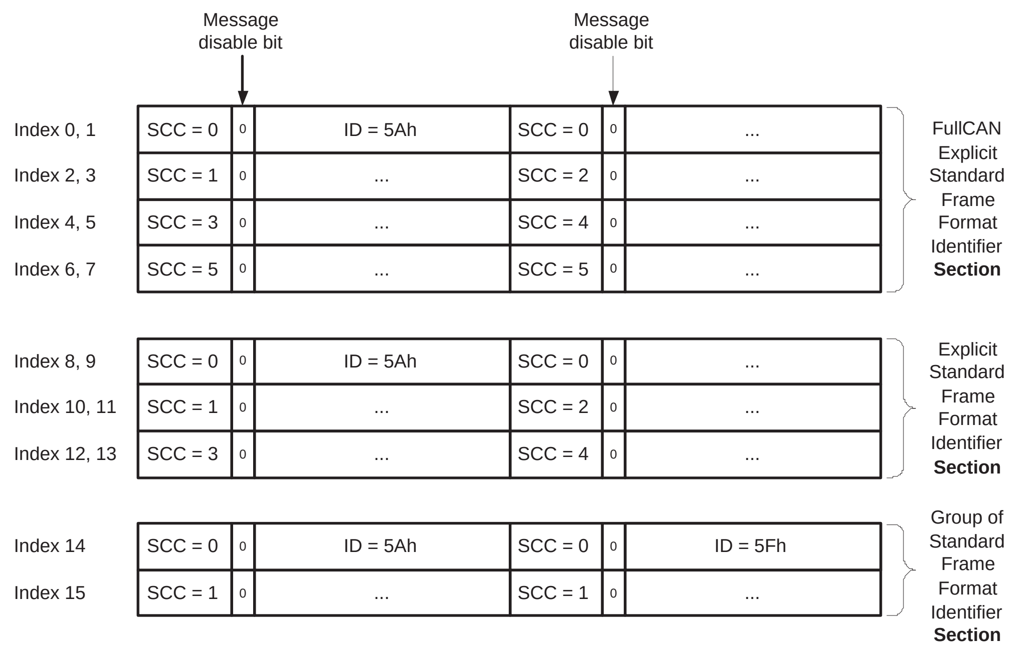


Fig 75. ID look-up table example explaining the search algorithm

In Figure 21–75, identifiers with their SCC have been defined in the FullCAN, explicit and group of standard frame-format identifier sections. The identifier 5Ah of Source CAN Channel 0 is defined in all three sections. With this configuration, incoming CAN messages on Source CAN Channel 0 with a 5Ah identifier find a match in the FullCAN section.

It is possible to disable the 5Ah identifier in the FullCAN section. Then, the screening process would be finished with the match in the explicit identifier section.

The first group in the group identifier section has been defined so that incoming CAN messages with identifiers of 5Ah up to 5Fh are accepted on SCC 0. As stated above, the identifier 5Ah would find a match in the FullCAN or explicit identifier sections if enabled. The rest of the defined identifiers of this group (5Bh to 5Fh) find a match in this group identifier section.

In this way the user can switch dynamically between different filter modes for the same identifiers.

### 8.11 CAN central status registers

For easy and fast access, all the CAN controller status bits from each CAN controller status register are bundled together. For example, the Tx status of all CAN controllers can be read at once with one 32-bit word access. The status registers are read-only and allow byte, half-word and word access.

## 9. Register overview

Table 264. Register overview: CAN controller (base address 0xE008 0000 (CAN0) , 0xE008 1000 (CAN1))

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| CCMODE | R/W | 00 | CAN controller mode register | 0x01 | see Table 21–268 |
| CCCMD | W | 0x04 | CAN controller command register | 0x00 | see Table 21–269 |
| CCGS | R/W | 0x08 | CAN controller global status register | 0x0000 003C | see Table 21–270 |
| CCIC | R | 0x0C | CAN controller interrupt and capture register | 0x0000 0000 | see Table 21–271 |
| CCIE | R/W | 0x10 | CAN controller interrupt- enable register | 0x000 | see Table 21–273 |
| CCBT | R/W | 0x14 | CAN controller bus-timing register | 0x1C 0000 | see Table 21–274 |
| CCEWL | R/W | 0x18 | CAN controller error- warning limit register | 0x60 | see Table 21–275 |
| CCSTAT | R | 0x1C | CAN controller status register | 0x3C 3C3C | see Table 21–276 |
| CCRXBMI | R/W | 0x20 | CAN controller receive- buffer message info register | 0x0000 0000 | see Table 21–277 |
| CCRXBID | R/W | 0x24 | CAN controller receive- buffer identifier register | 0x0000 0000 | see Table 21–278 |
| CCRXBDA | R/W | 0x28 | CAN controller receive- buffer data A register | 0x0000 0000 | see Table 21–279 |
| CCRXBDB | R/W | 0x2C | CAN controller receive- buffer data B register | 0x0000 0000 | see Table 21–280 |
| CCTXB1MI | R/W | 0x30 | CAN controller transmit- buffer 1 message info register | 0x0000 0000 | see Table 21–281 |
| CCTXB1ID | R/W | 0x34 | CAN controller transmit- buffer 1 identifier register | 0x0000 0000 | see Table 21–282 |
| CCTXB1DA | R/W | 0x38 | CAN controller transmit- buffer 1 data A register | 0x0000 0000 | see Table 21–283 |
| CCTXB1DB | R/W | 0x3C | CAN controller transmit- buffer 1 data B register | 0x0000 0000 | see Table 21–284 |
| CCTXB2MI | R/W | 0x40 | CAN controller transmit- buffer 2 message info register | 0x0000 0000 | see Table 21–281 |

**Table 264. Register overview: CAN controller (base address 0xE008 0000 (CAN0) , 0xE008 1000 (CAN1))** *…continued*

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| CCTXB2ID | R/W | 0x44 | CAN controller transmit- buffer 2 identifier register | 0x0000 0000 | see Table 21–282 |
| CCTXB2DA | R/W | 0x48 | CAN controller transmit- buffer 2 data A register | 0x0000 0000 | see Table 21–283 |
| CCTXB2DB | R/W | 0x4C | CAN controller transmit- buffer 2 data B register | 0x0000 0000 | see Table 21–284 |
| CCTXB3MI | R/W | 0x50 | CAN controller transmit- buffer 3 message info register | 0x0000 0000 | see Table 21–281 |
| CCTXB3ID | R/W | 0x54 | CAN controller transmit- buffer 3 identifier register | 0x0000 0000 | see Table 21–282 |
| CCTXB3DA | R/W | 0x58 | CAN controller transmit- buffer 3 data A register | 0x0000 0000 | see Table 21–283 |
| CCTXB3DB | R/W | 0x5C | CAN controller transmit- buffer 3 data B register | 0x0000 0000 | see Table 21–284 |

**Table 265. Register overview: CANAFM Look-up table memory (base adress: 0xE008 5000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| CAFMEM | R/W | 0x000 to 0x7FC | CAN ID look-up table memory | - | see Table 21–289 to Table 21–292 |

**Table 266. Register overview: CANAFR acceptance filter (base address: 0xE008 7000)**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| CAMODE | R/W | 0x00 | CAN acceptance-filter mode register | 0x1 | see Table 21–285 |
| CASFESA | R/W | 0x04 | CAN acceptance-filter standard frame explicit start-address register | 0x000 | see Table 21–286 |
| CASFGSA | R/W | 0x08 | CAN acceptance-filter standard frame group start-address register | 0x000 | see Table 21–287 |
| CAEFESA | R/W | 0x0C | CAN acceptance-filter extended frame explicit start-address register | 0x000 | see Table 21–288 |
| CAEFGSA | R/W | 0x10 | CAN acceptance-filter extended frame group start-address register | 0x000 | see Table 21–289 |
| CAEOTA | R/W | 0x14 | CAN acceptance-filter end- of-table address register | 0x000 | see Table 21–290 |
| CALUTEA | R | 0x18 | CAN acceptance-filter look-up table error address register | 0x000 | see Table 21–291 |
| CALUTE | R | 0x1C | CAN acceptance-filter look-up table error register | 0x0 | see Table 21–292 |
| FCANIE | R/W | 0x20 | Global FullCAN acceptance register | 0x0 | Table 21–293 |
| FCANIC0 | R/W | 0x24 | FullCAN interrupt and capture register 0 | 0x0 | Table 21–294 |
| FCANIC1 | R/W | 0x28 | FullCAN interrupt and capture register 1 | 0x0 | Table 21–295 |

Table 267. Register overview: CANCS central status (base address: 0xE008 8000)

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| CCCTS | R | 0x0 | CAN controllers central transmit-status register | 0x030303 | see Table 21–296 |
| CCCRS | R | 0x4 | CAN controllers central receive-status register | 0x03 | see Table 21–297 |
| CCCMS | R | 0x8 | CAN controllers central miscellaneous status register | 0x0000 | see Table 21–298 |

Besides the hardware reset value the CAN controller registers have a soft reset mode value.

- A hardware reset overrules a software reset
- If no soft reset value is specified the content is unchanged by a soft reset
- Bits with a single '*' are unchanged on setting the soft reset mode.

The reset value shows the result of a hardware reset, while the soft reset value indicates the result of a software reset when the RM bit is set either by software or due to a bus-off condition.

## 9.1 CAN controller mode register

The CAN controller mode register is used to change the behavior of the CAN controller.

Table 21–268 shows the bit assignment of the CCMODE register.

Table 268. CCMODE register bit description (CCMODE, address 0xE008 0000 (CAN0) and 0xE008 1000 (CAN1))
** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 | RPM[1] | R/W | | Reverse polarity mode |
| | | | 1 | RXDC and TXDC pins are HIGH for a dominant bit |
| | | | 0* | RXDC and TXDC pins are LOW for a dominant bit |
| 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 3 | TPM[1][2] | R/W | | Transmit priority mode |
| | | | 1 | Priority depends on the contents of the transmit priority register within the transmit buffer |
| | | | 0* | Transmit priority depends on the CAN identifier |
| 2 | STM[1] | R/W | | Self-test mode |
| | | | 1 | The controller will consider a transmitted message successful if there is no acknowledgment. Use this state in conjunction with the self-reception request bit in the CAN controller command register |
| | | | 0* | Transmitted message must be acknowledged to be considered as successful |

**Table 268. CCMODE register bit description (CCMODE, address 0xE008 0000 (CAN0) and 0xE008 1000 (CAN1))** *…continued*

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 1 | LOM[1][3] | R/W | | Listen-only mode |
| | | | 1 | The controller gives no acknowledgment on CAN even if a message is successfully received. Messages cannot be sent, and the controller operates in error-passive mode |
| | | | 0* | The CAN controller acknowledges a successfully received message |
| 0 | RM[4][5] | R/W | | Soft reset mode |
| | | | 1** | CAN operation is disabled, and writable registers can be written to |
| | | | 0 | CAN controller operates and certain registers cannot be written to |

[1] A write-access to the RPM, TPM, STM and LOM registers is possible only if soft reset mode has previously been entered.

[2] In cases where the same transmit priority or the same ID is chosen for more than one buffer, the transmit buffer with the lowest buffer number is sent first.

[3] This mode of operation forces the CAN controller to be error-passive. Message transmission is not possible.

[4] During a hardware reset or when the bus status bit is set to 1 (bus-off), the soft reset mode bit is set to 1 (present). After the soft reset mode bit has been set to 0 the CAN controller will wait for:

a) one occurrence of bus-free signal (11 recessive bits) if the preceding reset has been caused by a hardware reset or a CPU-initiated reset.

b) 128 occurrences of bus-free signal, if the preceding reset has been caused by a CAN controller-initiated bus-off, before re-entering the bus-on mode

[5] When entering soft reset mode it is not possible to access any other register within the same instruction.

## 9.2 CAN controller command register

The CAN controller command register initiates an action in the transfer layer of the CAN controller.

The CCCMD register is write-only. Table 21–269 shows the bit assignment of the CCCMD register.

**Table 269. CAN controller command register bit description (CCCMD, address 0xE008 0004 (CAN0) and 0xE008 1004 (CAN1))**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 | STB3 | W | | Select transmit buffer 3 |
| | | | 1 | Transmit buffer 3 is selected for transmission. |
| 6 | STB2 | W | | Select transmit buffer 2 |
| | | | 1 | Transmit buffer 2 is selected for transmission |
| 5 | STB1 | W | | Select transmit buffer 1 |
| | | | 1 | Transmit buffer 1 is selected for transmission |

**Table 269. CAN controller command register bit description (CCCMD, address 0xE008 0004 (CAN0) and 0xE008 1004 (CAN1))** *…continued*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 4 | SRR[1][2][3] | W | | Self-reception request |
| | | | 1 | A message is transmitted from the selected transmit buffer and received simultaneously. Transmission and self-reception request has to be set simultaneously with STB3, STB2 or STB1 |
| 3 | CDO | W | | Clear data overrun |
| | | | 1 | The data-overrun bit in the CAN controller status register is cleared. This command bit is used to clear the data-overrun condition signalled by the data-overrun status bit. As long as the data-overrun status bit is set no further data-overrun interrupt is generated |
| 2 | RRB[4] | W | | Release receive buffer |
| | | | 1 | The receive buffer, representing the message memory space in the double receive buffer, is released |
| 1 | AT[3][5] | W | | Abort transmission |
| | | | 1 | If not already in progress, a pending transmission request for the selected transmit buffer is cancelled. If the abort-transmission and transmit-request bits are set in the same write operation, frame transmission is attempted once. No retransmission is attempted if an error is flagged or if arbitration has been lost |
| 0 | TR[2][3][5] | W | - | Transmission request |
| | | | 1 | A message from the selected transmit buffer is queued for transmission |

[1] On self-reception request a message is transmitted and simultaneously received if the acceptance filter is set to the corresponding identifier. A receive and a transmit interrupt indicates correct self-reception (see also the self-test mode (STM) bit in the mode register; see Table 21–268).

[2] It is possible to select more than one message buffer for transmission. If more than one buffer is selected (TR = 1 or SRR = 1) the internal transmit-message queue is organized so that, depending on the transmit-priority mode TPM, the transmit buffer with the lowest CAN identifier (ID) or the lowest 'local priority' (TXPRIO) wins the prioritization and is sent first.

[3] Setting the command bits TR and AT simultaneously results in transmitting a message once. No retransmission will be performed in the case of an error or lost arbitration (single-shot transmission). Setting the command bits SRR and AT simultaneously results in sending the transmit message once using the self-reception feature. No retransmission will be performed in the case of an error or lost arbitration. Setting the command bits TR, AT and SRR simultaneously results in transmitting a message once as described for TR and AT. Immediately the transmit status bit is set within the status register the internal transmission request bit is automatically cleared. Setting TR and SRR simultaneously will ignore the set SRR bit.

[4] After reading the contents of the receive buffer the CPU can release this memory space by setting the RRB bit to 1. This may result in another message becoming immediately available. If there is no other message available the receive-interrupt bit is reset. If the RRB command is given it will take at least two internal clock cycles before a new interrupt is generated.

[5] The AT bit is used when the CPU requires suspension of the previously requested transmission; e.g. to transmit a more urgent message first. A transmission already in progress is not stopped. To see if the original message has been either transmitted successfully or aborted, the transmission-complete status bit should be checked. This should be done after the transmit buffer-status bit has been set to 1or a transmit interrupt has been generated.

[6] If the TR or the SRR bits were set to 1 in a previous command, this cannot be cancelled by resetting the bits. The requested transmission can only be cancelled by setting the AT bit.

## 9.3 CAN controller global status register

The CAN controller global status register reflects the global status of the CAN controller including the transmit and receive error counter values.

Table 21–270 shows the bit assignment of the CCGS register.

**Table 270. CCGS register bit description (CCGS, address 0xE008 0008 (CAN0) and 0xE008 1008 (CAN1))**
** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | TXERR[7:0] | R/W | | Transmit error counter. This register reflects the current value of the transmit error counter. It is only writable in soft reset mode. If a bus-off event occurs the transmit error counter is initialized to 127 to count the minimum protocol-defined time (128 occurrences of the bus-free signal). Reading the transmit error counter during this time gives information about the status of the bus-off recovery. If bus-off is active a write-access to the transmit error counter in the range 0 to 254 clears the bus-off flag, and the controller waits for one occurrence of 11 consecutive recessive bits (bus-free) after clearing the soft reset mode bit |
| | | | 00h* | |
| 23 to 16 | RXERR[7:0] | R/W | | Receive error counter.This register reflects the current value of the receive error counter and is only writable in soft reset mode. If a bus-off event occurs the receive error counter is initialized to 00h. As long as the bus-off condition is valid writing to this register has no effect |
| | | | 00h* | |
| 15 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 | BS[1] | R | | Bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0** | |
| 6 | ES[2] | R | | Error status |
| | | | 1 | One or both of the transmit and receive error counters has reached the limit set in the error warning limit register |
| | | | 0** | |

**Table 270. CCGS register bit description (CCGS, address 0xE008 0008 (CAN0) and 0xE008 1008 (CAN1))** *…continued*

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 5 | TS[3] | R | | Transmit status |
| | | | 1** | The CAN controller is transmitting a message |
| 4 | RS[3] | R | | Receive status |
| | | | 1** | The CAN controller is receiving a message |
| 3 | TCS[4] | R | | Transmission-complete status |
| | | | 1* | All requested message transmissions have completed successfully |
| | | | 0 | At least one of the previously requested transmissions has not yet completed |
| 2 | TBS | R | | Transmit-buffer status |
| | | | 1** | All transmit buffers are available for the CPU |
| | | | 0 | At least one of the transmit buffers contains a previously queued message that has not yet been sent |
| 1 | DOS[5] | R | | Data-overrun status |
| | | | 1 | A message was lost because the preceding message to this CAN controller was not read and released quickly enough |
| | | | 0** | No data overrun has occurred |
| 0 | RBS[6] | R | | Receive-buffer status |
| | | | 1 | At least one complete message is available in the double receive buffer. If no subsequent received message is available this bit is cleared by the release receive-buffer command in the CAN controller command register |
| | | | 0** | No message is available in the double receive buffer |

[1] When the transmit error counter exceeds the limit of 255 the BS bit is set to 1(bus-off), the CAN controller sets the soft reset mode bit to 1 (present) and an error warning interrupt is generated if enabled. Afterwards the transmit error counter is set to 127 and the receive error counter is cleared. It stays in this mode until the CPU clears the soft-reset mode bit. Once this is completed the CAN controller waits the minimum protocol-defined time (128 occurrences of the bus-free signal) counting down the transmit error counter. After that the BS bit is cleared (bus-on), the error status bit is set to 0 (OK), the error counters are reset and an error warning interrupt is generated if enabled. Reading the Tx error counter during this time gives information about the status of the bus-off recovery.

[2] Errors detected during reception or transmission affect the error counters according to the CAN specification. The ES bit is set when at least one of the error counters has reached or exceeded the error-warning limit. An error-warning interrupt is generated if enabled. The default value of the error-warning limit after hardware reset is 96 decimal, see also Table 21–275, CCEWL register bits.

[3] If both the RS and the TS bits are 0 (idle) the CAN bus is idle. If both bits are set the controller is waiting to become idle again. After hardware reset 11 consecutive recessive bits have to be detected until idle status is reached. After bus-off this takes 128 detection cycles of 11 consecutive recessive bits.

[4] The TCS bit is set to 0 (incomplete) whenever the transmission request bit or the self -reception request bit is set to 1 for at least one out of the three transmit buffers. The TCS bit remains 0 until all messages have been successfully transmitted .

[5] If there is not enough space to store the message within the receive buffer, that message is dropped and the data-overrun condition is signalled to the CPU the moment the message becomes valid. If this message is not completed successfully (e.g. because of an error) no overrun condition is signalled.

[6] After reading all messages and releasing their memory space with the command 'release receive buffer' this bit is cleared.

## 9.4 CAN controller interrupt and capture register

The CAN controller interrupt and capture register allows the identification of an interrupt source. Reading the interrupt register clears all interrupt bits except the receive interrupt bit, which requires the release receive-buffer command. If there is another message available within the receive buffer after the release receive-buffer command the receive interrupt is set again: otherwise the receive interrupt stays cleared.

Bus errors are captured in a detailed error report. When a transmitted message loses arbitration the bit where the arbitration was lost is captured. Once either of these registers is captured its value remains the same until it is read, after which it is released to capture a new value.

The CCIC register is read-only. Table 21–271 shows its bit assignment.

Table 271. CAN controller interrupt and capture register bit description (CCIC, address 0xE008 000C (CAN0) and 0xE008 100C (CAN1))

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 29 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 28 to 24 | ALCBIT[4:0] | R | | Arbitration-lost bit. If arbitration is lost while transmitting a message the bit number within the frame is captured into this register |
| | | | 00h* | Arbitration lost in the first (most significant) bit of the identifier |
| | | | : | : |
| | | | 0Bh | 11: arbitration lost in SRTR bit (RTR bit for standard-frame messages) |
| | | | 0Ch | 12: arbitration lost in IDE bit 13: arbitration lost in 12th bit of identifier (extended-frame only) |
| | | | : | : |
| | | | 1Eh | 30: arbitration lost in last bit of identifier (extended-frame only) |
| | | | 1Fh | 31: arbitration lost in RTR bit (extended frames only) |
| 23 and 22 | ERRT[1:0] | R | | Error type. The bus error type is captured in this register |
| | | | 00* | Bit error |
| | | | 01 | Form error |
| | | | 10 | Stuff error |
| | | | 11 | Other error |
| 21 | ERRDIR | R | | Error direction |
| | | | 1 | The bus error is captured during receiving |
| | | | 0* | The bus error is captured during transmitting |
| 20 to 16 | ERRCC[4:0] | R | | Error-code capture. The location of the error within the frame is captured in this register; see Table 21–272 |
| | | | 00h* | |
| 15 to 11 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 10 | TI3 | R | | Transmit interrupt 3 |
| | | | 1 | Transmit buffer status 3 is released (transition from logic 0 to logic 1) and the transmit interrupt-enable 3 is set |
| | | | 0** | |
| 9 | TI2 | R | | Transmit interrupt 2 |
| | | | 1 | Transmit buffer status 2 is released (transition from logic 0 to logic 1) and the transmit interrupt-enable 2 is set |
| | | | 0** | |

**Table 271. CAN controller interrupt and capture register bit description (CCIC, address 0xE008 000C (CAN0) and 0xE008 100C (CAN1))** *…continued*

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 8 | IDI | R | | ID ready interrupt |
| | | | 1 | A CAN identifier has been received in acceptance filter bypass-mode and the ID ready interrupt-enable is set |
| | | | 0** | |
| 7 | BEI | R | | Bus error interrupt |
| | | | 1 | A CAN controller has detected a bus error and the bus error interrupt-enable is set |
| | | | 0* | |
| 6 | ALI | R | | Arbitration-lost interrupt |
| | | | 1 | The CAN controller has lost arbitration while attempting to transmit and the arbitration lost interrupt-enable is set |
| | | | 0** | |
| 5 | EPI | R | | Error-passive interrupt |
| | | | 1 | The CAN controller has reached the error-passive status (at least one error counter exceeds the CAN protocol defined level of 127) or if the CAN controller is in error-passive status and enters error-active status again, and the error-passive interrupt enable is set |
| | | | 0** | |
| 4 | reserved | R | - | Reserved; read as logic 0 |
| 3 | DOI | R | | Data-overrun interrupt |
| | | | 1 | The data-overrun occurred and the data-overrun interrupt enable is set |
| | | | 0** | |
| 2 | EWI | R | | Error warning interrupt |
| | | | 1 | A change of either the error status or bus status occurred and the error warning interrupt-enable is set |
| | | | 0* | |
| 1 | TI1 | R | | Transmit interrupt 1 |
| | | | 1 | The transmitter buffer status 1 is released (transition from logic 0 to logic 1) and the transmit interrupt-enable 1 is set |
| | | | 0** | |
| 0 | RI[1] | R | | Receive interrupt |
| | | | 1 | The receive-buffer status is logic 1 and the receive interrupt-enable is set |
| | | | 0** | |

[1] The RI bit is not cleared on a read-access to the interrupt register. Giving the command ‘Release receive buffer will clear RI temporarily. If there is another message available within the receive buffer after the release command, RI is set again: otherwise RI stays cleared.

**Table 272. Bus error capture code values**

| ERRCC [4:0] | Function |
|---|---|
| 0 0000 | Reserved |
| 0 0001 | Reserved |
| 0 0010 | Identifier bits 21 to 28 |
| 0 0011 | Start of frame |
| 0 0100 | Standard frame RTR bit |
| 0 0101 | IDE bit |
| 0 0110 | Reserved |
| 0 0111 | Identifier bits 13 to 17 |
| 0 1000 | CRC sequence |
| 0 1001 | Reserved bit 0 |
| 0 1010 | Data field |
| 0 1011 | Data-length code |
| 0 1100 | Extended-frame RTR bit |
| 0 1101 | Reserved bit 1 |
| 0 1110 | Identifier bits 0 to 4 |
| 0 1111 | Identifier bits 5 to 12 |
| 1 0000 | Reserved |
| 1 0001 | Active error flag |
| 1 0010 | Intermission |
| 1 0011 | Tolerate dominant bits |
| 1 0100 | Reserved |
| 1 0101 | Reserved |
| 1 0110 | Passive error flag |
| 1 0111 | Error delimiter |
| 1 1000 | CRC delimiter |
| 1 1001 | Acknowledge slot |
| 1 1010 | End of frame |
| 1 1011 | Acknowledge delimiter |
| 1 1100 | Overload flag |
| 1 1101 | Reserved |
| 1 1110 | Reserved |
| 1 1111 | Reserved |

## 9.5 CAN controller interrupt-enable register

The CAN controller interrupt-enable register CCIE enables the different types of CAN controller interrupts.

Table 21–273 shows the bit assignment of the CCIE register.

**Table 273. CAN controller interrupt-enable register bit descriptioN (CCIE, address 0xE008 0010 (CAN0) and 0xE008 1010 (CAN1))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 11 | reserved | R | | Reserved; do not modify. Read as logic 0 |
| 10 | TI3E | R/W | | Transmit interrupt-enable 3 |
| | | | 1 | An interrupt is generated if the transmit buffer status 3 is released (transition from logic 0 to logic 1) |
| | | | 0* | |
| 9 | TI2E | R/W | | Transmit interrupt-enable 2 |
| | | | 1 | An interrupt is generated if the transmit buffer status 2 is released (transition from logic 0 to logic 1) |
| | | | 0* | |
| 8 | IDIE | R/W | | ID ready interrupt enable |
| | | | 1 | An interrupt is generated if a CAN identifier has been received in acceptance filter bypass mode. |
| | | | 0* | |
| 7 | BEIE | R/W | | Bus-error interrupt enable |
| | | | 1 | An interrupt is generated if a CAN controller has detected a bus error |
| | | | 0* | |
| 6 | ALIE | R/W | | Arbitration-lost interrupt enable |
| | | | 1 | An interrupt is generated if the CAN controller has lost arbitration while attempting to transmit |
| | | | 0* | |
| 5 | EPIE | R/W | | Error-passive interrupt enable |
| | | | 1 | An interrupt is generated if the CAN controller has reached error-passive status (at least one error counter exceeds the CAN protocol-defined level of 127) or if the CAN controller is in error-passive status and enters error-active status again |
| | | | 0* | |
| 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 3 | DOIE | R/W | | Data-overrun interrupt enable |
| | | | 1 | An interrupt is generated if the data overrun occurred |
| | | | 0* | |

**Table 273. CAN controller interrupt-enable register bit descriptioN (CCIE, address 0xE008 0010 (CAN0) and 0xE008 1010 (CAN1))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 2 | EWIE | R/W | | Error warning interrupt-enable |
| | | | 1 | An interrupt is generated if either the error status or bus status have changed |
| | | | 0* | |
| 1 | TIE1 | R/W | | Transmit interrupt-enable 1 |
| | | | 1 | An interrupt is generated if the transmit buffer status 1 is released (transition from logic 0 to logic 1) |
| | | | 0* | |
| 0 | RIE | R/W | | Receive- interrupt enable |
| | | | 1 | An interrupt is generated if the receive buffer is not empty |
| | | | 0* | |

## 9.6 CAN controller bus timing register

The CAN controller bus timing register CCBT defines the timing characteristics of the CAN bus. The register is only writable in soft-reset mode.

Table 21–274 shows the bit assignment of the CCBT register.

**Table 274. CAN controller bust timing register bit description (CCBT, address 0xE008 0014 (CAN0) and 0xE008 1014 (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 23 | SAM | R/W | 1 | The bus is sampled three times. Recommended for low- or medium-speed buses where filtering spikes on the bus line are beneficial. |
| | | | 0* | The bus is sampled once. Recommended for high-speed busses |
| 22 to 20 | TSEG2[2:0] | R/W | | Timing segment 2. This is the time segment after the sample point, determined by the formula of [1] |
| | | | 1h* | |
| 19 to 16 | TSEG1[3:0] | R/W | | timing segment 1; time segment before the sample point which is determined by the formula of [2] |
| | | | Ch* | |
| 15 and 14 | SJW[1:0] | R/W | | Synchronization jump width. The synchronization jump length is determined by the formula of [3] |
| | | | 0h* | |

Table 274. CAN controller bust timing register bit description (CCBT, address 0xE008 0014 (CAN0) and 0xE008 1014 (CAN1)) *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 13 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 to 0 | BRP[9:0] | R/W | | Baud-rate prescaler. This derives the CAN clock tscl from the BASE_IVNSS_CLK (branch clocks to the CAN controller: CLK_IVNSS_CANC*)[4] |
| | | | 000h* | |

[1] $t_{seg2} = t_{scl} \times (TSEG2 + 1)$

[2] $t_{seg1} = t_{scl} \times (TSEG1 + 1)$

[3] $t_{sjw} = t_{scl} \times (SJW + 1)$

[4] $t_{scl} = \frac{BRP + 1}{f_{CLK\_CAN}}$

## 9.7 CAN controller error-warning limit register

The CAN controller error-warning limit register CCEWL sets a limit to the transmit or receive errors at which an interrupt can occur. This register is only writable in soft-reset mode.

Table 21–275 shows the bit assignment of the CCEWL register.

Table 275. CAN controller error-warning limit register bit description (CCEWL, address 0xE008 0018 (CAN0) and 0xE008 1018 (CAN1))

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 to 0 | EWL[7:0] | R/W | | Error warning limit. During CAN operation this value is compared with both the transmit and receive error counters, and if either counter matches the value the error status bit is set |
| | | | 60h* | |

## 9.8 CAN controller status register

The CAN controller status register CCSTAT reflects the transmit status of all three transmit buffers, and also the global status of the CAN controller itself.

The register is read-only. Table 21–276 shows the bit assignment of the CCSTAT register.

**Table 276. CAN controller status register bit description (CCSTAT, address 0xE008 001C (CAN0) and 0xE008 101C (CAN1))**

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 23 | BS | R | | Bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0** | |
| 22 | ES | R | | Error status |
| | | | 1 | One or both of the transmit and receive error counters has reached the limit set in the error warning-limit register |
| | | | 0** | |
| 21 | TS3 | R | | Transmit status 3 |
| | | | 1** | The CAN controller is transmitting a message from transmit buffer 3 |
| 20 | RS | R | | Receive status |
| | | | 1** | The CAN controller is receiving a message |
| 19 | TCS3[1] | R | | Transmission complete status 3 |
| | | | 1* | The last requested message transmissions from transmit buffer 3 have been successfully completed |
| | | | 0 | The previously requested transmission is not yet complete |
| 18 | TBS3[2] | R | | Transmit buffer status 3 |
| | | | 1** | Transmit buffer 3 is available for the CPU |
| | | | 0 | Transmit buffer 3 contains a previously queued message that has not yet been sent |
| 17 | DOS | R | | Data-overrun status |
| | | | 1 | A message was lost because the preceding message to this CAN controller was not read and released quickly enough |
| | | | 0** | No data overrun has occurred |
| 16 | RBS | R | | Receive buffer status |
| | | | 1 | At least one complete message is available in the double receive buffer. If no subsequent received message is available this bit is cleared by the release receive-buffer command in the CAN controller command register |
| | | | 0** | No message is available in the double receive buffer |

**Table 276. CAN controller status register bit description (CCSTAT, address 0xE008 001C (CAN0) and 0xE008 101C (CAN1))** *…continued*

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 15 | BS | R | | Bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0** | |
| 14 | ES | R | | Error status |
| | | | 1 | One or both of the transmit and receive error counters has reached the limit set in the error warning-limit register |
| | | | 0** | |
| 13 | TS2 | R | | Transmit status 2 |
| | | | 1** | The CAN controller is transmitting a message from transmit buffer 2 |
| 12 | RS | R | | Receive status |
| | | | 1** | The CAN controller is receiving a message |
| 11 | TCS2[1] | R | | Transmission complete status 2 |
| | | | 1* | The requested message transmission from transmit buffer 2 has been successfully completed |
| | | | 0 | The previously requested transmission from transmit buffer 2 is not yet completed |
| 10 | TBS2[2] | R | | Transmit buffer status 2 |
| | | | 1** | Transmit buffer 2 is available for the CPU |
| | | | 0 | Transmit buffer 2 contains a previously queued message that has not yet been sent |
| 9 | DOS | R | | Data-overrun status |
| | | | 1 | A message was lost because the preceding message to this CAN controller was not read and released quickly enough |
| | | | 0** | No data overrun has occurred |
| 8 | RBS | R | | Receive buffer status |
| | | | 1 | At least one complete message is available in the double receive buffer. If no subsequent received message is available this bit is cleared by the release receive buffer command in the CAN controller command register |
| | | | 0** | No message is available in the double receive buffer |
| 7 | BS | R | | Bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0** | |

Table 276. CAN controller status register bit description (CCSTAT, address 0xE008 001C (CAN0) and 0xE008 101C (CAN1)) *…continued*

** = reset value; **both reset value and soft reset mode value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 6 | ES | R | | Error status |
| | | | 1 | One or both of the transmit and receive error counters has reached the limit set in the error warning-limit register |
| | | | 0** | |
| 5 | TS1 | R | | Transmit status 1 |
| | | | 1** | The CAN controller is transmitting a message from transmit buffer 1 |
| 4 | RS | R | | Receive status |
| | | | 1** | The CAN controller is receiving a message |
| 3 | TCS1[1] | R | | Transmission-complete status 1 |
| | | | 1* | The requested message transmission from transmit buffer 1 has been successfully completed |
| | | | 0 | The previously requested transmission from transmit buffer 1 has not yet completed |
| 2 | TBS1[2] | R | | Transmit-buffer status |
| | | | 1** | Transmit buffer 1 is available for the CPU |
| | | | 0 | Transmit buffer 1 contains a previously queued message that has not yet been sent |
| 1 | DOS | R | | Data-overrun status |
| | | | 1 | A message was lost because the preceding message to this CAN controller was not read and released quickly enough |
| | | | 0** | No data overrun has occurred |
| 0 | RBS | R | | Receive-buffer status |
| | | | 1 | At least one complete message is available in the double receive buffer. If no subsequent received message is available this bit is cleared by the release receive-buffer command in the CAN controller command register |
| | | | 0** | No message is available in the double receive buffer |

[1] The TCS1 bit is set to 0 (incomplete) whenever the transmission request bit or the self-reception request bit is set to 1 for this TX buffer. The TCS1 bit will remain 0 until a message is successfully transmitted.

[2] If the CPU tries to write to this transmit buffer when the TBS1 bit is 0 (locked), the written byte will not be accepted and will be lost without this being signalled.

## 9.9 CAN controller receive-buffer message info register

The CAN controller receive-buffer message info register CCRXBMI gives the characteristics of the received message. This register is only writable in soft-reset mode.

Table 21–277 shows the bit assignment of the CCRXBMI register.

Table 277. CAN controller receive-buffer message info register bit description (CCRXBMI, address 0xE008 0020 (CAN0) and 0xE008 1020 (CAN1))

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | FF | R | | Frame format |
| | | | 1 | An extended frame-format message has been received |
| | | | 0* | A standard frame-format message has been received |
| 30 | RTR | R | | Remote frame request |
| | | | 1 | A remote frame has been received |
| | | | 0* | A data frame has been received |
| 29 to 20 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 19 to 16 | DLC[3:0] | R | | Data-length code. This register contains the number of data bytes received if bit RTR is logic 0, or the requested number of data bytes if bit RTR is logic 1. Values greater than eight are handled as eight data bytes |
| | | | 0h* | |
| 15 to 11 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 10 | BP | R | | Bypass mode |
| | | | 1 | The message was received in acceptance filter bypass mode, which makes the identifier index field meaningless |
| | | | 0* | |
| 9 to 0 | IDI[9:0] | R | | Identifier index. If bit BP is not set this register contains the zero-based number of the look-up table entry at which the acceptance filter matched the received identifier. Disabled entries in the standard tables are included in this numbering, but will not be considered for filtering |
| | | | 000h* | |

## 9.10 CAN controller receive buffer identifier register

The CAN controller receive-buffer identifier register CCRXBID contains the identifier field of the received message. This register is only writable in soft-reset mode.

Table 21–278 shows the bit assignment of the CCRXBID register.

**Table 278. CAN controller receive buffer identifier register bit description (CCRXBID, address 0xE008 0024 (CAN0) and 0xE008 1024 (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 29 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 28 to 0 | ID[28:0] | R | | Identifier register. This contains the identifier of the received CAN message. If a standard frame-format message has been received the 11 least significant bits represent the 11-bit identifier |
| | | | 0000 0000h* | |

## 9.11 CAN controller receive buffer data A register

The CAN controller receive buffer data A register CCRXBDA contains the first four data bytes of the received message. This register is only writable in soft-reset mode.

Table 21–279 shows the bit assignment of the CCRXBDA register.

**Table 279. CAN controller receive buffer data A register bit description (CCRXBDA, address 0xE008 0028 (CAN0) and 0xE008 1028 (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DB4[7:0] | R | | Data byte 4. If the data-length code value is four or more this register contains the fourth data byte of the received message |
| | | | 00h* | |
| 23 to 16 | DB3[7:0] | R | | Data byte 3. If the data-length code value is three or more this register contains the third data byte of the received message |
| | | | 00h* | |
| 15 to 8 | DB2[7:0] | R | | Data byte 2. If the data-length code value is two or more this register contains the second data byte of the received message |
| | | | 00h* | |
| 7 to 0 | DB1[7:0] | R | | Data byte 1. If the data-length code value is one or more this register contains the first data byte of the received message |
| | | | 00h* | |

## 9.12 CAN controller receive-buffer data B register

The CAN controller receive buffer data B register CCRXBDB contains the second four data bytes of the received message. This register is only writable in soft-reset mode.

Table 21–280 shows the bit assignment of the CCRXBDB register.

**Table 280. CAN controller receive-buffer data B register bit description (CCRXBDB, address 0xE008 002C (CAN0) and 0xE008 102C (CAN1))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DB8[7:0] | R | | Data byte 8. If the data-length code value is eight or more this register contains the eighth data byte of the received message |
| | | | 00h* | |
| 23 to 16 | DB7[7:0] | R | | Data byte 7. If the data-length code value is seven or more this register contains the seventh data byte of the received message |
| | | | 00h* | |
| 15 to 8 | DB6[7:0] | R | | Data byte 6. If the data-length code value is six or more this register contains the sixth data byte of the received message |
| | | | 00h* | |
| 7 to 0 | DB5[7:0] | R | | Data byte 5. If the data-length code value is five or more this register contains the fifth data byte of the received message |
| | | | 00h* | |

## 9.13 CAN controller transmit-buffer message info registers

The CAN controller transmit-buffer message info registers CCTXB1MI, CCTXB2MI and CCTXB3MI each reflect the characteristics of the transmit message. These registers are only writable when the transmit buffer is released (i.e. corresponding transmit-buffer status bit is logic 1).

Table 21–281 shows the bit assignment of the CCTXB1MI, CCTXB2MI and CCTXB3MI registers.

**Table 281. CAN controller transmit-buffer message info register bit description (CCTXB1/2/3MI, addresses 0xE008 0030, 0xE008 0040, 0xE008 0050 (CAN0), and 0xE008 1030, 0xE008 1040, 0xE008 1050 (CAN1))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | FF | R/W | | Frame format |
| | | | 1 | An extended frame-format message is transmitted |
| | | | 0* | A standard frame-format message is transmitted |
| 30 | RTR | R/W | | Remote frame request |
| | | | 1 | A remote frame-format message is transmitted |
| | | | 0* | A data frame-format message is transmitted |
| 29 to 20 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 281. CAN controller transmit-buffer message info register bit description (CCTXB1/2/3MI, addresses 0xE008 0030, 0xE008 0040, 0xE008 0050 (CAN0), and 0xE008 1030, 0xE008 1040, 0xE008 1050 (CAN1))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 19 to 16 | DLC[3:0] | R/W | 0h | Data-length code. This register contains the number of data bytes to be transmitted if bit RTR is logic 0, or the requested number of data bytes if bit RTR is logic 1. Values greater than eight are handled as eight data bytes |
| | | | 0h* | |
| 15 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 to 0 | TXPRIO[7:0] | R/W | | Transmit priority. If the transmit-priority mode bit in the CAN controller mode register is set, the transmit buffer with the lowest transmit-priority value wins the prioritization and is sent first. In cases where the same transmit priority or the same ID is chosen for more than one transmit buffer, the transmit buffer with the lowest buffer number is sent first |
| | | | 00h* | |

## 9.14 CAN controller transmit-buffer identifier registers

The CAN controller transmit buffer identifier registers CCTXB1ID, CCTXB2ID and CCTXB3ID contain the identifier field of the transmit message. These registers are only writable when the transmit buffer is released (i.e corresponding transmit-buffer status bit is logic 1).

Table 21–282 shows the bit assignment of the CCTXB1ID, CCTXB2ID and CCTXB3ID registers.

**Table 282. CAN controller transmit-buffer identifier register bit description (CCTXB1/2/3ID, addresses 0xE008 0034, 0xE008 0044, 0xE008 0054 (CAN0), and 0xE008 1034, 0xE008 1044, 0xE008 1054 (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 29 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 28 to 0 | ID[28:0] | R/W | | Identifier register. This contains the identifier of the transmit CAN message. If a standard frame-format is transmitted the 11 least significant bits must represent the 11-bit identifier |
| | | | 0000 0000h* | |

## 9.15 CAN controller transmit-buffer data A registers

The CAN controller transmit-buffer data A registers CCTXB1DA, CCTXB2DA and CCTXB3DA contain the first four data bytes of the transmit message. These registers are only writable when the transmit buffer is released (i.e. corresponding transmit-buffer status bit is logic 1).

Table 21–283 shows the bit assignment of the CCTXB1DA, CCTXB2DA and CCTXB3DA registers.

**Table 283. CAN controller transmit-buffer data A registers register bit description (CCTXB1/2/3DA, addresses 0xE008 0038, 0xE008 0048, 0xE008 0058 (CAN0), and 0xE008 1038, 0xE008 1048, 0xE008 1058 (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DB4[7:0] | R/W | | Data byte 4. If the data-length code value is four or more this register contains the fourth data byte of the received message |
| | | | 00h* | |
| 23 to 16 | DB3[7:0] | R/W | | Data byte 3. If the data length code value is three or more this register contains the third data byte of the received message |
| | | | 00h* | |
| 15 to 8 | DB2[7:0] | R/W | | Data byte 2. If the data-length code value is two or more this register contains the second data byte of the received message |
| | | | 00h* | |
| 7 to 0 | DB1[7:0] | R/W | | Data byte 1. If the data-length code value is one or more this register contains the first data byte of the received message |
| | | | 00h* | |

## 9.16 CAN controller transmit-buffer data B registers

The CAN controller transmit-buffer data B registers CCTXB1DB, CCTXB2DB and CCTXB3DB contain the second four data bytes of the transmit message. These registers are only writable when the transmit buffer is released (i.e the corresponding transmit-buffer status bit is logic 1).

Table 21–284 shows the bit assignment of the CCTXB1DB, CCTXB2DB and CCTXB3DB registers.

**Table 284. CAN controller transmit-buffer data B register bit description (CCTX1/2/3DB, addresses 0xE008 003C, 0xE008 004C, 0xE008 005C (CAN0), and 0xE008 103C, 0xE008 104C, 0xE008 105C (CAN1))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DB8[7:0] | R/W | | Data byte 8. If the data -length code value is eight or more this register contains the eighth data byte of the received message |
| | | | 00h* | |
| 23 to 16 | DB7[7:0] | R/W | | Data byte 7. If the data-length code value is seven or more this register contains the seventh data byte of the received message |
| | | | 00h* | |

**Table 284. CAN controller transmit-buffer data B register bit description (CCTX1/2/3DB, addresses 0xE008 003C, 0xE008 004C, 0xE008 005C (CAN0), and 0xE008 103C, 0xE008 104C, 0xE008 105C (CAN1))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 15 to 8 | DB6[7:0] | R/W | | Data byte 6. If the data-length code value is six or more this register contains the sixth data byte of the received message |
| | | | 00h* | |
| 7 to 0 | DB5[7:0] | R/W | | Data byte 5. If the data-length code value is five or more this register contains the fifth data byte of the received message |
| | | | 00h* | |

## 10. CAN acceptance-filter register overview

### 10.1 CAN acceptance-filter mode register

The CAN acceptance-filter mode register CAMODE is used to change the behavior of the acceptance filter.

Table 21–285 shows the bit assignment of the CAMODE register.

**Table 285. CAN acceptance-filter mode register bit description (CAMODE, address 0xE008 7000)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 2 | EFCAN | R/W | | FullCAN extension mode |
| | | | 1 | FullCAN functionality is enabled |
| | | | 0* | FullCAN functionality is disabled |
| 1 | ACCBP | R/W | | Acceptance filter bypass |
| | | | 1 | All Rx messages are accepted on enabled CAN controller. Software must set this bit before modifying the contents of any of the acceptance-filter registers, and before modifying the contents of look-up table RAM in any other way than setting or clearing the disable bits in standard-identifier entries |
| | | | 0* | When both this bit and bit ACCOFF are logic 0, the acceptance filter operates to screen received CAN identifiers |
| 0 | ACCOFF | R/W | | Acceptance filter off |
| | | | 1* | If bit ACCBP = 0 the acceptance filter is not operational and all received CAN messages are ignored |
| | | | 0 | The acceptance filter is operational |

## 10.2 CAN acceptance-filter standard-frame explicit start-address register

The CAN acceptance filter standard-frame explicit start-address register CASFESA defines the start address of the section of explicit standard identifiers in the acceptance-filter look-up table. It also indicates the size of the section of standard identifiers which the acceptance filter will search.

Table 21–286 shows the bit assignment of the CASFESA register.

**Table 286. CAN acceptance-filter standard-frame explicit start-address register bit description (CASFESA, address 0xE008 7004)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 to 2 | SFESA[9:0] | R/W | | Standard-frame explicit start-address. This register defines the start address of the section of explicit standard identifiers in acceptance filter look-up table. If the section is empty, write the same value into this register and the SFGSA register. If bit EFCAN = 1, this value also indicates the size of the section of standard identifiers which the acceptance filter will search and (if found) automatically store received messages from in the acceptance-filter section. Write access is only possible during acceptance-filter bypass or acceptance-filter off modes. Read access is possible in acceptance-filter on and off modes.<br>The standard-frame explicit start address is aligned on word boundaries, and therefore the lowest two bits must be always be logic 0 |
| | | | 00h* | |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.3 CAN acceptance-filter standard-frame group start-address register

The CAN acceptance-filter standard-frame group start-address register CASFGSA defines the start address of the section of grouped standard identifiers in the acceptance-filter look-up table.

Table 21–287 shows the bit assignment of the CASFGSA register.

**Table 287. CAN acceptance-filter standard-frame group start-address register bit description (CASFGSA, address 0xE008 7008)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 to 2 | SFGSA[9:0] | R/W | | Standard-frame group start address. This register defines the start address of the section of grouped standard identifiers in the acceptance-filter look-up table. If this section is empty, write the same value in this register and the EFESA register. The largest value that should be written to this register is 7FCh when only the standard explicit section is used and the last word (address 7F8h) in the acceptance-filter look-up table is used. Write access is only possible during acceptance-filter bypass or acceptance-filter off modes; read access is possible in acceptance-filter on and off modes. The standard-frame group start address is aligned on word boundaries and therefore the lowest two bits must be always logic 0 |
| | | | 00h* | |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.4 CAN acceptance-filter extended-frame explicit start-address register

The CAN acceptance-filter extended-frame explicit start-address register CAEFESA defines the explicit start address of the section of extended identifiers in the acceptance-filter look-up table.

Table 21–288 shows the bit assignment of the CAEFESA register.

**Table 288. CAEFESA register bit description (CAEFESA, address 0xE008 700C)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 288. CAEFESA register bit description (CAEFESA, address 0xE008 700C)** *…continued*
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 11 to 2 | EFESA[9:0] | R/W | | Extended-frame explicit start address. This register defines the start address of the section of explicit extended identifiers in acceptance-filter look-up table. If the section is empty write the same value in this register and the EFGSA register. The largest value that should be written to this register is 7FCh, when both extended sections are empty and the last word (address 7F8h) in the acceptance-filter look-up table is used. Write access is only possible in acceptance-filter bypass or acceptance-filter off modes. Read access is possible in acceptance-filter on and off modes. The extended-frame explicit start-address is aligned on word boundaries, and therefore the lowest two bits must be always logic 0 |
| | | | 00h* | |
| 1 and 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.5 CAN acceptance-filter extended-frame group start-address register

The CAN acceptance filter extended frame group start address register CAEFGSA defines the start address of the section of grouped extended-frame identifiers in the acceptance-filter look-up table.

Table 21–289 shows the bit assignment of the CAEFGSA register.

**Table 289. CAN acceptance-filter extended-frame group start-address register bit description (CAEFGSA, address 0xE008 7010)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 to 2 | EFGSA[9:0] | R/W | | Extended-frame group start-address. This register defines the start address of the section of grouped extended identifiers in the acceptance-filter look-up table. If the section is empty write the same value in this register and the EOTA register. The largest value that should be written to this register is 7FCh when the section is empty and the last word (address 7F8h) in the acceptance-filter look-up table is used. Write access is only possible in acceptance-filter bypass or acceptance-filter off modes. Read access is possible in acceptance-filter on and off modes. The extended-frame group start-address is aligned on word boundaries, and therefore the lowest two bits must be always logic 0. |
| | | | 00h* | |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.6 CAN acceptance-filter end of look-up table address register

The CAN acceptance filter end of look-up table address register CAEOTA contains the end-address of the acceptance-filter look-up table.

Table 21–290 shows the bit assignment of the CAEOTA register.

**Table 290. CAN acceptance-filter end of look-up table address register bit description (CAEOTA, address 0xE008 7014)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 to 2 | EOTA[9:0] | R/W | | End of look-up table address. The largest value of the register CAEOTA should never exceed 7FC.<br>If bit EFCAN = 0 the register should contain the next address above the last active acceptance-filter identifier section;<br>If bit EFCAN = 1 the register contains the start address of the FullCAN message object section. In the case of an identifier match in the standard frame-format FullCAN identifier section during acceptance filtering, the received FullCAN message object data is moved from the receive buffer of the appropriate CAN Controller into the FullCAN message object section. Each defined FullCAN Message needs three address lines for the message data in the FullCAN message object data section. Write access is only possible in acceptance-filter bypass or acceptance-filter off modes. Read access is possible in acceptance-filter on and off modes. |
| | | | 00h* | |
| 1 to 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.7 CAN acceptance filter look-up table error address register

The CAN acceptance filter look-up table error address register CALUTEA represents the address in the look-up table at which a problem has been detected when the look-up table error bit is set.

The CALUTEA register is read-only. Table 21–291 shows the bit assignment of the CALUTEA register.

Table 291. CAN acceptance-filter look-up table error address register bit description (CALUTEA, address 0xE008 7018)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 11 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 10 to 2 | LUTEA[8:0] | R | | Look-up table error address. This register contains the address in the look-up table at which the acceptance filter encountered an error in the content of the tables. It is valid when the look-up table error bit is set. Reading this register clears the look-up table error bit LUTE |
| | | | 00h* | |
| 1 and 0 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

## 10.8 CAN acceptance-filter look-up table error register

The CAN acceptance filter look-up table error register CALUTE provides the configuration status of the look-up table contents. In the event of an error an interrupt is generated via the general CAN-interrupt input source of the Vectored Interrupt Controller.

The CALUTE register is read-only. Table 21–292 shows the bit assignment of the CALUTE register.

Table 292. CAN acceptance-filter look-up table error register bit description (CALUTE, address 0xE008 701C)

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 1 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 0 | LUTE | R | | Look-up table error |
| | | | 1 | The acceptance filter has encountered an error in the contents of the look-up table. Reading the LUTEA register clears this bit. This error condition is part of the general CAN-interrupt input source |
| | | | 0* | |

## 10.9 Global FullCANInterrupt Enable register

A write access to the Global FullCAN Interrupt Enable register is only possible when the Acceptance Filter is in the off mode.

Table 293. Global FullCAN Enable register (FCANIE - address 0xE008 7020) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | FCANIE | Global FullCAN Interrupt Enable. When 1, this interrupt is enabled. | 0 |
| 31:1 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

## 10.10 FullCAN Interrupt and Capture registers

For detailed description on these two registers, see Section 21–11.

Table 294. FullCAN Interrupt and Capture register 0 (FCANIC0 - address 0xE008 7024) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | IntPnd0 | FullCan Interrupt Pending bit 0. | 0 |
| ... | IntPndx (0<x<31) | FullCan Interrupt Pending bit x. | 0 |
| 31 | IntPnd31 | FullCan Interrupt Pending bit 31. | 0 |

Table 295. FullCAN Interrupt and Capture register 1 (FCANIC1 - address 0xE008 7028) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 0 | IntPnd32 | FullCan Interrupt Pending bit 32. | 0 |
| ... | IntPndx (32<x<63) | FullCan Interrupt Pending bit x. | 0 |
| 31 | IntPnd63 | FullCan Interrupt Pending bit 63. | 0 |

## 10.11 CAN controller central transmit-status register

The CAN controller central transmit-status register CCCTS provides bundled access to the transmission status of all the CAN controllers. The status flags are the same as those in the status register of the corresponding CAN controller.

The CCCTS register is read-only. Table 21–296 shows the bit assignment of the CCCTS register.

Table 296. CAN controller central transmit-status register bit description
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 18 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 17 | TCS1 | R | | CAN controller 1 transmission-completed status |
| | | | 1* | Transmission was successfully completed |
| 16 | TCS0 | R | | CAN controller 0 transmission-completed status |
| | | | 1* | Transmission was successfully completed |
| 15 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 | TBS1 | R | | CAN controller 1 transmit-buffer status |
| | | | 1* | Transmit buffers are empty |
| 8 | TBS0 | R | | CAN controller 0 transmit-buffer status |
| | | | 1* | Transmit buffers are empty |
| 7 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | TS1 | R | | CAN controller 1 transmit status |
| | | | 1* | A message is being transmitted |
| 0 | TS0 | R | | CAN controller 0 transmit status |
| | | | 1* | A message is being transmitted |

## 10.12 CAN controller central receive-status register

The CAN controller central receive-status register CCCRS provides bundled access to the reception status of all CAN controllers. The status flags are the same as those in the status register of the corresponding CAN controller.

The CCCRS register is read only. Table 21–297 shows the bit assignment of the CCCRS register.

**Table 297. CAN controller central receive-status register bit description**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 18 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 17 | DOS1[1] | R | | CAN controller 1 data-overrun status |
| | | | 1 | Received message was lost due to slow read-out of the preceding message |
| | | | 0* | |
| 16 | DOS0[1] | R | | CAN controller 0 data-overrun status |
| | | | 1 | Received message was lost due to slow read-out of the preceding message |
| | | | 0* | |
| 15 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 | RBS1[1] | R | | CAN controller 1 receive-buffer status |
| | | | 1 | Receive buffers contain a received message |
| | | | 0* | |
| 8 | RBS0[1] | R | | CAN controller 0 receive-buffer status |
| | | | 1 | Receive buffers contain a received message |
| | | | 0* | |
| 7 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | RS1 | R | | CAN controller 1 receive status |
| | | | 1* | A message is being received |
| 0 | RS0 | R | | CAN controller 0 receive status |
| | | | 1* | A message is being received |

[1] This bit is unchanged if a FullCAN message is received.

## 10.13 CAN controller central miscellaneous-status register

The CAN controller central miscellaneous-status register CCCMS provides bundled access to the bus and error status of all the CAN controllers. The status flags are the same as those in the status register of the corresponding CAN controller.

The CCCMS register is read only. Table 21–298 shows the bit assignment of the CCCMS register.

**Table 298. CAN controller central miscellaneous-status register bit description**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 | BS1 | R | | CAN controller 1 bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0* | |
| 8 | BS0 | R | | CAN controller 0 bus status |
| | | | 1 | The CAN controller is currently prohibited from bus activity because the transmit error counter has reached its limiting value of FFh |
| | | | 0* | |
| 7 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | ES1 | R | | CAN controller 1 error status |
| | | | 1 | The error warning limit has been exceeded |
| | | | 0* | |
| 0 | ES0 | R | | CAN controller 0 error status |
| | | | 1 | The error warning limit has been exceeded |
| | | | 0* | |

## 11. FullCAN mode

The FullCAN mode is based on capabilities provided by the CAN Gateway module used in the LPC2000 family of products. This block uses the Acceptance Filter to provide filtering for both CAN channels.

The concept of the CAN Gateway block is mainly based on a BasicCAN functionality. This concept fits perfectly in systems where a gateway is used to transfer messages or message data between different CAN channels. A BasicCAN device is generating a receive interrupt whenever a CAN message is accepted and received. Software has to move the received message out of the receive buffer from the according CAN controller into the user RAM.

To cover dashboard like applications where the controller typically receives data from several CAN channels for further processing, the CAN Gateway block was extended by a so-called FullCAN receive function. This additional feature uses an internal message handler to move received FullCAN messages from the receive buffer of the according CAN controller into the FullCAN message object data space of Look-up Table RAM.

When fullCAN mode is enabled, the Acceptance Filter itself takes care of receiving and storing messages for selected Standard ID values on selected CAN buses, in the style of “FullCAN” controllers.

In order to set this bit and use this mode, two other conditions must be met with respect to the contents of Acceptance Filter RAM and the pointers into it:

- The Standard Frame Individual Start Address Register (SFF_sa) must be greater than or equal to the number of IDs for which automatic receive storage is to be done, times two. SFF_sa must be rounded up to a multiple of 4 if necessary.
- The EndOfTable register must be less than or equal to 0x800 minus 6 times the SFF_sa value, to allow 12 bytes of message storage for each ID for which automatic receive storage will be done.

When these conditions are met and eFCAN is set:

- The area between the start of Acceptance Filter RAM and the SFF_sa address, is used for a table of individual Standard IDs and CAN Controller/bus identification, sorted in ascending order and in the same format as in the Individual Standard ID table. Entries can be marked as “disabled” as in the other Standard tables. If there are an odd number of “FullCAN” ID’s, at least one entry in this table must be so marked.
- The first (SFF_sa)/2 IDindex values are assigned to these automatically-stored ID’s. That is, IDindex values stored in the Rx Frame Status Register, for IDs not handled in this way, are increased by (SFF_sa)/2 compared to the values they would have when eFCAN is 0.
- When a Standard ID is received, the Acceptance Filter searches this table before the Standard Individual and Group tables.
- When a message is received for a controller and ID in this table, the Acceptance filter reads the received message out of the CAN controller and stores it in Acceptance Filter RAM, starting at (EndOfTable) + its IDindex*12.
- The format of such messages is shown in Table 21–299.

## 11.1 FullCAN message layout

**Table 299. Format of automatically stored Rx messages**

| Address | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | FF | RTR | 0000 | | | | SEM [1:0] | | 0000 | | | | DLC | | | | 00000 | | | | | ID.28 ... ID.18 | | | | | | | | | | |
| +4 | Rx Data 4 | | | | | | | | Rx Data 3 | | | | | | | | Rx Data 2 | | | | | | | | Rx Data 1 | | | | | | | |
| +8 | Rx Data 8 | | | | | | | | Rx Data 7 | | | | | | | | Rx Data 6 | | | | | | | | Rx Data 5 | | | | | | | |

The FF, RTR, and DLC fields are as described in Table 21–277.

Since the FullCAN message object section of the Look-up table RAM can be accessed both by the Acceptance Filter and the CPU, there is a method for insuring that no CPU reads from FullCAN message object occurs while the Acceptance Filter hardware is writing to that object.

For this purpose the Acceptance Filter uses a 3-state semaphore, encoded with the two semaphore bits SEM1 and SEM0 (see Table 21–299 “Format of automatically stored Rx messages”) for each message object. This mechanism provides the CPU with information about the current state of the Acceptance Filter activity in the FullCAN message object section.

The semaphore operates in the following manner:

**Table 300. FullCAN semaphore operation**

| SEM1 | SEM0 | activity |
|---|---|---|
| 0 | 1 | Acceptance Filter is updating the content |
| 1 | 1 | Acceptance Filter has finished updating the content |
| 0 | 0 | CPU is in process of reading from the Acceptance Filter |

Prior to writing the first data byte into a message object, the Acceptance Filter will write the FrameInfo byte into the according buffer location with SEM[1:0] = 01.

After having written the last data byte into the message object, the Acceptance Filter will update the semaphore bits by setting SEM[1:0] = 11.

Before reading a message object, the CPU should read SEM[1:0] to determine the current state of the Acceptance Filter activity therein. If SEM[1:0]  = 01, then the Acceptance Filter is currently active in this message object. If SEM[1:0] = 11, then the message object is available to be read.

Before the CPU begins reading from the message object, it should clear SEM[1:0] = 00.

When the CPU is finished reading, it can check SEM[1:0] again. At the time of this final check, if SEM[1:0] = 01 or 11, then the Acceptance Filter has updated the message object during the time when the CPU reads were taking place, and the CPU should discard the data. If, on the other hand, SEM[1:0] = 00 as expected, then valid data has been successfully read by the CPU.

Figure 21–76 shows how software should use the SEM field to ensure that all three words read from the message are all from the same received message.

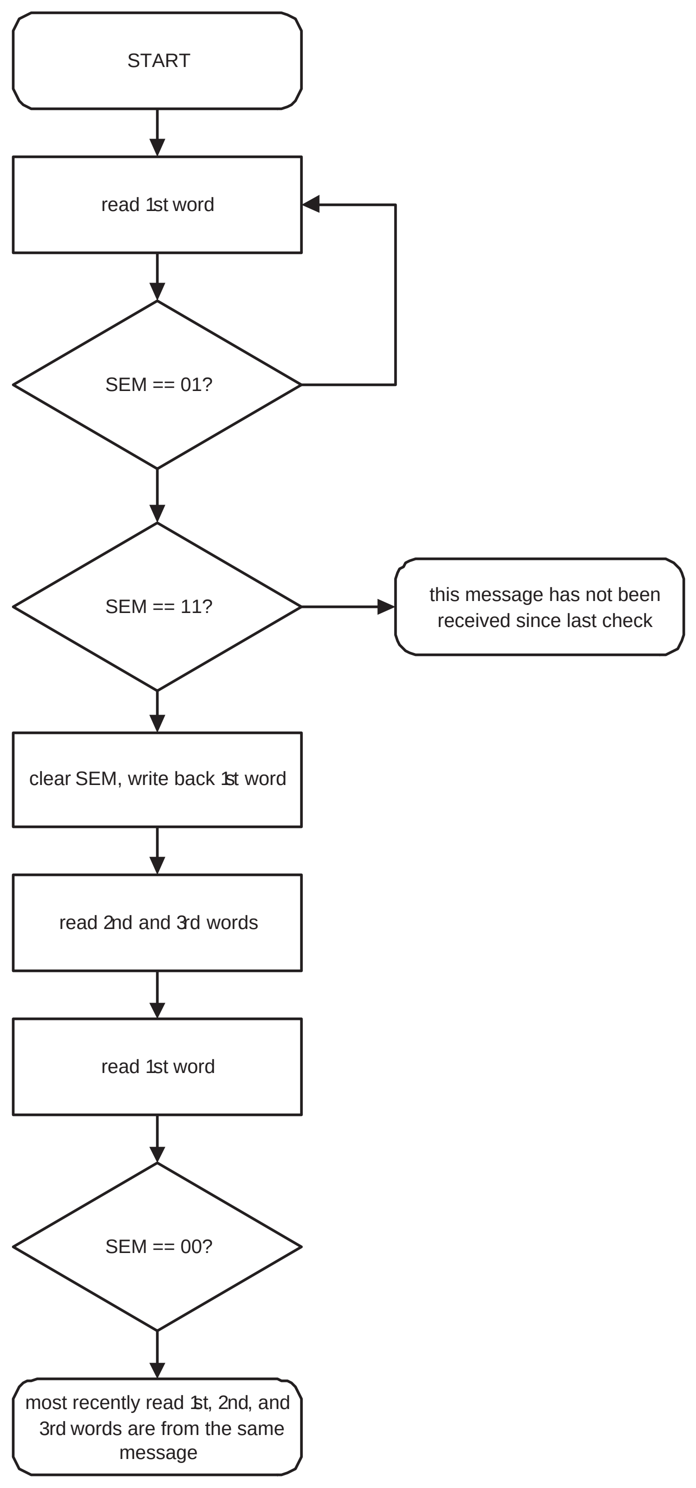

START
read 1st word
SEM == 01?
SEM == 11?
this message has not been received since last check
clear SEM, write back 1st word
read 2nd and 3rd words
read 1st word
SEM == 00?
most recently read 1st, 2nd, and 3rd words are from the same message

**Fig 76. Semaphore procedure for reading an auto-stored message**

## 11.2 FullCAN interrupts

The CAN Gateway Block contains a 2 kB ID Look-up Table RAM. With this size a maximum number of 146 FullCAN objects can be defined if the whole Look-up Table RAM is used for FullCAN objects only. Only the first 64 FullCAN objects can be configured to participate in the interrupt scheme. It is still possible to define more than 64 FullCAN objects. The only difference is, that the remaining FullCAN objects will not provide a FullCAN interrupt.

The FullCAN Interrupt Register-set contains interrupt flags (IntPndx) for (pending) FullCAN receive interrupts. As soon as a FullCAN message is received, the according interrupt bit (IntPndx) in the FCAN Interrupt Register gets asserted. In case that the Global FullCAN Interrupt Enable bit is set, the FullCAN Receive Interrupt is passed to the Vectored Interrupt Controller.

Application Software has to solve the following:

1. Index/Object number calculation based on the bit position in the FCANIC Interrupt Register for more than one pending interrupt.
2. Interrupt priority handling if more than one FullCAN receive interrupt is pending.

The software that covers the interrupt priority handling has to assign a receive interrupt priority to every FullCAN object. If more than one interrupt is pending, then the software has to decide, which received FullCAN object has to be served next.

To each FullCAN object a new FullCAN Interrupt Enable bit (FCANIntxEn) is added, so that it is possible to enable or disable FullCAN interrupts for each object individually. The new Message Lost flag (MsgLstx) is introduced to indicate whether more than one FullCAN message has been received since last time this message object was read by the CPU. The Interrupt Enable and the Message Lost bits reside in the existing Look-up Table RAM.

### 11.2.1 FullCAN message interrupt enable bit

In Figure 21–77 8 FullCAN Identifiers with their Source CAN Channel are defined in the FullCAN, Section. The new introduced FullCAN Message Interrupt enable bit can be used to enable for each FullCAN message an Interrupt.

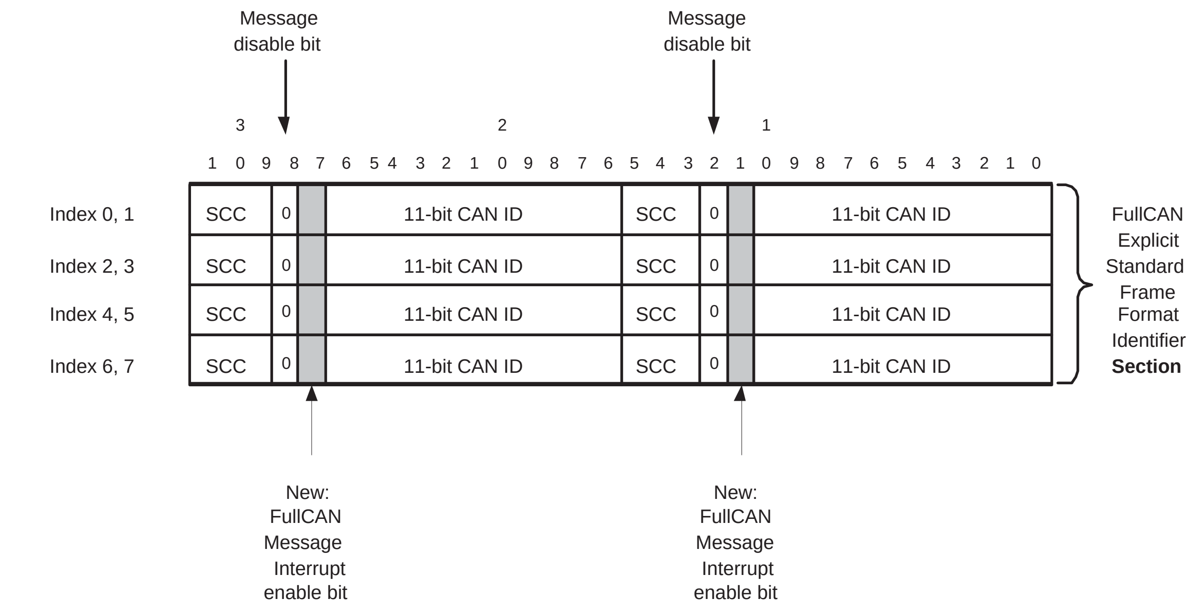


**Fig 77. FullCAN section example of the ID look-up table**

### 11.2.2 Message lost bit and CAN channel number

Figure 21–78 is the detailed layout structure of one FullCAN message stored in the FullCAN message object section of the Look-up Table.

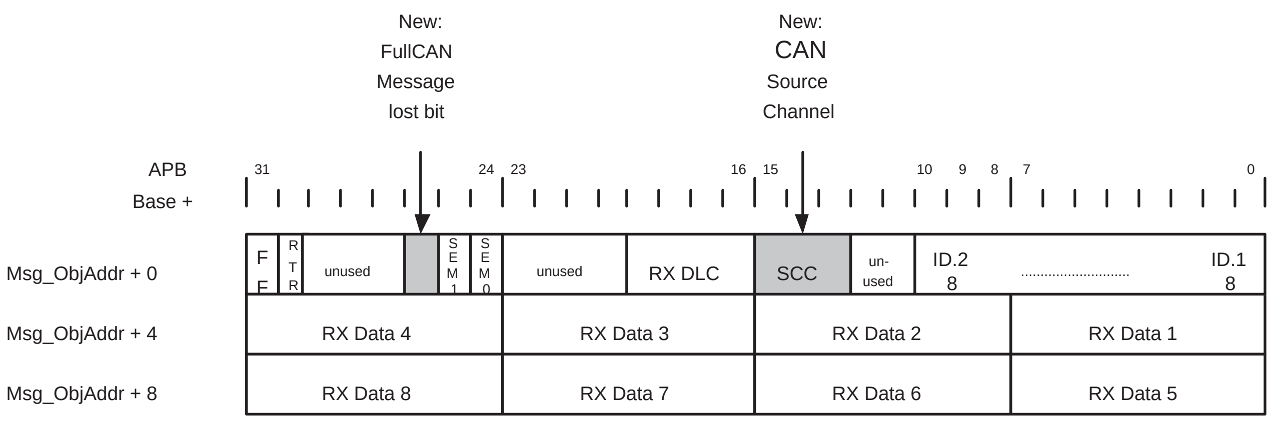


**Fig 78. FullCAN message object layout**

The new message lost bit (MsgLst) is introduced to indicate whether more than one FullCAN message has been received since last time this message object was read. For more information the CAN Source Channel (SCC) of the received FullCAN message is added to Message Object.

### 11.2.3 Setting the interrupt pending bits (IntPnd 63 to 0)

The interrupt pending bit (IntPndx) gets asserted in case of an accepted FullCAN message and if the interrupt of the according FullCAN Object is enabled (enable bit FCANIntxEn) is set).

During the **last write access** from the data storage of a FullCAN message object the interrupt pending bit of a FullCAN object (IntPndx) gets asserted.

### 11.2.4 Clearing the interrupt pending bits (IntPnd 63 to 0)

Each of the FullCAN Interrupt Pending requests gets cleared when the semaphore bits of a message object are cleared by Software (ARM CPU).

### 11.2.5 Setting the message lost bit of a FullCAN message object (MsgLost 63 to 0)

The Message Lost bit of a FullCAN message object gets asserted in case of an accepted FullCAN message and when the FullCAN Interrupt of the same object is asserted already.

During the **first write access** from the data storage of a FullCAN message object the Message Lost bit of a FullCAN object (MsgLostx) gets asserted if the interrupt pending bit is set already.

### 11.2.6 Clearing the message lost bit of a FullCAN message object (MsgLost 63 to 0)

The Message Lost bit of a FullCAN message object gets cleared when the FullCAN Interrupt of the same object is not asserted.

During the **first write access** from the data storage of a FullCAN message object the Message Lost bit of a FullCAN object (MsgLostx) gets cleared if the interrupt pending bit is not set.

## 11.3 Set and clear mechanism of the FullCAN interrupt

Special precaution is needed for the built-in set and clear mechanism of the FullCAN Interrupts. The following text illustrates how the already existing Semaphore Bits (see Section 21–11.1 “FullCAN message layout” for more details) and how the new introduced features (IntPndx, MsgLstx) will behave.

### 11.3.1 Scenario 1: Normal case, no message lost

Figure 21–79 below shows a typical “normal” scenario in which an accepted FullCAN message is stored in the FullCAN Message Object Section. After storage the message is read out by Software (ARM CPU).

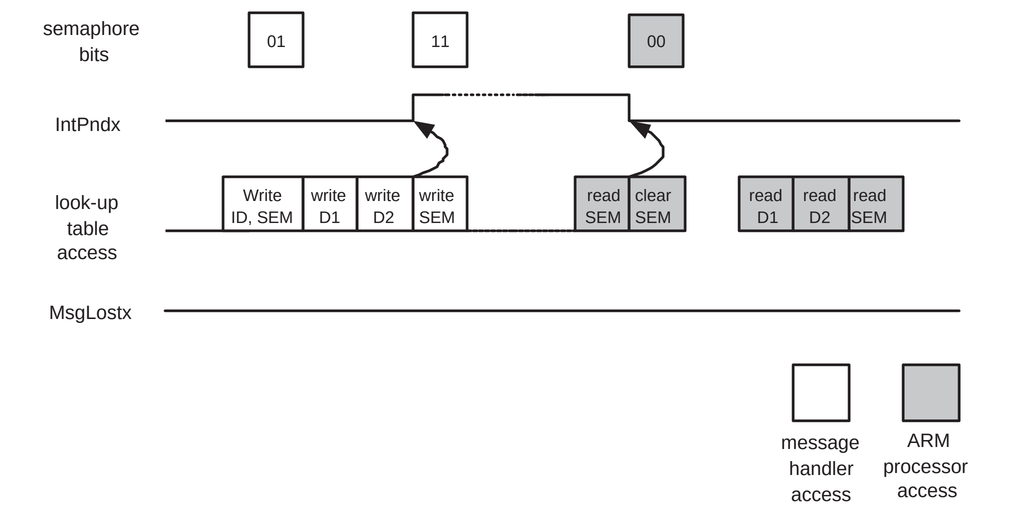


Fig 79. Normal case, no messages lost

### 11.3.2 Scenario 2: Message lost

In this scenario a first FullCAN Message is stored and read out by Software (1st Object write and read). In a second course a second message is stored (2nd Object write) but not read out before a third message gets stored (3rd Object write). Since the FullCAN Interrupt of that Object (IntPndx) is already asserted, the Message Lost Signal gets asserted.

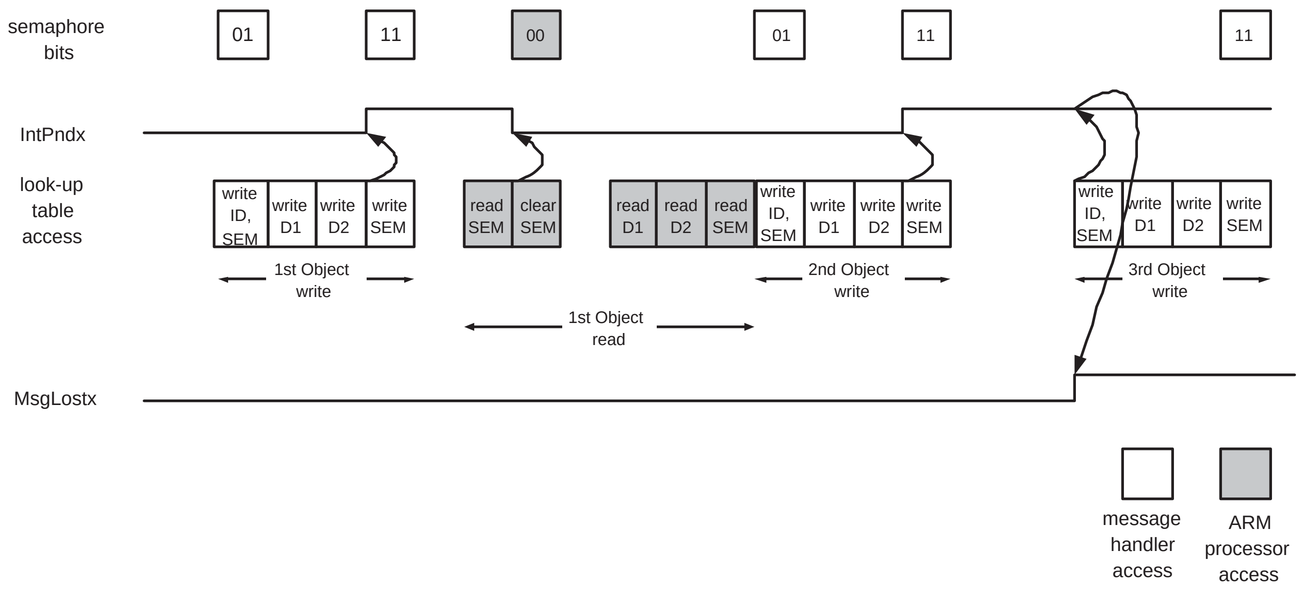


Fig 80. Message lost

### 11.3.3 Scenario 3: Message gets overwritten indicated by Semaphore bits

This scenario is a special case in which the lost message is indicated by the existing semaphore bits. The scenario is entered, if during a Software read of a message object another new message gets stored by the message handler. In this case, the FullCAN Interrupt bit gets set for a second time with the 2nd Object write.

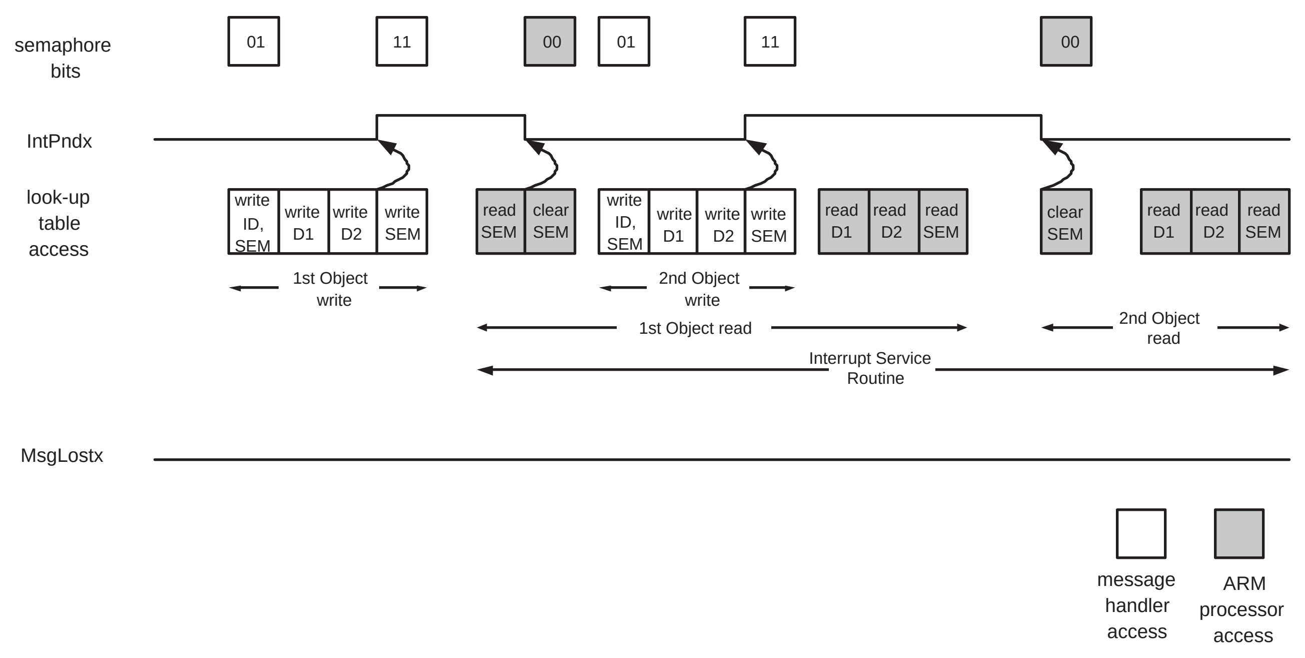


Fig 81. Message gets overwritten

### 11.3.4 Scenario 3.1: Message gets overwritten indicated by Semaphore bits and Message Lost

This scenario is a sub-case to Scenario 3 in which the lost message is indicated by the existing semaphore bits and by Message Lost.

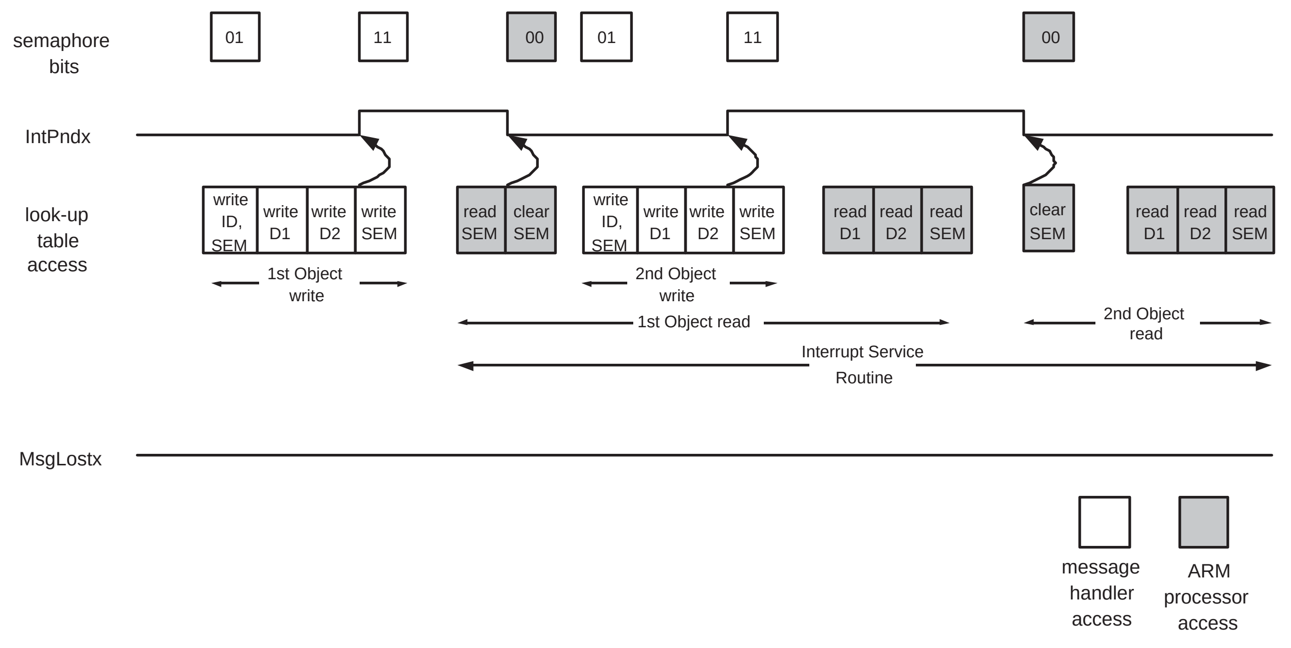


**Fig 82. Message overwritten indicated by semaphore bits and message lost**

## 11.3.5 Scenario 3.2: Message gets overwritten indicated by Message Lost

This scenario is a sub-case to Scenario 3 in which the lost message is indicated by Message Lost.

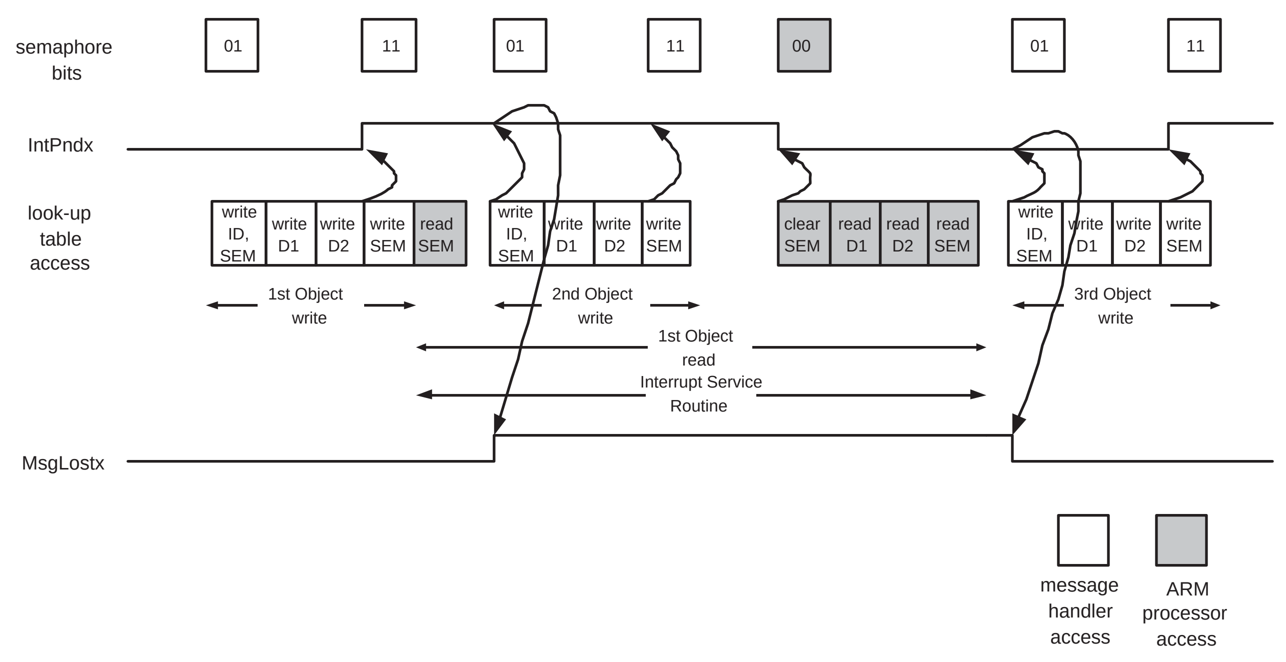


Fig 83. Message overwritten indicated by message lost

## 11.3.6 Scenario 4: Clearing Message Lost bit

This scenario is a special case in which the lost message bit of an object gets set during an overwrite of a none read message object (2nd Object write). The subsequent read out of that object by Software (1st Object read) clears the pending Interrupt. The 3rd Object write clears the Message Lost bit. Every “write ID, SEM” clears Message Lost bit if no pending Interrupt of that object is set.

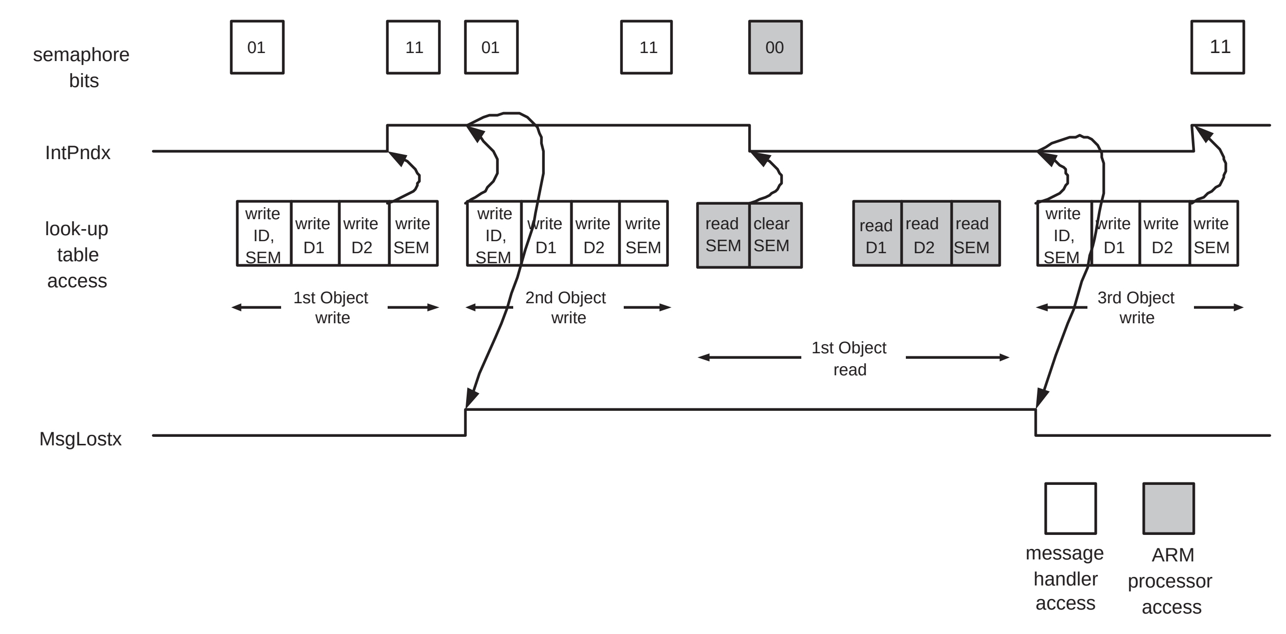


Fig 84. Clearing message lost

## 12. CAN configuration example 1

Table 21–301 shows which sections and types of CAN identifiers are used and activated. The ID look-up table configuration of this example is shown in Figure 21–85.

Table 301. Used ID look-up table sections of example 1

| ID look-up table section | Usage |
|---|---|
| FullCAN | Not Activated |
| Explicit standard frame-format | Activated |
| Group of standard frame-format | Activated |
| Explicit extended frame-format | Activated |
| Group of extended frame-format | Activated |

### 12.1 Explicit standard-frame format identifier section (11-bit CAN ID)

The start address of the explicit standard frame-format section is defined in the CASFESA register with a value of 00h. The end of this section is defined in the CASFGSA register.

In the explicit standard frame-format section of the ID look-up table, two CAN identifiers with their source CAN channels (SCCs) share one 32-bit word. Unused or disabled CAN identifiers can be marked by setting the message-disable bit.

To provide memory space for eight explicit standard frame-format identifiers, the CASFGSA register value is set to 10h. The identifier with Index 7 of this section is not used and is therefore disabled.

## 12.2 Group of standard frame-format identifier section (11-bit CAN ID)

The start address of the group of the standard frame-format section is defined in the CASFGSA register with a value of 10h. The end of this section is defined in the CAEFESA register.

In the group of standard frame-format sections, two CAN Identifiers with the same SCC share one 32-bit word and represent a range of CAN Identifiers to be accepted. Bits 31 down to 16 represent the lower boundary and bits 15 down to 0 represent the upper boundary of the range of CAN Identifiers. All identifiers within this range (including the boundary identifiers) are accepted. A whole group can be disabled and not used by the acceptance filter by setting the message-disable bit in the upper and lower boundary identifiers.

To provide memory space for four Groups of standard frame-format identifiers the CAEFESA register value is set to 20h. The identifier group with Index 9 of this section is not used and is therefore disabled.

## 12.3 Explicit standard frame-format identifier section (29-bit CAN ID)

The start address of the explicit extended frame-format section is defined in the CAEFESA register with a value of 20h. The end of this section is defined in the CAEFGSA register.

In the explicit extended frame-format section, only one CAN Identifier with its SCC is programmed per address line.

To provide memory space for four explicit extended frame-format identifiers the CAEFGSA register value is set to 30h.

## 12.4 Group of extended frame-format identifier section (29-bit CAN ID)

The start address of the extended frame-format group is defined by the CAEFGSA register with a value of 30h. The end of this section is defined by the end-of-table address register CAEOTA.

In the extended frame-format section group boundaries are programmed with a pair of address lines. The first is the lower boundary, the second the upper boundary.

To provide memory space for two groups of extended frame-format Identifiers the CAEOTA register value is set to 40h.

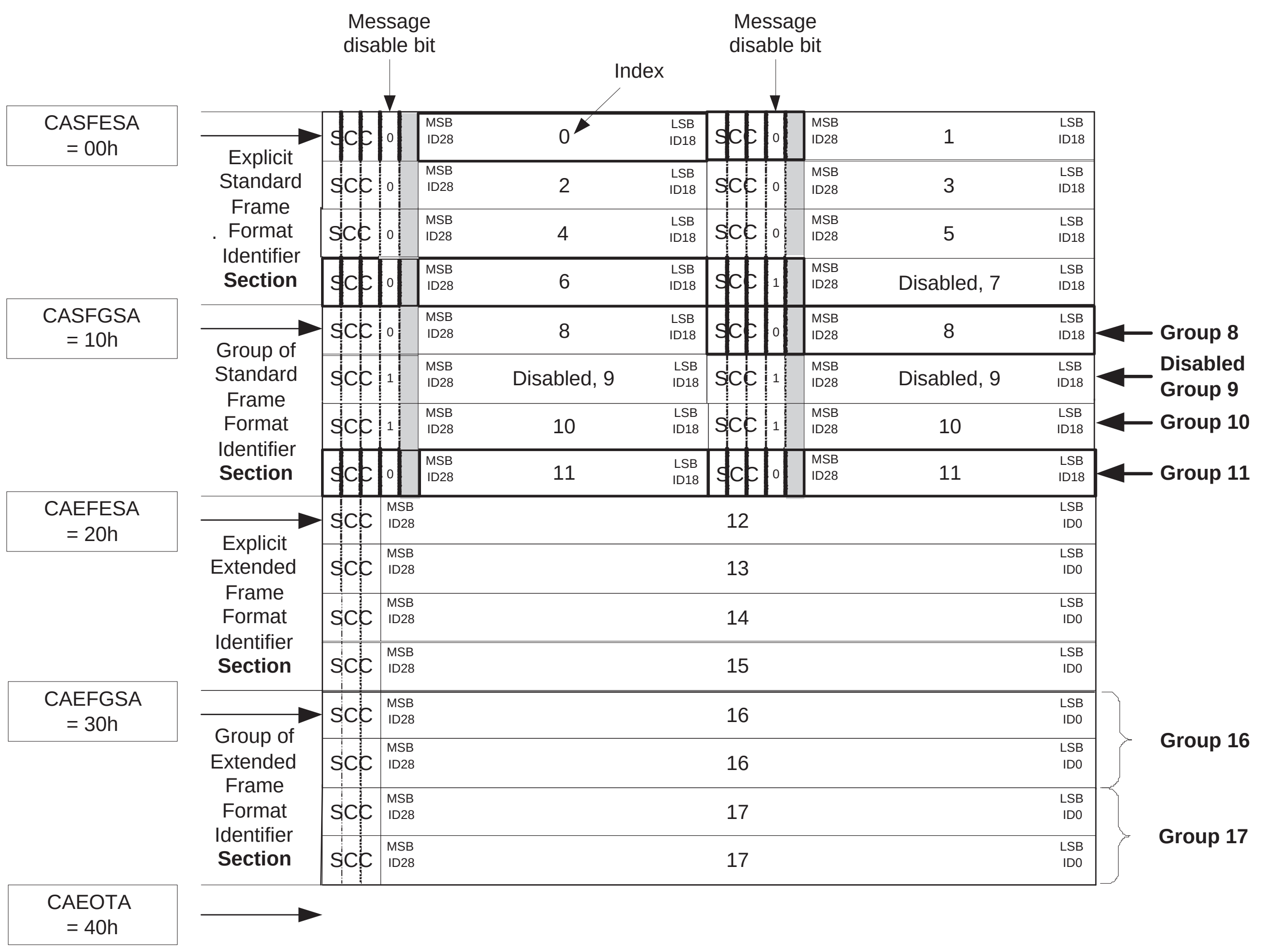


Fig 85. ID-look-up table, configuration example 1

## 13. CAN configuration example 2

Table 21–302 shows which sections and types of CAN identifiers are used and activated. The ID look-up table configuration of this example is shown in Figure 21–86.

This example uses a typical configuration in which FullCAN as well as explicit standard frame-format messages are defined. As described in Section 21–8.10, acceptance filtering takes place in a certain order. With the FullCAN section enabled, the identifier-screening process of the acceptance filter always starts in the FullCAN section before continuing with the rest of the enabled sections.

Table 302. Used ID look-up table sections of example 2

| ID-look-up table section | Usage |
|---|---|
| FullCAN | Activated and enabled |
| Explicit standard frame format | Activated |

Table 302. Used ID look-up table sections of example 2

| ID-look-up table section | Usage |
|---|---|
| Group of standard frame format | Not Activated |
| Explicit extended frame format | Not Activated |
| Group of extended frame format | Not Activated |

## 13.1 FullCAN explicit standard frame-format section (11-bit CAN ID)

The start address of the FullCAN explicit standard frame-format section is automatically set to 00h. The end of this section is defined in the CASFESA register.

In the FullCAN ID section, only FullCAN object identifiers are stored for acceptance filtering. In this section two CAN Identifiers with their SCCs share one 32-bit word. Unused or disabled CAN Identifiers can be marked by setting the message-disable bit. The FullCAN object data for each defined identifier can be found in the FullCAN message object section. In the event of an identifier match during the acceptance filter process, the received FullCAN message-object data is moved from the receive buffer of the appropriate CAN controller into the FullCAN message-object section.

To provide memory space for eight FullCAN explicit standard frame-format identifiers the CASFESA register value is set to 10h. Identifier index 1 of this section is not used and is therefore disabled.

## 13.2 Explicit standard frame-format section (11-bit CAN ID)

The start address of the explicit standard frame-format section is defined in the CASFESA register with a value of 10h. The end of this section is defined in the end-of-table address register CAEOTA.

In the explicit standard frame-format section of the ID look-up table, two CAN Identifiers with their SCCs share one 32-bit word. Unused or disabled CAN Identifiers can be marked by setting the message-disable bit.

To provide memory space for eight explicit standard frame-format identifiers the EOTA register value is set to 20h.

## 13.3 FullCAN message-object data section

The start address of the FullCAN message-object data section is defined in the EOTA register. The number of enabled FullCAN identifiers is limited to the available memory space in the data section.

Each defined FullCAN message needs three address lines in the data section for the message data.

The FullCAN message-object section is organized so that each index number in the FullCAN identifier section corresponds to a message-object number in the message-object section.

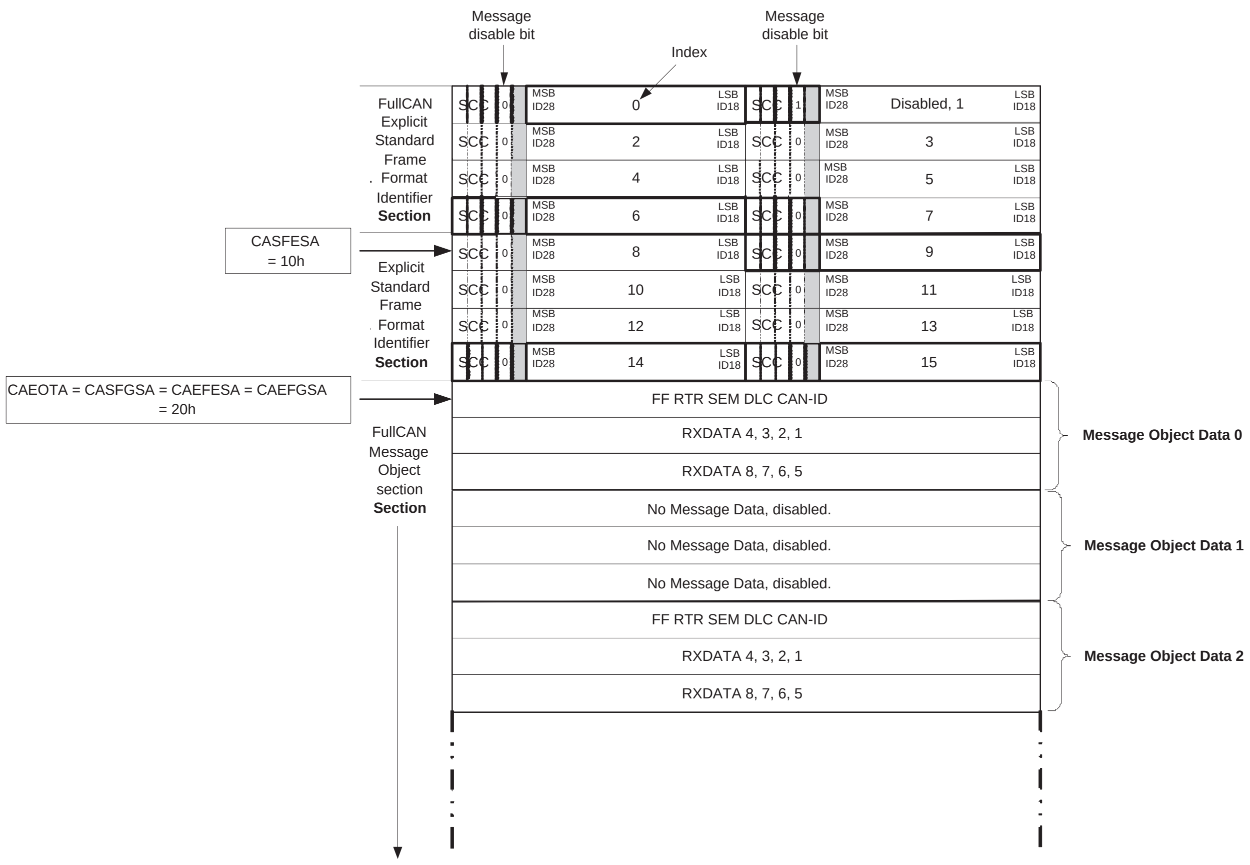


**Fig 86. ID look-up table, configuration example 2**

## 14. CAN look-up table programming guidelines

All identifier sections of the ID look-up table must be programmed so that each active section is a sorted list or table with the source CAN channels (SCCs)in ascending order, together with CAN Identifier in each section.

Where a syntax error is encountered in the ID look-up table the address of the incorrect line will be available in the look-up table error address Register CALUTEA.

The reporting process in the CALUTEA register is a run-time process. Address lines with syntax errors are reported only if they have passed through the acceptance-filtering process.

General rules for programming the look-up table are as follows:

- Each section must be organized as a sorted list or table, with the SCCs in ascending order and in conjunction with the CAN Identifier. There is no exception for disabled identifiers.
- The upper and lower bound in an identifier-group definition has to be from the same SCC.

- To disable a group of identifiers the message-disable bit must be set for both the upper and lower bounds.

# UM10316

## Chapter 22: LPC29xx LIN/UART 0/1

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The LIN/UART block is identical for all LPC29xx parts.

## 2. Introduction

The LPC29xx LIN controller can function as UART or LIN interface and consists of two independent sub controllers for UART and LIN functions and a fractional baud rate generator for both. A set of common registers is used to select and configure UART and LIN modes and to control the fractional baud rate controller. UART and LIN specific registers share the same address space but are only visible in UART or LIN mode, respectively. UART and LIN use a common set of receive and transmit lines.

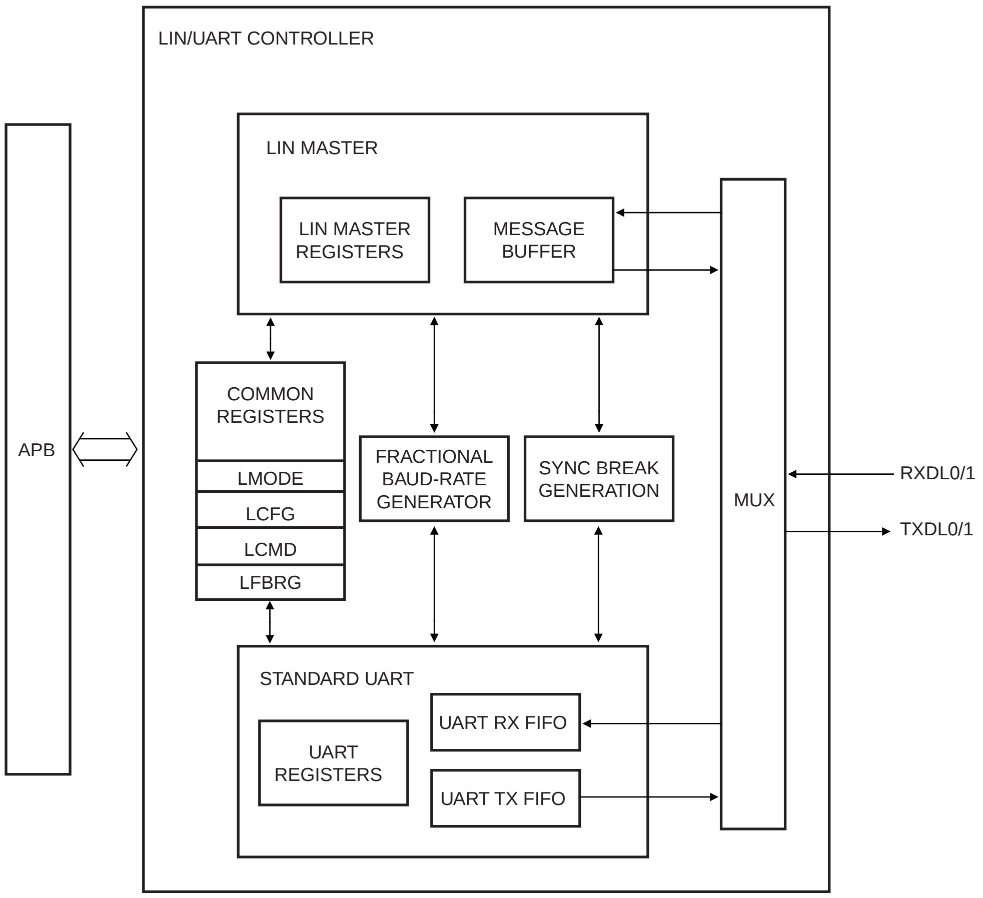


**Fig 87. LIN/UART block diagram**

## 3. LIN features

- Complete LIN 2.0 message handling and transfer

- One interrupt per LIN message
- Slave response time-out detection
- Programmable sync-break length
- Automatic sync-field and sync-break generation
- Programmable inter-byte space
- Hardware or software parity generation
- Automatic checksum generation
- Fault confinement
- Fractional baud rate generator

## 4. UART 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.
- Fractional divider for baud rate control and mechanism that enables software flow control implementation.

## 5. LIN functional description

Each LIN master controller can be used as a dedicated LIN master with additional support for sync-break generation. Figure 22–88 gives a brief overview of the LIN master controller and shows the shared hardware used from the LIN master.

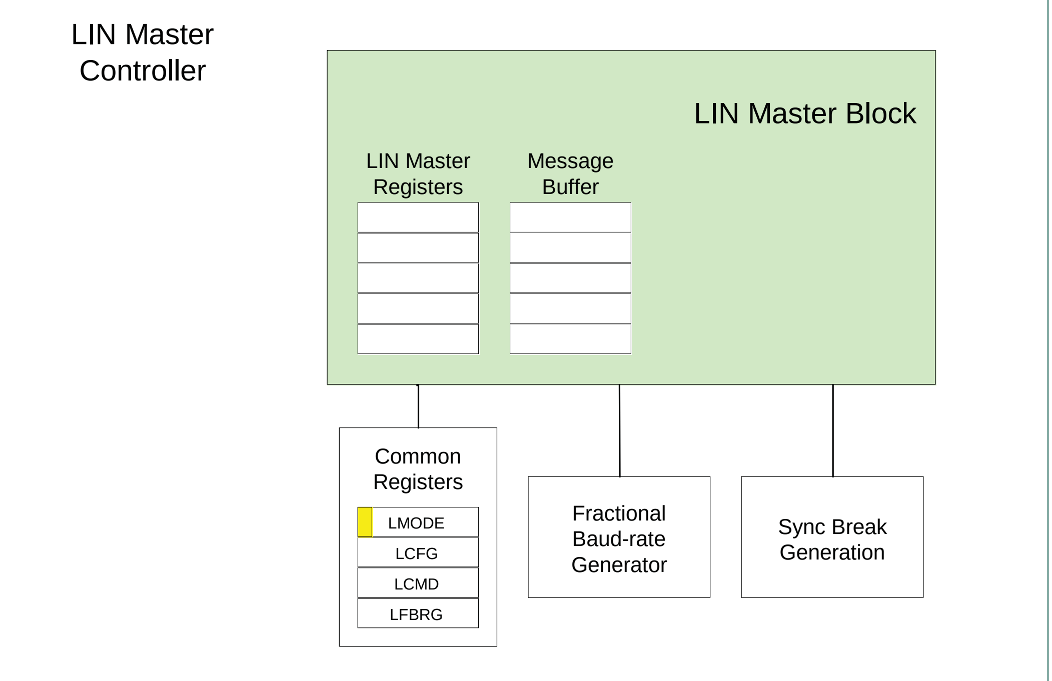


**Fig 88. LIN master controller block diagram**

## 5.1 LIN master

The LIN master controller can send complete message frames without interrupting the CPU. Generation of a new message frame is always initiated by a transmission-request command. This LIN master command forces the LIN master to send the LIN header field including synch break, synch field and a user-specified ID field. According to the LIN specification all fields are sent with LSB first.

Depending on the specified data direction, the LIN master then either continues to send data fields or waits for data from an external slave node.

When the LIN master is sending response fields (master sending; slave receiving), the specified number of data fields stored in the message buffer is transmitted automatically. The checksum field is generated and sent after the data fields.

When the LIN master is expecting response fields (slave sending: master receiving), the received data fields are stored within the message buffer. In this case the checksum is calculated and compared with the received checksum field.

At the end of the message frame either a TX message-complete or an RX message-complete condition is generated for the user. Message-complete conditions are signaled either via the status register or by message-complete Interrupts.

Error detection takes place during the whole message frame. As soon as an error condition is detected, the message frame is aborted at the end of the current field. Error conditions are signaled either via the status register or by error interrupts.

### 5.1.1 LIN sync-break generation

The LIN master controller design offers an easy method for sync-break field generation.

As shown in Figure 22–89, the synchronization break field consists of two different parts. The first part is a dominant bus value with a duration of TSYNBRK: the second part is a recessive synchronization delimiter with a minimum duration of TSYNDEL.

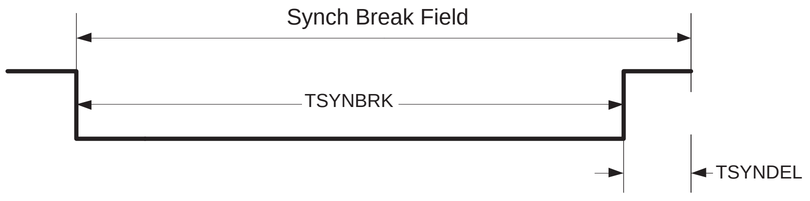


**Fig 89. Synch-break field**

The length of the sync-break field is programmable in the range TSYNBRK = 10 to 16 bit. It is defined with the SBL bits in the configuration register LCFG.

The sync-break field is automatically sent out with the message frame. When the TR bit in the LCMD register is set, transmission of a complete LIN message is initiated.

### 5.1.2 Registers and mapping

The complete register layout of the LIN master controller is shown in Ref. 31–6. Refer to this for resolving register, register-slice and bit names.

### 5.1.3 Error detection

The LIN master Controller contains error-detection logic which can detect the following wake-up or error conditions:

- Wake-up/LIN protocol error
- Bit errors
- RXD/TXD line-clamped errors

All wake-up or error conditions can be enabled as interrupts.

Bit-errors and RXD/TXD line-clamped errors can only be detected when the LIN master is actively transmitting. The only exception to this is that during reception of slave responses stop-bit errors can also be detected.

In cases where bit errors are detected, the status of the bit error is signalled at the end of the field in which it occurred and further transmission is then aborted.

Table 22–303 shows in more detail when and under what conditions a bit error, an RXD/TXD line-clamped error or a wake-up/LIN protocol error can occur.

**Table 303. Error conditions and detection**

| Cause of error occurrence | Condition | Error description | LIN master | |
|---|---|---|---|---|
| | | | Interrupt flags | Error capture code |
| During LIN bus idle | Dominant level on RXD pin, RXD=0 | RXD/TXD stuck dominant | WPI | |
| | | Wake-up/Protocol error | | |
| Master is sending; HS=1 or TS=1 | | | | |
| During every LIN field | No falling edge (start bit) on RXD pin detected and RXD is recessive | RXD/TXD stuck recessive | RTLCEI | 0x1000 |
| | No falling edge (start bit) on RXD pin detected and RXD is dominant | RXD/TXD stuck dominant | RTLCEI | 0x1001 |
| | LIN_RSR <> LIN_TSR | Bit error(s) | BEI | |
| Master is receiving; RS=1 | | | | |
| During response fields | Stop bit = 0b | Dominant level during stop bit | BEI | |

### 5.1.4 Line-clamped detection versus bit-error detection

Depending on the situation when a line-clamped error is detected, it can be difficult to distinguish between a line-clamped and a bit error. A typical situation could be that during transmission of a LIN field the RXD or TXD line gets clamped permanently. In this case a bit error will be detected first for this field since the differences between the transmitted and received bits lead to this conclusion.

The LIN master aborts message transmission at the end of a field where a bit error was detected.

To safely distinguish between a bit error and a line clamped error, the LIN master should send a second message as soon as a bit error is detected. With the second message the LIN master will be able to distinguish clearly between bit errors and line-clamped errors.

### 5.1.5 Wake-up interrupt handling

According to the LIN specification, any node in a sleeping LIN cluster may request a wake-up. The wake-up request is issued by forcing the bus to dominant state for a period of between 250 µs and 5 ms. When a LIN slave requests a wake-up by issuing a dominant state the LIN master wake-up interrupt is asserted at the beginning of the dominant state. The wake-up interrupt service routine should be written so that the wake-up response frame from the LIN Master is not sent immediately. To give a slave-ready time the LIN master has to wait for about 100 ms before sending the wake-up response frame (according to the LIN specification, see Ref. 31–6), or at least for the time defined in the slave's node-capability file.

### 5.1.6 Slave-not-responding error and the LIN master time-out register

The LIN master time-out register defines the maximum number of bit times ($T_{Bit}$) that may elapse until the responses from all LIN slaves to the master have been completed. The time-out starts as soon as the LIN header is transmitted (the value of the time-out register is decremented with every bit time) and a slave response is expected. When enabled, the slave-not-responding error interrupt NRI is asserted as soon as the time-out limit is exceeded.

## 6. Pin description

The UART and LIN interfaces share the RX and TX pins.

**Table 304. LIN/UART0/1 pin description**

| Pin | Type | Description |
|---|---|---|
| RXDL0/1 | Input | **Serial Input.** Serial receive data in LIN and UART mode. |
| TXDL0/1 | Output | **Serial Output.** Serial transmit data in LIN and UART mode. |

## 7. Register overview

The register space of the LIN/UART controller consists of a block of common registers and two sets of registers for the LIN and UART block respectively, which share the same address space. LIN registers are visible when the MODE bit in the LMODE register (Table 22–306) is set to zero (this is the default), UART registers are visible when the MODE bit in the LMODE register is set to one.

**Table 305. Register overview: LIN/UART0/1 (base address: 0xE008 9000 (LIN/UART0), 0xE008 A000 (LIN/UART1))**

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| **Common LIN/UART registers** | | | | | |
| LMODE | R/W | 0x00 | LIN/UART master-controller mode register | 0x01 | see Table 22–306 |
| LCFG | R/W | 0x04 | LIN/UART master-controller configuration register | 0x00 | see Table 22–307 |
| LCMD | R/W | 0x08 | LIN/UART master-controller command register | 0x00 | see Table 22–308 |

Table 305. Register overview: LIN/UART0/1 (base address: 0xE008 9000 (LIN/UART0), 0xE008 A000 (LIN/UART1))

| Name | Access | Address offset | Description | Reset value | Reference |
|---|---|---|---|---|---|
| LFBRG | R/W | 0x0C | LIN/UART master-controller fractional baud-rate generator register | 0x0 0001 | see Table 22–309 |
| **LIN master registers (LMODE = 0)** | | | | | |
| LSTAT | R | 0x10 | LIN master-controller status register | 0x342 | see Table 22–310 |
| LIC | R | 0x14 | LIN master-controller interrupt and capture register | 0x000 | see Table 22–311 |
| LIE | R/W | 0x18 | LIN master-controller interrupt-enable register | 0x10 | see Table 22–312 |
| - | R | 0x1C | Reserved for future expansion | - | - |
| LCS | R/W | 0x20 | LIN master-controller checksum register | 0x00 | see Table 22–313 |
| LTO | R/W | 0x24 | LIN master-controller time-out register | 0x00 | see Table 22–314 |
| LID | R/W | 0x28 | LIN master-controller message buffer identifier register | 0x0000 0000 | see Table 22–315 |
| LDATA | R/W | 0x2C | LIN master-controller message buffer data A register | 0x0000 0000 | see Table 22–316 |
| LDATB | R/W | 0x30 | LIN master-controller message buffer data B register | 0x0000 0000 | see Table 22–317 |
| LDATC | R/W | 0x34 | LIN master-controller message buffer data C register | 0x0000 0000 | see Table 22–318 |
| LDATD | R/W | 0x38 | LIN master-controller message buffer data D register | 0x0000 0000 | see Table 22–319 |
| **UART registers (LMODE = 1)** | | | | | |
| RBR | RO | 0x10 | UART Receiver Buffer Register | NA | see Table 22–320 |
| THR | WO | 0x10 | UART Transmit Holding Register | NA | see Table 22–321 |
| IER | R/W | 0x14 | UART Interrupt Enable Register | 0x00 | see Table 22–322 |
| IIR | RO | 0x18 | UART Interrupt ID Register | 0x01 | see Table 22–323 |
| FCR | WO | 0x18 | UART FIFO Control Register | 0x00 | see Table 22–325 |
| LCR | R/W | 0x1C | UART Line Control Register | 0x00 | see Table 22–326 |
| - | - | 0x20 | Reserved | - | - |
| LSR | RO | 0x24 | UART Line Status Register | 0x60 | see Table 22–327 |
| - | - | 0x28 | Reserved | - | - |
| SCR | R/W | 0x2C | UART Scratch Pad Register | 0x00 | see Table 22–328 |
| - | - | 0x34 - 0x3C | Reserved | - | - |
| TER | R/W | 0x40 | UART Transmit Enable Register | 0x80 | see Table 22–329 |

## 7.1 Common LIN/UART registers

### 7.1.1 LIN master-controller mode register

The register LMODE contains the software reset control for the LIN controller.

Table 22–306 shows the bit assignment of the LMODE register.

**Table 306. LIN master-controller mode register bit description (LIN0_LMODE, address 0xE008 9000, LIN1_LMODE, address 0xE008 A000)**
** = reset value*

| Bit | Symbol | Access | | Value | Description |
|---|---|---|---|---|---|
| | | LIN mode | UART mode | | |
| 31 to 8 | reserved | R | - | - | Reserved; do not modify. Read as logic 0 |
| 7 | MODE | R/W | R/W | | LIN master/UART mode |
| | | | | 1 | the LIN controller is in UART mode |
| | | | | 0*[1] | the LIN master controller is in LIN mode |
| 6 to 1 | reserved | R | - | - | Reserved; do not modify. Read as logic 0 |
| 0 | LRM | R/W | R/W | | LIN reset mode; only writable in LIN master-controller mode |
| | | | | 1* | the LIN master controller is in reset mode and the current message transmission or reception is aborted. The registers LCMD, LSTAT, LIC, LCS, LID, LDATA, LDATB, LDATC and LDATD get their reset value |
| | | | | 0 | the LIN master controller is in normal operation mode |

[1] On reset the LIN controller is in LIN mode. Changing into UART mode is only possible when reset mode (bit LRM) was entered in a previous command.

## 7.1.2 LIN master-controller configuration register

The LIN master-controller configuration register LCFG is used to change the length of the sync-break field and the inter-byte space, and also contains software-enable bits for the identifier parity and checksum calculations. Depending on the selected mode certain bits are not writable, but all bits are always readable.

Table 22–307 shows the bit assignment of the LCFG register.

**Table 307. LIN master-controller configuration register bit description (LIN0_LCFG, address 0xE008 9004, LIN1_LCFG, address 0xE008 A004)**
** = reset value*

| Bit | Symbol | Access | | Value | Description |
|---|---|---|---|---|---|
| | | LIN mode | UART mode | | |
| 31 to 8 | reserved | R | - | - | Reserved; do not modify. Read as logic 0. |
| 7 | SWPA | R/W | R | | Software ID parity |
| | | | | 1 | Software-generated ID parity from the message buffer is used to send onto the LIN bus. |
| | | | | 0* | Only the hardware-generated parity is used to send onto the LIN bus. |
| 6 | SWCS | R/W | R | | Software checksum |
| | | | | 1 | Checksum is generated by software. |
| | | | | 0* | Checksum is generated by hardware. |
| 5 | reserved | R | - | - | Reserved; do not modify. Read as logic 0. |

**Table 307. LIN master-controller configuration register bit description (LIN0_LCFG, address 0xE008 9004, LIN1_LCFG, address 0xE008 A004)** *…continued*
** = reset value*

| Bit | Symbol | Access | | Value | Description |
|---|---|---|---|---|---|
| | | LIN mode | UART mode | | |
| 4 and 3 | IBS[1:0] | R/W | R | | Inter-byte space length. This is inserted during transmission. |
| | | | | 00* | 0 bits inter-byte space length |
| | | | | 01 | 1 bit inter-byte space length |
| | | | | 10 | 2 bits inter-byte space length |
| | | | | 11 | 3 bits inter-byte space length |
| 2 to 0 | SBL[2:0] | R/W | R/W | | Synch-break logic 0 length. Writing a value of 0x7 will always read as 0x6. |
| | | | | 000* | 10 bits sync-break length |
| | | | | 001 | 11 bits sync-break length |
| | | | | 010 | 12 bits sync-break length |
| | | | | 011 | 13 bits sync-break length |
| | | | | 100 | 14 bits sync-break length |
| | | | | 101 | 15 bits sync-break length |
| | | | | 110 | 16 bits sync-break length |
| | | | | 111 | 17 bits sync-break length |

## 7.1.3 LIN master-controller command register

The LIN master-controller command register LCMD initiates a LIN message transmission.

Table 22–308 shows the bit assignment of the LCMD register.

**Table 308. LIN master-controller command register bit description (LIN0_LCMD, address 0xE008 9008, LIN1_LCMD, address 0xE008 A008)**
** = reset value*

| Bit | Symbol | Access | | Value | Description |
|---|---|---|---|---|---|
| | | LIN mode | UART mode | | |
| 31 to 8 | reserved | R | - | - | Reserved; do not modify. Read as logic 0 |
| 7 | SSB | R/W | R/W | | Send sync break |
| | | | | 1 | A sync break is sent onto the LIN bus. This bit is cleared automatically |
| | | | | 0* | |
| 6 to 1 | reserved | R | - | - | Reserved; do not modify. Read as logic 0 |
| 0 | TR | R/W | R | | Transmit request |
| | | | | 1 | Transmission of a complete LIN message will be initiated. This bit is cleared automatically |
| | | | | 0* | |

### 7.1.4 LIN master-controller fractional baud rate generator register

The LIN master-controller fractional baud rate generator register LFBRG stores the divisor in 16-bit binary format and the fraction in 4-bit binary format for the programmable baud-rate generator. The output frequency of the baud-rate generator is 16 times the baud rate. The input frequency of the baud generator is the BASE_IVNSS_CLK frequency (branch clock CLK_IVNSS_LIN) $\times$ $f_{CLK(LIN)}$ divided by the divisor plus fraction value. In LIN master-controller mode this register is only writable in reset mode.

The baud rate can be calculated with the following formula:

$$baudrate = \frac{t_{CLK\_LIN}}{16 \times INT + FRAC}$$

Example:

System clock frequency = 16 MHz, baudrate = 19.2 kBd:

INT = 52 = 0x34

FRAC = 0.08333 $\times$ 16 $\approx$ 1

$$\left(INT + \frac{FRAC}{16}\right) = \frac{Fclk(sys)}{16 \times baudrate} = \frac{16,000,000}{16 \times 19,200} = 52.08333$$

The value for this example of the fractional baud-rate generator register is LFBRG = 0x0001 0034.

Table 22–309 shows the bit assignment of the LFBRG register.

**Table 309. LIN master-controller fractional baud-rate generator register bit description (LIN0_LFBRG, address 0xE008 900C, LIN1_LFBRG, address 0xE008 A00C)**

** = reset value*

| Bit | Symbol | Access | | Value | Description |
|---|---|---|---|---|---|
| | | LIN mode | UART mode | | |
| 31 to 20 | reserved | R | - | - | Reserved; do not modify. Read as logic 0 |
| 19 to 16 | FRAC | R/W | R/W | | Fractional value. Contains the 4-bit fraction of the baud division |
| | | | | 0x* | |
| 15 to 0 | INT | R/W | R/W | | Integer value. Contains the 16-bit baud rate divisor |
| | | | | 0x0001* | |

## 7.2 LIN master controller registers

### 7.2.1 LIN master-controller status register

The LIN master-controller status register LSTAT reflects the status of the LIN master controller.

Figure 22–90 shows the status-flag handling in terms of transmitting and receiving header and response fields.

The LSTAT register is read-only. Table 22–310 shows its bit assignment.

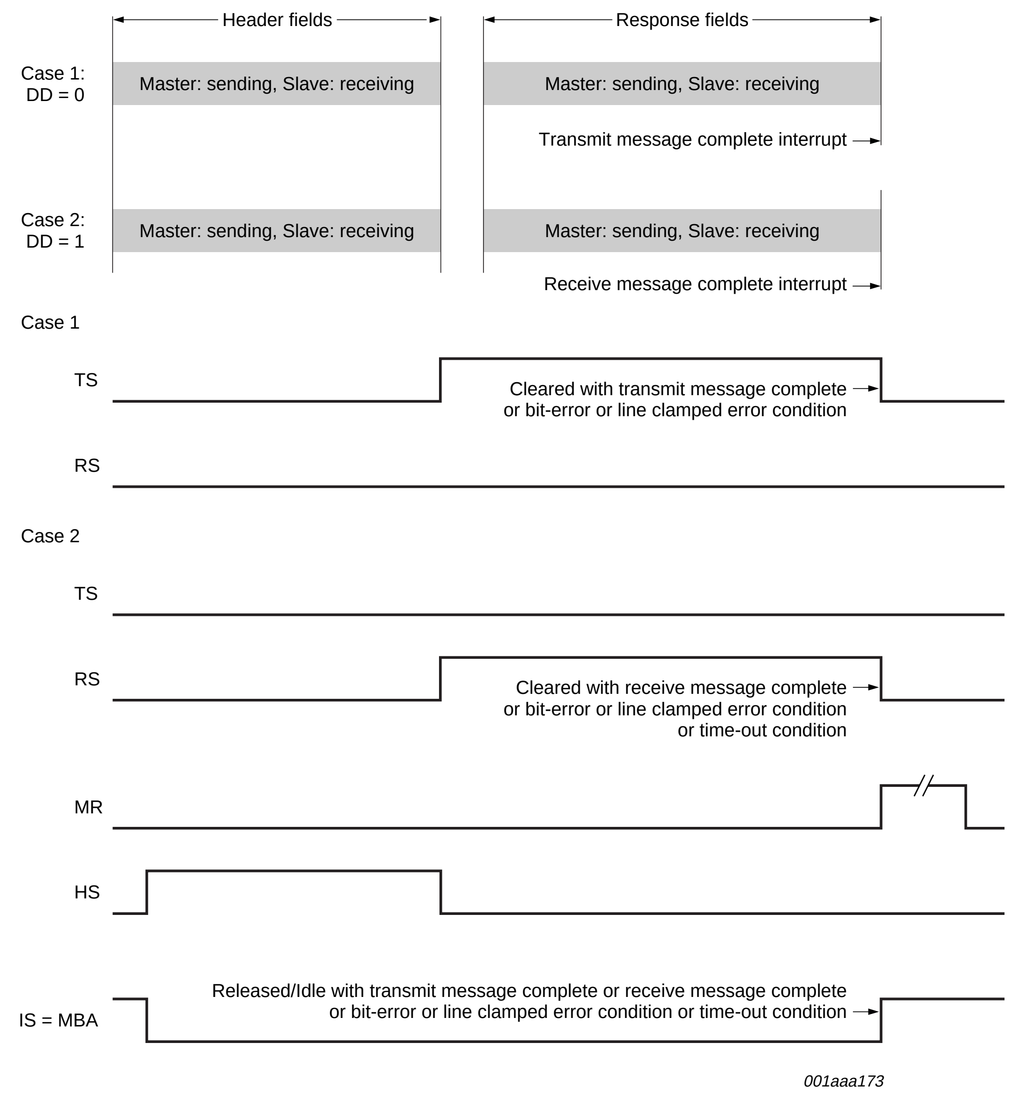

Header fields
Response fields
Case 1: DD = 0
Master: sending, Slave: receiving
Master: sending, Slave: receiving
Transmit message complete interrupt
Case 2: DD = 1
Master: sending, Slave: receiving
Master: sending, Slave: receiving
Receive message complete interrupt
Case 1
TS
Cleared with transmit message complete or bit-error or line clamped error condition
RS
Case 2
TS
RS
Cleared with receive message complete or bit-error or line clamped error condition or time-out condition
MR
HS
IS = MBA
Released/Idle with transmit message complete or receive message complete or bit-error or line clamped error condition or time-out condition
001aaa173

**Fig 90. LIN master-controller status-flag handling**

**Table 310. LIN master-controller status register bit description (LIN0_STAT, address 0xE008 9010, LIN1_LSTAT, address 0xE008 A010)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; read as logic 0 |
| 9 | TTL | R | | TXD line level |
| | | | 1* | The current TXD line level is dominant |
| | | | 0 | The current TXD line level is recessive |

Table 310. LIN master-controller status register bit description (LIN0_STAT, address 0xE008 9010, LIN1_LSTAT, address 0xE008 A010) *…continued*
* = *reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 8 | RLL | R | | RXD line level. |
| | | | 1* | The current RXD line level is dominant |
| | | | 0 | The current RXD line level is recessive |
| 7 | reserved | R | - | Reserved; read as logic 0 |
| 6 | IS | R | | Idle status |
| | | | 1* | The LIN bus is idle |
| | | | 0 | The LIN bus is active |
| 5 | ES | R | | Error status |
| | | | 1 | A bit-error or line-clamped error condition was detected |
| | | | 0* | No errors have been detected. The error status is cleared automatically when a new transmission is initiated |
| 4 | TS | R | | Transmit status |
| | | | 1 | The LIN master controller is transmitting LIN response fields |
| | | | 0* | |
| 3 | RS | R | | Receive status |
| | | | 1 | The LIN master controller is receiving LIN response fields |
| | | | 0* | |
| 2 | HS | R | | Header status |
| | | | 1 | The LIN master controller is transmitting LIN header fields |
| | | | 0* | |
| 1 | MBA | R | | Message buffer access |
| | | | 1* | The message buffer is released and available for CPU access |
| | | | 0 | The message buffer is locked and the CPU cannot access it. Either a message is waiting for transmission or is being transmitted, or the buffer is in the process of receiving a message |
| 0 | MR | R | | Message received |
| | | | 1 | The message buffer contains a valid received message |
| | | | 0* | The message buffer does not contain a valid message. The message-received status is cleared automatically with a write access to the message buffer or by a new transmission request |

### 7.2.2 LIN master-controller interrupt and capture register

The LIN master-controller interrupt and capture register LIC determines when the LIN master controller gives an interrupt request if the corresponding interrupt-enable has been set. Reading the interrupt register clears the interrupt source. A detailed bus-error capture is reported.

The LIC register is read-only. Table 22–311 shows its bit assignment.

**Table 311. LIN master-controller interrupt and capture register bit description (LIN0_LIC, address 0xE008 9014, LIN1_LIC, address 0xE008 A014)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; read as logic 0 |
| 11 to 8 | EC[3:0] | R | | Error capture |
| | | | 0000* | Bit error in sync-break field |
| | | | 0001 | Bit error in sync field |
| | | | 0010 | Bit error in identifier field |
| | | | 0011 | Bit error in data field |
| | | | 0100 | Bit error in checksum field |
| | | | 0101 | Bit error in inter-byte space |
| | | | 0110 | Bit error in stop bit of received slave responses |
| | | | 0111 | Reserved |
| | | | 1000 | Recessive line-clamped error. RXD / TXD line stuck recessive |
| | | | 1001 | Dominant line-clamped error. RXD / TXD line stuck dominant |
| | | | 1010 | Reserved |
| | | | : | : |
| | | | 1111 | Reserved |
| 7 | reserved | R | - | Reserved; read as logic 0 |
| 6 | WPI | R | | Wake-up and LIN protocol-error interrupt |
| | | | 1 | A dominant bus level has been detected when the LIN bus was idle. A dominant level on the LIN bus can be caused by a wake-up message from a slave node, or by arbitrarily created or faulty messages generated by LIN slaves, or by a stuck dominant level |
| | | | 0* | |
| 5 | RTLCEI | R | | Line-clamped error interrupt [1] |
| | | | 1 | No valid message can be generated on the LIN bus due to a clamped dominant or recessive RXD or TXD line |
| | | | 0* | |
| 4 | NRI | R | | Slave-not-responding error interrupt |
| | | | 1 | The slave response was not completed within a certain time-out period. The time-out period is configurable via the time-out register |
| | | | 0* | |

**Table 311. LIN master-controller interrupt and capture register bit description (LIN0_LIC, address 0xE008 9014, LIN1_LIC, address 0xE008 A014)** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 3 | CSI | R | | Checksum-error interrupt |
| | | | 1 | The received checksum field does not match the calculated checksum |
| | | | 0* | |
| 2 | BEI | R | | Bit-error interrupt [1] |
| | | | 1 | The error-capture bits represent detailed status in the case of<br>• A difference detected between the transmit and receive bit streams<br>• Violation of the configured inter-byte space length<br>• A stop-bit of fields from received slave responses was not recessive |
| | | | 0* | |
| 1 | TI | R | | Transmit-message complete interrupt |
| | | | 1 | A complete LIN message frame was transmitted, or in cases where data-length code is set to logic 0 (i.e. no response fields can be expected) |
| | | | 0* | |
| 0 | RI | R | | Receive-message complete interrupt |
| | | | 1 | The last byte. The checksum field of the incoming bit stream is moved from the receive shift register into the message buffer |
| | | | 0* | |

[1] The line-clamped interrupt RTLCEIE and the bit-error interrupt BEIE must be jointly enabled. Enabling only one interrupt is not allowed.

### 7.2.3 LIN master-controller interrupt enable register

The LIN master-controller interrupt enable register LIE determines when the LIN master-controller gives an interrupt request if the corresponding interrupt enable has been set.

Table 22–312 shows the bit assignment of the LIE register.

**Table 312. LIN master-controller interrupt enable register bit description (LIN0_LIE, address 0xE008 9018, LIN1_LIE, address 0xE008 A018)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 7 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

Table 312. LIN master-controller interrupt enable register bit description (LIN0_LIE, address 0xE008 9018, LIN1_LIE, address 0xE008 A018) *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 6 | WPIE | R/W | | Wake-up and LIN protocol error-interrupt enable |
| | | | 1 | Detection of a dominant bus level when the LIN bus was idle results in the corresponding interrupt |
| | | | 0* | |
| 5 | RTLCEIE | R/W | | Line-clamped error interrupt enable [1] |
| | | | 1 | Results in the corresponding interrupt when no valid message can be generated on the LIN bus |
| | | | 0* | |
| 4 | NRIE | R/W | | Slave-not-responding error interrupt enable |
| | | | 1* | Results in the corresponding interrupt when the slave response has not completed within the configured time-out period, |
| 3 | CSIE | R/W | | checksum error interrupt enable |
| | | | 1 | Results in the corresponding interrupt when the received checksum field does not match with the calculated checksum |
| | | | 0* | |
| 2 | BEIE | R/W | | Bit-error interrupt enable [1] |
| | | | 1 | Detection of a bit error results in the corresponding interrupt |
| | | | 0* | |
| 1 | TIE | R/W | | Transmit-message complete interrupt enable |
| | | | 1 | Results in the corresponding interrupt when a complete LIN message frame was transmitted, or in cases where the data-length code is set to logic 0 (i.e. no response fields can be expected) |
| | | | 0* | |
| 0 | RIE | R/W | | Receive-message complete interrupt enable |
| | | | 1 | Results in the corresponding interrupt when the last byte, the checksum field of the incoming bit-stream, is moved from receive shift register into the message buffer |
| | | | 0* | |

[1] The line-clamped interrupt RTLCEIE and the bit-error interrupt BEIE must be jointly enabled. Enabling only one interrupt is not allowed.

### 7.2.4 LIN master-controller checksum register

The LIN master-controller checksum register LCS contains the checksum value. When the LIN master controller is transmitting response fields this register contains the checksum value to be transmitted onto the LIN bus: when the LIN master controller is receiving response fields it contains the received checksum from the slave. If the software checksum bit in the configuration register is set to logic 0 the checksum register appears to the CPU as read-only memory. By setting the software checksum bit the checksum register appears to the CPU as read/write memory. In this case, and before a transmission is initiated, the software has to provide the checksum to the checksum register.

Table 22–313 shows the bit assignment of the LCS register.

**Table 313. LIN master-controller checksum register bit description (LIN0_LCS, address 0xE008 9020, LIN1_LCS, address 0xE008 A020)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 to 0 | CS | R/W | | LIN message checksum. When the LIN master controller is transmitting the checksum register contains the hardware- or software-calculated checksum value depending on the software checksum bit. When the LIN master controller is receiving the register contains the received checksum value from the slave node |
| | | | 0x00* | |

## 7.2.5 LIN master-controller time-out register

The LIN master-controller time-out register LTO defines the maximum number of bit times (TO) within which a response from all the LIN slaves connected to one node should have completed. The time-out starts as soon as the LIN header is transmitted (the value of the time-out register is decremented with every bit-time) and a slave response is expected. If it has been enabled, the slave-not-responding error interrupt NRI is asserted as soon as the time-out limit is exceeded.

In an application there are two possible ways of configuring the time-out condition:

1. Configure the time-out condition only once, during the initialization phase (applicable when all expected slave responses have the same or a similar length).
2. Configure the time-out condition prior to each LIN message (applicable when the expected slave-response length varies).

The time-out period to be programmed can be calculated from the following formula:

$$TO = \frac{t_{response(maximum)}}{T_{bit}} = 1.4 \times \frac{t_{response(nominal)}}{T_{bit}}$$

where:

$$t_{response(nominal)} = 10 \times \langle N_{data} + 1 \rangle \times T_{bit}$$

$T_{bit}$ is the nominal time required to transmit a bit, as defined in LIN physical layer; $N_{data}$ is the number of data fields sent with the slave response.

Table 22–314 shows the bit assignment of the LTO register.

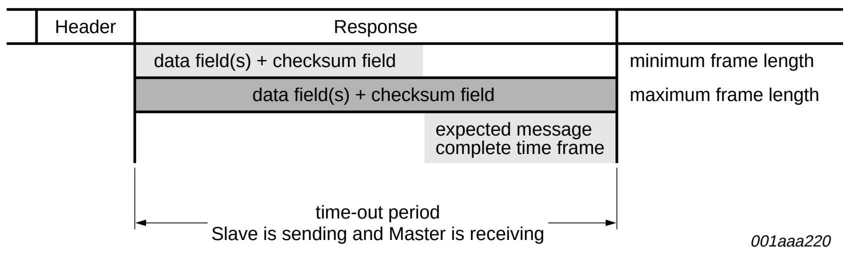

Header
Response
data field(s) + checksum field
minimum frame length
data field(s) + checksum field
maximum frame length
expected message complete time frame
time-out period
Slave is sending and Master is receiving
001aaa220

**Fig 91. Time-out period for all LIN slave nodes**

As Figure 22–92 below shows, the time-out period depends on the response time of the LIN slaves, and also on the number of data fields and the checksum field.

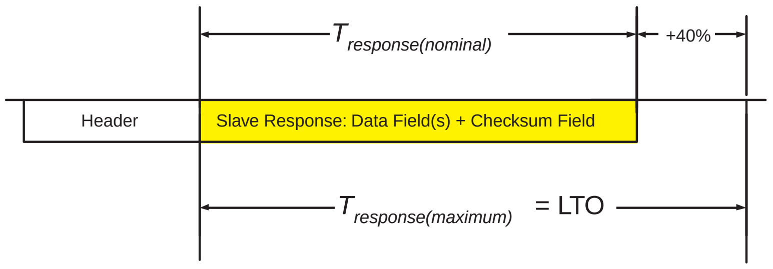

$T_{response(nominal)}$
+40%
Header
Slave Response: Data Field(s) + Checksum Field
$T_{response(maximum)}$ = LTO

**Fig 92. Time-out scenario**

The equation shown here is to calculate the LIN master time-out register (LTO) value:

$$LTO = \frac{T_{response(maximum)}}{T_{bit}}$$

In addition a further example shows how to use the equation to calculate the number of time-out bits:

$$T_{response(maximum)} = 1.4T_{response(nominal)} = 1.4(N_{data} + 1) \times T_{bit}$$

$$LTO = \frac{T_{response(maximum)}}{T_{bit}} = \frac{1.4T_{response(nominal)}}{T_{bit}} = \frac{1.4 \times 10(N_{data} + 1)T_{bit}}{T_{bit}}$$

For examples with definitions and equations from the LIN specification, see Ref. 31–6:

**Table 314. LIN master-controller time-out register bit description (LIN0_LTO, address 0xE008 9024, LIN1_LTO, address 0xE008 A024)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 to 0 | TO | R/W | | LIN message time-out. This defines the maximum number of bit-times within which a response from all slave nodes should have completed |
| | | | 0x00* | |

### 7.2.6 LIN master-controller message buffer registers

Access to the message buffer is limited and controlled by the message-buffer access bit of the status register. Access to the LIN master-controller message buffer registers is only possible when the LIN master-controller IP is in operating mode. Before accessing the message buffer the CPU should always read the message-buffer access bit first to determine whether an access is possible or not. In cases where the message buffer is locked a write-access will not succeed, but a read-access will deliver logic 0 as a result.

The first part of the message buffer is the LIN message-identifier register LID containing the header information and control format of the LIN message.

Time-out calculation examples:

Example 1 with one data field ($N_{data}$ = 1) in the expected slave response:

$$LTO = 1.4 \times 10(1 + 1) = 28$$

The value for this example of the LIN master time-out register is LTO = 0x0000 001C.

Example 2 with eight data fields ($N_{Data}$ = 8) in the expected slave response:

$$LTO = 1.4 \times 10(8 + 1) = 126$$

The value for this example of the LIN master time-out register is LTO = 0x0000 007E.

Table 22–315 shows the bit assignment of the LID register.

**Table 315. LIN message-identifier register bit description (LIN0_LID, address 0xE008 9028, LIN1_LID, address 0xE008 A028)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 26 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 315. LIN message-identifier register bit description (LIN0_LID, address 0xE008 9028, LIN1_LID, address 0xE008 A028)** *…continued*
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 25 | CSID | R/W | | Checksum ID inclusion |
| | | | 1 | The identifier field is included in the checksum calculation |
| | | | 0* | The identifier field is not included in the checksum calculation |
| 24 | DD | R/W | | Data direction |
| | | | 1 | The response field is expected to be sent by a slave node |
| | | | 0* | The response field is sent by the LIN master controller |
| 23 to 21 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 20 to 16 | DLC[4:0] | R/W | | Data-length code. This represents the binary number of data bytes in the LIN message-response field. Data-length code values greater than 16 are handled as the maximum number of 16 |
| | | | 0x00* | |
| 15 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 | P1 | R/W | | LIN message-parity bit 1 |
| | | | 0* | |
| 6 | P0 | R/W | | LIN message-parity bit 0 |
| | | | 0* | |
| 5 to 0 | ID | R/W | | LIN message identifier |
| | | | 0x00* | |

The rest of the message buffer contains the LIN message-data registers LDATA, LDATB, LDATC and LDATD.

Table 22–316 to Table 22–319 show the bit assignment of the LDATA, LDATB, LDATC and LDATD registers respectively.

**Table 316. LDATA register bit description (LIN0_LDATA, address 0xE008 902C, LIN1_LDATA, address 0xE008 A02C)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DF4[7:0] | R/W | | LIN message-data field 4 |
| | | | 0x00* | |
| 23 to 16 | DF3[7:0] | R/W | | LIN message-data field 3 |
| | | | 0x00* | |
| 15 to 8 | DF2[7:0] | R/W | | LIN message-data field 2 |
| | | | 0x00* | |
| 7 to 0 | DF1[7:0] | R/W | | LIN message-data field 1 |
| | | | 0x00* | |

**Table 317. LDATB register bit description (LIN0_LDATB, address 0xE008 9030, LIN1_LDATB, address 0xE008 A030)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DF8[7:0] | R/W | | LIN message-data field 8 |
| | | | 0x00* | |
| 23 to 16 | DF7[7:0] | R/W | | LIN message-data field 7 |
| | | | 0x00* | |
| 15 to 8 | DF6[7:0] | R/W | | LIN message-data field 6 |
| | | | 0x00* | |
| 7 to 0 | DF5[7:0] | R/W | | LIN message-data field 5 |
| | | | 0x00* | |

**Table 318. LDATC register bit description (LIN0_LDATC, address 0xE008 9034, LIN1_LDATC, address 0xE008 A034)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DF12[7:0] | R/W | | LIN message-data field 12 |
| | | | 0x00* | |
| 23 to 16 | DF11[7:0] | R/W | | LIN message-data field 11 |
| | | | 0x00* | |
| 15 to 8 | DF10[7:0] | R/W | | LIN message-data field 10 |
| | | | 0x00* | |
| 7 to 0 | DF9[7:0] | R/W | | LIN message-data field 9 |
| | | | 0x00* | |

**Table 319. LDATD register bit description (LIN0_LDATD, address 0xE008 9038, LIN1_LDATD, address 0xE008 A038)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 24 | DF16[7:0] | R/W | | LIN message-data field 16 |
| | | | 0x00* | |
| 23 to 16 | DF15[7:0] | R/W | | LIN message-data field 15 |
| | | | 0x00* | |
| 15 to 8 | DF14[7:0] | R/W | | LIN message-data field 14 |
| | | | 0x00* | |
| 7 to 0 | DF13[7:0] | R/W | | LIN message-data field 13 |
| | | | 0x00* | |

## 7.3 LIN/UART registers (LIN in UART mode)

### 7.3.1 LIN/UARTn Receiver Buffer Register

The LU0/1RBR is the top byte of the LIN/UARTn RX FIFO. The top byte of the RX FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

Since PE, FE and BI bits (see Table 22–327) 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 LSR register, and then to read a byte from the LU0/1RBR.

**Table 320. Lin/UARTn Receiver Buffer Register (LU0RBR - address 0xE008 9010, LU1RBR - 0xE008 A010, Read Only) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 31:8 | - | Reserved | NA |
| 7:0 | RBR | The LIN/UARTn Receiver Buffer Register contains the oldest received byte in the LIN/UARTn RX FIFO. | Undefined |

### 7.3.2 LIN/UARTn Transmit Holding Register

The LU0/1THR is the top byte of the LIN/UARTn TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

**Table 321. LIN/UARTn Transmit Holding Register (LU0THR - address 0xE008 9010, LU1THR - 0xE008 A010, Write Only) bit description**

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

### 7.3.3 LIN/UARTn Interrupt Enable Register

The LU0/1IER is used to enable the three LIN/UARTn interrupt sources.

**Table 322. LIN/UARTn Interrupt Enable Register (LU0IER - address 0xE008 9014, LU1IER - 0xE008 A014) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7:3 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | RX Line Status Interrupt Enable | | enables the LIN/UARTn RX line status interrupts. The status of this interrupt can be read from LU0/1LSR[4:1]. | 0 |
| | | 0 | Disable the RX line status interrupts. | |
| | | 1 | Enable the RX line status interrupts. | |

Table 322. LIN/UARTn Interrupt Enable Register (LU0IER - address 0xE008 9014, LU1IER - 0xE008 A014) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 1 | THRE Interrupt Enable | | enables the THRE interrupt for LIN/UARTn. The status of this can be read from LU0/1LSR[5]. | 0 |
| | | 0 | Disable the THRE interrupts. | |
| | | 1 | Enable the THRE interrupts. | |
| 0 | RBR Interrupt Enable | | enables the Receive Data Available interrupt for LIN/UARTn. It also controls the Character Receive Time-out interrupt. | 0 |
| | | 0 | Disable the RDA interrupts. | |
| | | 1 | Enable the RDA interrupts. | |

## 7.3.4 LIN/UARTn Interrupt Identification Register

The LU0/1IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an LU0/1IIR access. If an interrupt occurs during an LU0/1IIR access, the interrupt is recorded for the next LU0/1IIR access.

Table 323. LIN/UARTn Interrupt Identification Register (LU0IIR - address 0xE008 9018, LU1IIR - 0xE008 A018, Read Only) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 7:6 | FIFO Enable | | These bits are equivalent to LU0/1FCR[0]. | 0 |
| 5:4 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 3:1 | IntId | | Interrupt identification. LU0/1IER[3:1] identifies an interrupt corresponding to the LIN/UARTn RX FIFO. All other combinations of LU0/1IER[3:1] not listed above are reserved (000,100,101,111). | 0 |
| | | 011 | 1 - Receive Line Status (RLS). | |
| | | 010 | 2a - Receive Data Available (RDA). | |
| | | 110 | 2b - Character Time-out Indicator (CTI). | |
| | | 001 | 3 - THRE Interrupt | |
| 0 | IntStatus | | Interrupt status. Note that U1IIR[0] is active low. The pending interrupt can be determined by evaluating LU0/1IIR[3:1]. | 1 |
| | | 0 | At least one interrupt is pending. | |
| | | 1 | No interrupt is pending. | |

Bit LU0/1IIR[9:8] are set by the auto-baud function and signal a time-out or end of auto-baud condition. The auto-baud interrupt conditions are cleared by setting the corresponding Clear bits in the Auto-baud Control Register.

If the IntStatus bit is 1 no interrupt is pending and the IntId bits will be zero. If the IntStatus is 0, a non auto-baud interrupt is pending in which case the IntId bits identify the type of interrupt and handling as described in Table 22–324. Given the status of LU0/1IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The LU0/1IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The RLS interrupt (LU0/1IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the RX input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The RX error condition that set the interrupt can be observed via LU0/1LSR[4:1]. The interrupt is cleared upon an LU0/1LSR read.

The RDA interrupt (LU0/1IIR[3:1] = 010) shares the second level priority with the CTI interrupt (LU0/1IIR[3:1] = 110). The RDA is activated when the LIN/UARTn RX FIFO reaches the trigger level defined in LU0/1FCR[7:6] and is reset when the LIN/UARTn RX FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (LU0/1IIR[3:1] = 110) is a second level interrupt and is set when the LIN/UARTn RX FIFO contains at least one character and no LIN/UARTn RX FIFO activity has occurred in 3.5 to 4.5 character times. Any LIN/UARTn RX FIFO activity (read or write of LIN/UARTn RSR) will clear the interrupt. This interrupt is intended to flush the 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 324. LIN/UARTn 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] | LU0/1LSR Read[2] |
| 0100 | Second | RX Data Available | RX data available or trigger level reached in FIFO (LU0/1FCR0=1) | LU0/1RBR Read[3] or LIN/UARTn FIFO drops below trigger level |
| 1100 | Second | Character Time-out indication | Minimum of one character in the RX FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times).<br>The exact time will be:<br>[(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs | LU0/1RBR Read[3] |
| 0010 | Third | THRE | THRE[2] | LU0/1IIR Read (if source of interrupt) or THR write[4] |

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

[2] For details see Section 22–7.3.7 “LIN/UARTn Line Status Register”

[3] For details see Section 22–7.3.1 “LIN/UARTn Receiver Buffer Register”

[4] For details see Section 22–7.3.4 “LIN/UARTn Interrupt Identification Register” and Section 22–7.3.2 “LIN/UARTn Transmit Holding Register”

The THRE interrupt (LU0/1IIR[3:1] = 001) is a third level interrupt and is activated when the THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the 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 LU0/1THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to LU0/1THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the THR FIFO has held two or more characters at one time and currently, the LU0/1THR is empty. The THRE interrupt is reset when a LU0/1THR write occurs or a read of the LU0/1IIR occurs and the THRE is the highest interrupt (LU0/1IIR[3:1] = 001).

## 7.3.5 LIN/UARTn FIFO Control Register

The LU0/1FCR controls the operation of the LIN/UARTn RX and TX FIFOs.

Table 325. LIN/UARTn FIFO Control Register (LU0FCR - address 0xE008 9018, LU1FCR - 0xE008 A018, Write Only) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31:8 | - | - | Reserved | NA |
| 7:6 | RX Trigger Level | | These two bits determine how many receiver LIN/UARTn FIFO characters must be written before an interrupt is activated. | 0 |
| | | 00 | Trigger level 0 (1 character or 0x01) | |
| | | 01 | Trigger level 1 (4 characters or 0x04) | |
| | | 10 | Trigger level 2 (8 characters or 0x08) | |
| | | 11 | Trigger level 3 (14 characters or 0x0E) | |
| 5:3 | - | 0 | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 2 | TX FIFO Reset | 0 | No impact on either of LIN/UARTn FIFOs. | 0 |
| | | 1 | Writing a logic 1 to LU0/1FCR[2] will clear all bytes in LIN/UARTn TX FIFO and reset the pointer logic. This bit is self-clearing. | |
| 1 | RX FIFO Reset | 0 | No impact on either of LIN/UARTn FIFOs. | 0 |
| | | 1 | Writing a logic 1 to LU0/1FCR[1] will clear all bytes in LIN/UARTn RX FIFO and reset the pointer logic. This bit is self-clearing. | |
| 0 | FIFO Enable | 0 | LIN/UARTn FIFOs are disabled. Must not be used in the application. | 0 |
| | | 1 | Active high enable for both LIN/UARTn RX and TX FIFOs and LU0/1FCR[7:1] access. **This bit must be set for proper LIN/UARTn operation.** Any transition on this bit will automatically clear the LIN/UARTn FIFOs. | |

### 7.3.6 LIN/UARTn Line Control Register

The LU0/1LCR determines the format of the data character that is to be transmitted or received.

**Table 326. LIN/UARTn Line Control Register (LU0LCR - address 0xE008 901C, LU1LCR - 0xE008 A01C) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31: 7 | - | - | Reserved | NA |
| 6 | Break Control | 0 | Disable break transmission. | 0 |
| | | 1 | Enable break transmission. Output pin UART0 TXD is forced to logic 0 when LU0/1LCR[6] is active high. | |
| 5:4 | Parity Select | 00 | Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd. | 0 |
| | | 01 | Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even. | |
| | | 10 | Forced "1" stick parity. | |
| | | 11 | Forced "0" stick parity. | |
| 3 | Parity Enable | 0 | Disable parity generation and checking. | 0 |
| | | 1 | Enable parity generation and checking. | |
| 2 | Stop Bit Select | 0 | 1 stop bit. | 0 |
| | | 1 | 2 stop bits (1.5 if LU0/1LCR[1:0]=00). | |
| 1:0 | Word Length Select | 00 | 5 bit character length | 0 |
| | | 01 | 6 bit character length | |
| | | 10 | 7 bit character length | |
| | | 11 | 8 bit character length | |

### 7.3.7 LIN/UARTn Line Status Register

The LU0/1LSR is a read-only register that provides status information on the LIN/UARTn TX and RX blocks.

**Table 327. LIN/UARTn Line Status Register (LU0LSR - address 0xE008 9024, LU1LSR - 0xE008 A024, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 31: 8 | - | - | Reserved | - |
| 7 | Error in RX FIFO (RXFE) | | LU0/1LSR[7] is set when a character with a RX error such as framing error, parity error or break interrupt, is loaded into the LU0/1RBR. This bit is cleared when the LU0/1LSR register is read and there are no subsequent errors in the LIN/UARTn FIFO. | 0 |
| | | 0 | LU0/1RBR contains no LIN/UARTn RX errors or LU0/1FCR[0]=0. | |
| | | 1 | LIN/UARTn RBR contains at least one RX error. | |

Table 327. LIN/UARTn Line Status Register (LU0LSR - address 0xE008 9024, LU1LSR - 0xE008 A024, Read Only) bit description

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 6 | Transmitter Empty (TEMT) | | TEMT is set when both LU0/1THR and LU0/1TSR are empty; TEMT is cleared when either the LU0/1TSR or the LU0/1THR contain valid data. | 1 |
| | | 0 | LU0/1THR and/or the LU0/1TSR contains valid data. | |
| | | 1 | LU0/1THR and the LU0/1TSR are empty. | |
| 5 | Transmitter Holding Register Empty (THRE)) | | THRE is set immediately upon detection of an empty THR and is cleared on a LU0/1THR write. | 1 |
| | | 0 | LU0/1THR contains valid data. | |
| | | 1 | LU0/1THR is empty. | |
| 4 | Break Interrupt (BI) | | When RXDn is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RXDn goes to marking state (all 1's). An LU0/1LSR read clears this status bit. The time of break detection is dependent on LU0/1FCR[0].<br>**Note:** The break interrupt is associated with the character at the top of the LIN/UARTn RBR FIFO. | 0 |
| | | 0 | Break interrupt status is inactive. | |
| | | 1 | Break interrupt status is active. | |
| 3 | Framing Error (FE) | | When the stop bit of a received character is a logic 0, a framing error occurs. An LU0/1LSR read clears LU0/1LSR[3]. The time of the framing error detection is dependent on LU0/1FCR[0]. Upon detection of a framing error, the RX will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error.<br>**Note:** A framing error is associated with the character at the top of the LIN/UARTn RBR FIFO. | 0 |
| | | 0 | Framing error status is inactive. | |
| | | 1 | Framing error status is active. | |
| 2 | Parity Error (PE) | | When the parity bit of a received character is in the wrong state, a parity error occurs. An LU0/1LSR read clears LU0/1LSR[2]. Time of parity error detection is dependent on LU0/1FCR[0].<br>**Note:** A parity error is associated with the character at the top of the LIN/UARTn RBR FIFO. | 0 |
| | | 0 | Parity error status is inactive. | |
| | | 1 | Parity error status is active. | |
| 1 | Overrun Error (OE) | | The overrun error condition is set as soon as it occurs. An LU0/1LSR read clears LU0/1LSR1. LU0/1LSR1 is set when RSR has a new character assembled and the RBR FIFO is full. In this case, the RBR FIFO will not be overwritten and the character in the RSR will be lost. | 0 |
| | | 0 | Overrun error status is inactive. | |
| | | 1 | Overrun error status is active. | |

**Table 327. LIN/UARTn Line Status Register (LU0LSR - address 0xE008 9024, LU1LSR - 0xE008 A024, Read Only) bit description**

| Bit | Symbol | Value | Description | Reset Value |
|---|---|---|---|---|
| 0 | Receiver Data Ready (RDR) | | LU0/1LSR0 is set when the LU0/1RBR holds an unread character and is cleared when the RBR FIFO is empty. | 0 |
| | | 0 | LU0/1RBR is empty. | |
| | | 1 | LU0/1RBR contains valid data. | |

### 7.3.8 LIN/UARTn Scratch Pad Register

The LU0/1SCR has no effect on the LIN/UARTn operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the LU0/1SCR has occurred.

**Table 328. LIN/UARTn Scratch Pad Register (LU0SCR - address 0xE028 901C, LU1SCR - 0xE008 A02C) bit description**

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

### 7.3.9 LIN/UARTn Transmit Enable Register

The LU0/1TER register enables implementation of software flow control. When TXEn=1, LIN/UARTn transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, LIN/UARTn transmission will stop.

Table 22–329 describes how to use TXEn bit in order to achieve software flow control.

**Table 329. LIN/UARTn Transmit Enable Register (LU0TER - address 0xE008 9040, LU1TER - 0xE008 A040) bit description**

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

## 8. Step-by-step example for using the LIN master

The following is a short example showing how to configure the LIN master controller and transmit or receive LIN message frames.

The example uses hardware support from the LIN master, so the software ID parity (SWPA) and the checksum (SWCS) are generated automatically. LIN master interrupts are used for message reception and transmission.

General configuration of the LIN master controller is only performed once; typically during the initialization phase. The sequence is:

1. Do the LIN master port-pin configuration of the multiplexed I/O pins for the desired LIN channels.
2. Enter reset mode by setting the LRM bit of the LIN master-controller mode register LMODE.
3. Choose a baud rate by writing the appropriate value to the LIN master-controller fractional baud-rate generator register LFBRG, see also Section 22–7.
4. Do a general LIN master configuration and choose the LIN inter-byte space length (IBS) and sync-break low length (SBL) in the LIN master controller configuration register LCFG.
5. Put the LIN master in normal operating mode by clearing the LRM bit of the LIN master-controller mode register LMODE.

All LIN message activity is initiated by the LIN master. By sending a LIN header (see Ref. 31–6) the LIN master determines the configuration of the response. The response can be sent either by the master or the slave.

Table 22–330 provides a typical step-by-step description for both.

**Table 330. Transmitting/receiving step by step**

| LIN master transmits the response | LIN master receives the response |
|---|---|
| Prepare a transmit message: provide the LIN identifier, the data-length code DLC and, if required, the message data to the LIN master message buffer. The data direction bit DD is set to LOW. | Choose a slave-not-responding time-out condition and write the related value into the LIN master-controller time-out register LTO, see also Section 22–5.1.6. |
| Initiate the transmission by setting transmit-request bit TR in the LIN master controller command register LCMD. | Prepare a transmit message (LIN Header): Provide the LIN identifier and the data-length code DLC of the expected response to the LIN master message Buffer. The data direction bit DD is set to high. |
| A transmit-message complete interrupt signals successful transmission of the message. | Initiate the transmission by setting transmit-request bit TR in the LIN master-controller command register LCMD. |
| | The LIN master sends the LIN header. A receive-message complete interrupt signals successful reception of the slave response and the message is then stored in the LIN master's message buffer. |

# UM10316

## Chapter 23: LPC29xx I2C-interface

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Features

- I$^2$C0 and I$^2$C1 use standard I/O pins with bit rates of up to 400 kbit/s (Fast I$^2$C-bus) and do not support powering off of individual devices connected to the same bus lines.
- Easy to configure as master, slave, or master/slave.
- Programmable clocks allow versatile rate control.
- Bidirectional data transfer between masters and slaves.
- Multi-master bus (no central master).
- Arbitration between simultaneously transmitting masters without corruption of serial data on the bus.
- Serial clock synchronization allows devices with different bit rates to communicate via one serial bus.
- Serial clock synchronization can be used as a handshake mechanism to suspend and resume serial transfer.
- The I$^2$C-bus can be used for test and diagnostic purposes.
- All I$^2$C-bus controllers support multiple address recognition and a bus monitor mode.

## 3. Applications

Interfaces to external I$^2$C standard parts, such as serial RAMs, LCDs, tone generators, etc.

## 4. Description

A typical I$^2$C bus configuration is shown in Figure 23–93. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I$^2$C bus:

- Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte.
- 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$^2$C bus will not be released.

Each of the two I$^2$C interfaces on the LPC29xx is byte oriented, and has four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

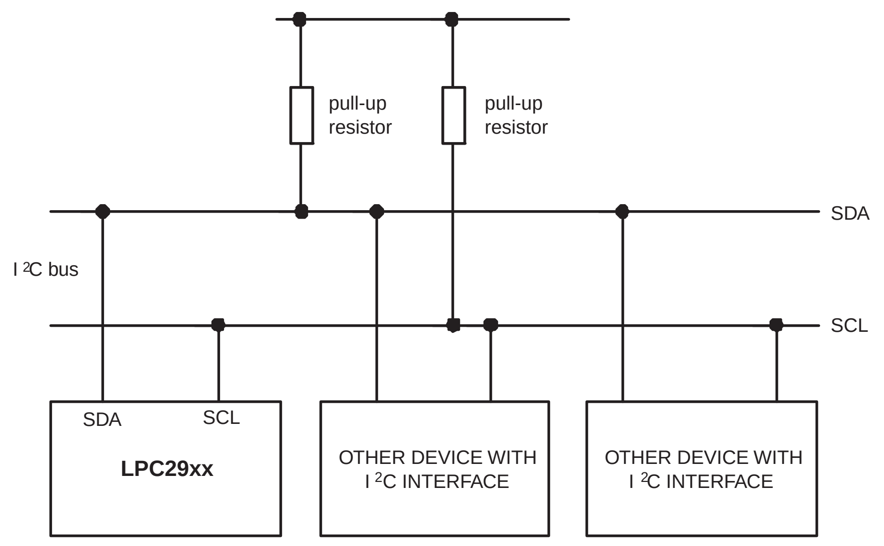

pull-up resistor
pull-up resistor
SDA
I 2C bus
SCL
SDA
SCL
LPC29xx
OTHER DEVICE WITH I 2C INTERFACE
OTHER DEVICE WITH I 2C INTERFACE

**Fig 93. I$^2$C bus configuration**

## 5. Pin description

When connecting to a 5 V I$^2$C bus, the internal pull-up resistor must be disabled in the SFSPn_m register Table 6–59. The value of the external pull-up resistor influences the rising edge of the SDA and SCL signals.

For low-frequency applications (< 20 kHz) and short data lines, the internal pull-up is sufficient to drive the I2C-bus.

**Table 331. I$^2$C pin description**

| Pin | Type | Description |
|---|---|---|
| I2C0_SDA, I2C1_SDA | Input/Output | I$^2$C Serial Data. |
| I2C0_SCL, I2C1_SCL | Input/Output | **I$^2$C Serial Clock.** |

## 6. I$^2$C operating modes

In a given application, the I$^2$C block may operate as a master, a slave, or both. In the slave mode, the I$^2$C hardware looks for its own slave address and the general call address. If one of these addresses is detected, an interrupt is requested. If the processor wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave operation is not interrupted. If bus arbitration is lost in the master mode, the I$^2$C block switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

## 6.1 Master Transmitter mode

In this mode data is transmitted from master to slave. Before the master transmitter mode can be entered, the I2CONSET register must be initialized as shown in Table 23–332. I2EN must be set to 1 to enable the I$^2$C function. If the AA bit is 0, the I$^2$C interface will not acknowledge any address when another device is master of the bus, so it can not enter slave mode. The STA, STO and SI bits must be 0. The SI Bit is cleared by writing 1 to the SIC bit in the I2CONCLR register.

**Table 332. I2CnCONSET 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$^2$C interface will enter master transmitter mode when software sets the STA bit. The I$^2$C logic will send the START condition as soon as the bus is free. After the START condition is transmitted, the SI bit is set, and the status code in the I2STAT register is 0x08. This status code is used to vector to a state service routine which will load the slave address and Write bit to the I2DAT register, and then clear the SI bit. SI is cleared by writing a 1 to the SIC bit in the I2CONCLR register. The STA bit should be cleared after writing the slave address.

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 23–351 to Table 23–354.

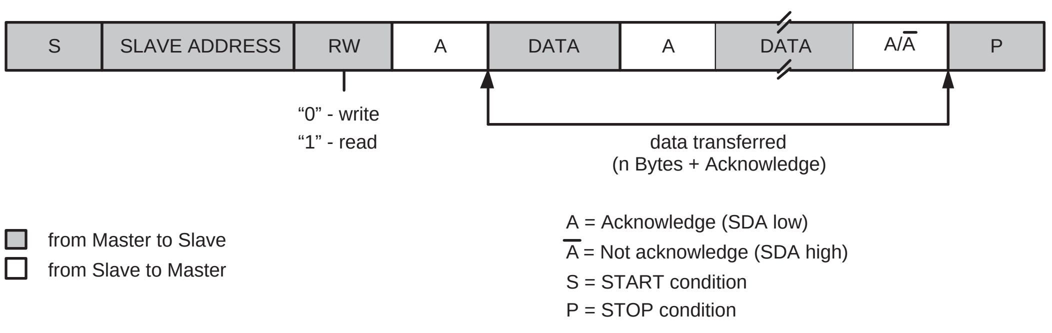


**Fig 94. Format in the Master Transmitter mode**

## 6.2 Master Receiver mode

In the master receiver mode, data is received from a slave transmitter. The transfer is initiated in the same way as in the master transmitter mode. When the START condition has been transmitted, the interrupt service routine must load the slave address and the data direction bit to the I$^2$C Data Register (I2DAT), and then clear the SI bit. In this case, the data direction bit (R/W) should be 1 to indicate a read.

When the slave address and data direction bit have been transmitted and an acknowledge bit has been received, the SI bit is set, and the Status Register will show the status code. For master mode, the possible status codes are 0x40, 0x48, or 0x38. For slave mode, the possible status codes are 0x68, 0x78, or 0xB0. For details, refer to Table 23–352.

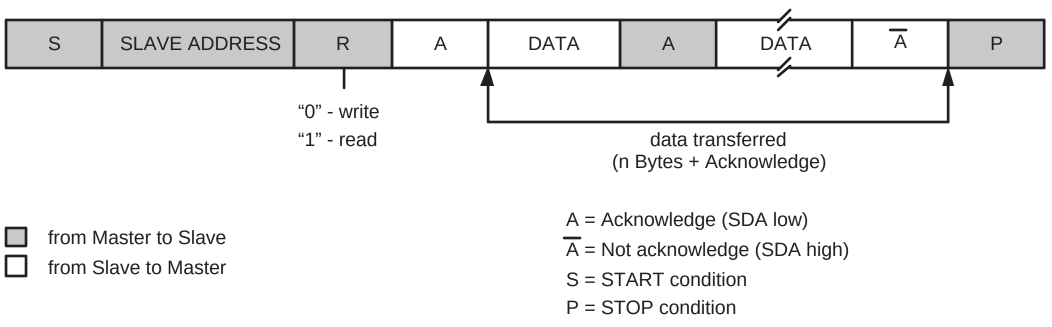


**Fig 95. Format of Master Receive mode**

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

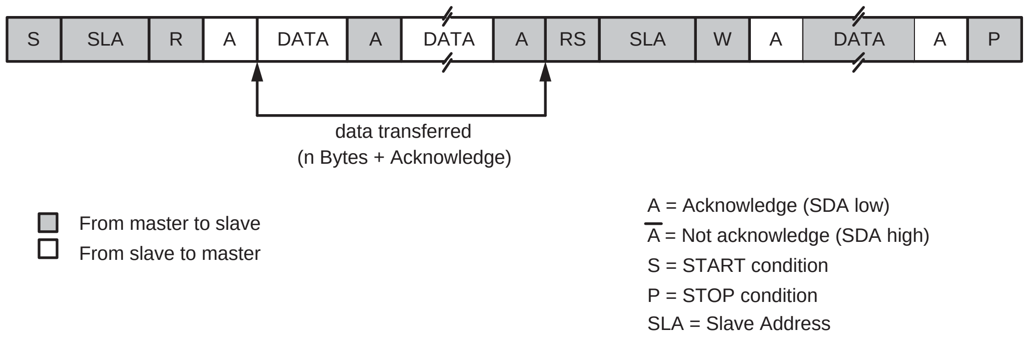


**Fig 96. A master receiver switch to master Transmitter after sending repeated START**

## 6.3 Slave Receiver mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, write any of the the Slave Address Registers (ADR0-3) and write the I$^2$C Control Set Register (I2CONSET) as shown in Table 23–333.

Table 333. I2CnCONSET used to configure Slave mode

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | 1 | - | - |

I2EN must be set to 1 to enable the I$^2$C 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 I2ADR and I2CONSET are initialized, the I$^2$C 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 (I2STAT). Refer to Table 23–353 for the status codes and actions.

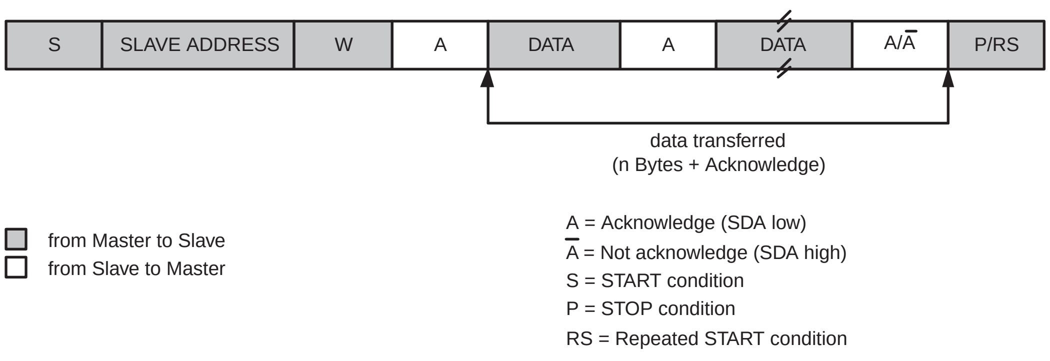


Fig 97. Format of Slave Receiver mode

## 6.4 Slave Transmitter mode

The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will be 1, indicating a read operation. Serial data is transmitted via SDA while the serial clock is input through SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. In a given application, I$^2$C may operate as a master and as a slave. In the slave mode, the I$^2$C 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$^2$C interface switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

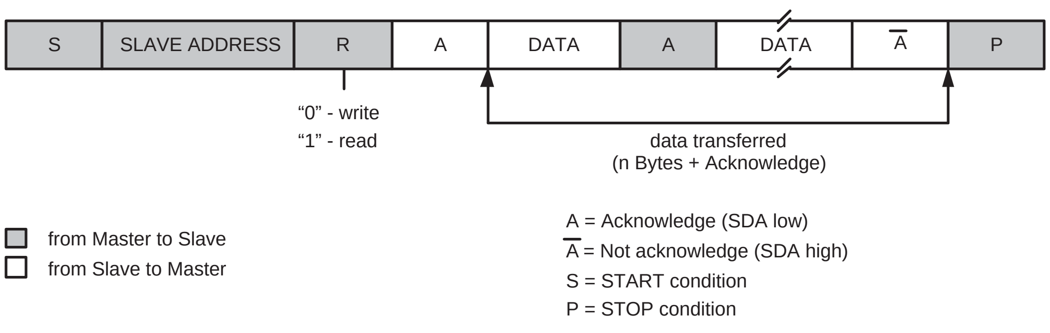


**Fig 98. Format of Slave Transmitter mode**

## 7. I$^2$C implementation and operation

### 7.1 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 I$^2$C is a special pad designed to conform to the I$^2$C specification. The outputs for I2C1 and I2C2 are standard port I/Os that support a subset of the full I$^2$C specification.

Figure 23–99 shows how the on-chip I$^2$C bus interface is implemented, and the following text describes the individual blocks.

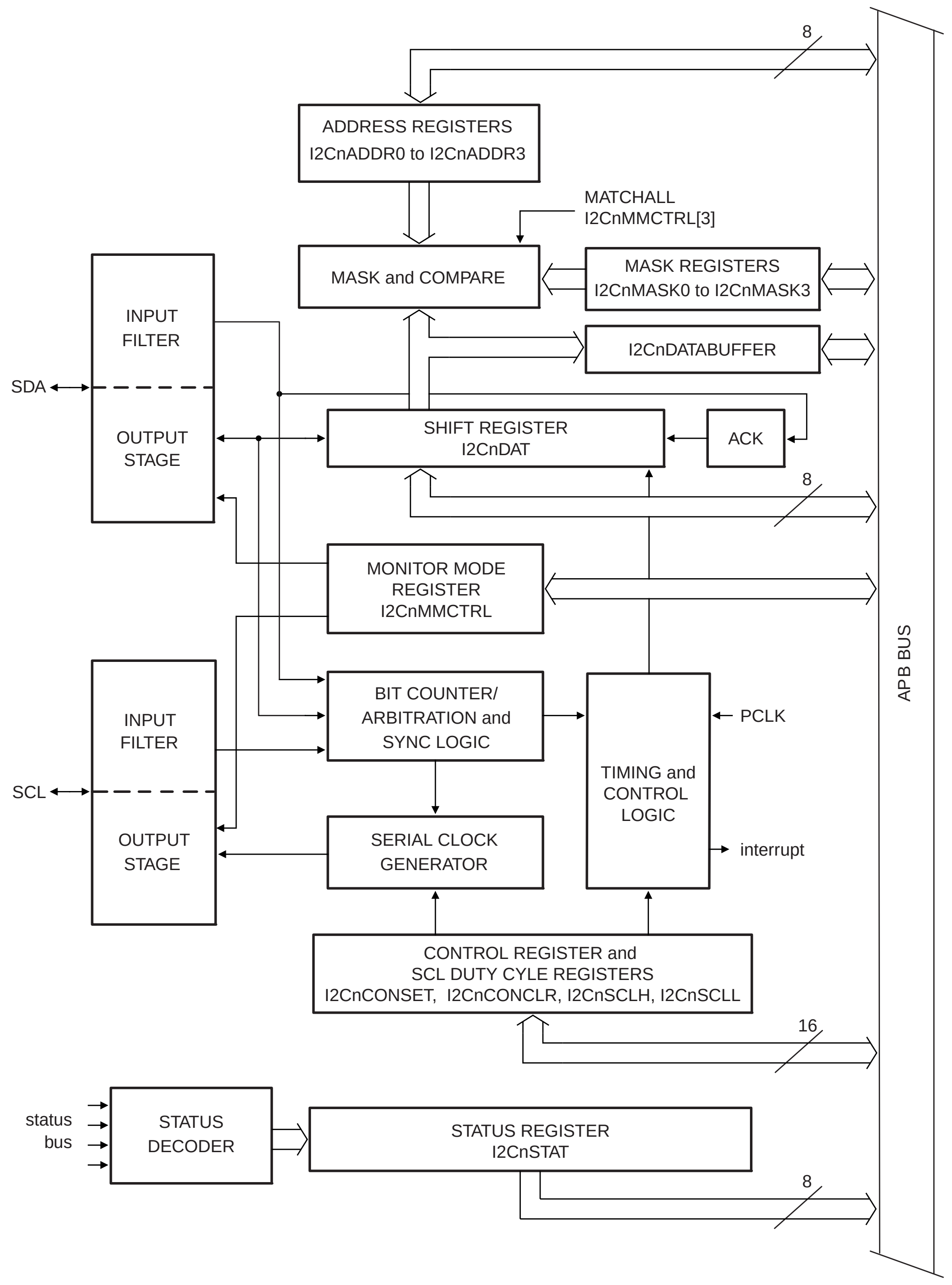


**Fig 99. I$^2$C Bus serial interface block diagram**

## 7.2 Address Registers, ADDR0 to ADDR3

These registers may be loaded with the 7-bit slave address (7 most significant bits) to which the I$^2$C block will respond when programmed as a slave transmitter or receiver. The LSB (GC) is used to enable general call address (0x00) recognition. When multiple slave addresses are enabled, the actual address received may be read from the DAT register at the state where the own slave address has been received.

## 7.3 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.

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

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.

## 7.4 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.

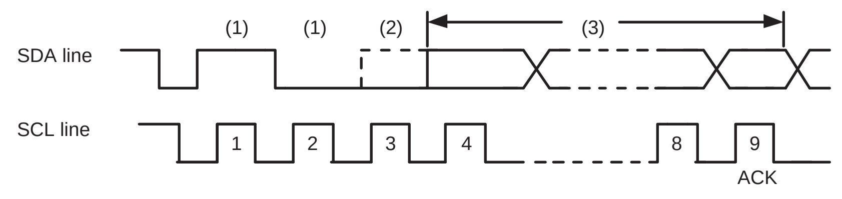
## 7.5 Shift register DAT

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.

## 7.6 Arbitration and synchronization logic

In the master transmitter mode, the arbitration logic checks that every transmitted logic 1 actually appears as a logic 1 on the I$^2$C bus. If another device on the bus overrules a logic 1 and pulls the SDA line low, arbitration is lost, and the I$^2$C block immediately changes from master transmitter to slave receiver. The I$^2$C block will continue to output clock pulses (on SCL) until transmission of the current serial byte is complete.

Arbitration may also be lost in the master receiver mode. Loss of arbitration in this mode can only occur while the I$^2$C block is returning a “not acknowledge: (logic 1) to the bus. Arbitration is lost when another device on the bus pulls this signal LOW. Since this can occur only at the end of a serial byte, the I$^2$C block generates no further clock pulses. Figure 23–100 shows the arbitration procedure.



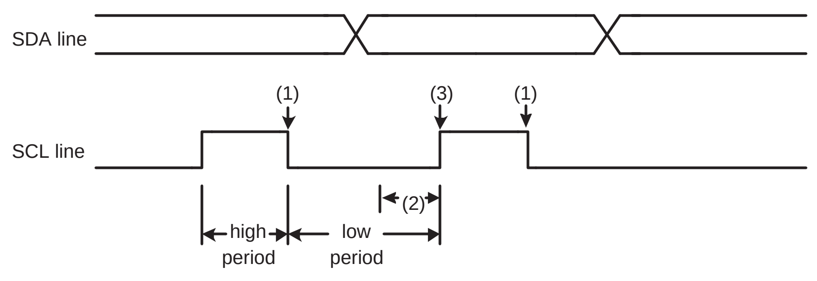
(1) A device transmits serial data.

(2) Another device overrules a logic 1 (dotted line), transmitted by this I$^2$C master, by pulling the SDA line low. Arbitration is lost, and this I$^2$C enters Slave Receiver mode.

(3) This I$^2$C is in Slave Receiver mode but still generates clock pulses until the current byte has been transmitted. This I$^2$C will not generate clock pulses for the next byte. Data on SDA originates from the new master once it has won arbitration.

**Fig 100. 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 23–101 shows the synchronization procedure.



(1) Another device pulls the SCL line low before this I$^2$C 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 I$^2$C has timed a complete low time and released SCL. The I$^2$C 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 101. 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 I$^2$C block will stretch the SCL space duration after a byte has been transmitted or received and the acknowledge bit has been transferred. The serial interrupt flag (SI) is set, and the stretching continues until the serial interrupt flag is cleared.

## 7.7 Serial clock generator

This programmable clock pulse generator provides the SCL clock pulses when the I$^2$C block is in the master transmitter or master receiver mode. It is switched off when the I$^2$C block is in a slave mode. The I$^2$C output clock frequency and duty cycle is programmable via the I$^2$C Clock Control Registers. See the description of the I2CSCLL and I2CSCLH registers for details. The output clock pulses have a duty cycle as programmed unless the bus is synchronizing with other SCL clock sources as described above.

### 7.8 Timing and control

The timing and control logic generates the timing and control signals for serial byte handling. This logic block provides the shift pulses for I2DAT, enables the comparator, generates and detects start and stop conditions, receives and transmits acknowledge bits, controls the master and slave modes, contains interrupt request logic, and monitors the I$^2$C bus status.

### 7.9 Control register CONSET and CONCLR

The I$^2$C control register contains bits used to control the following I$^2$C block functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.

The contents of the I$^2$C control register may be read as CONSET. Writing to ICONSET will set bits in the I$^2$C control register that correspond to ones in the value written. Conversely, writing to CONCLR will clear bits in the I$^2$C control register that correspond to ones in the value written.

### 7.10 Status decoder and status register

The status decoder takes all of the internal status bits and compresses them into a 5 bit code. This code is unique for each I$^2$C bus status. The 5 bit code may be used to generate vector addresses for fast processing of the various service routines. Each service routine processes a particular bus status. There are 26 possible bus states if all four modes of the I$^2$C block are used. The 5 bit status code is latched into the five most significant bits of the status register when the serial interrupt flag is set (by hardware) and remains stable until the interrupt flag is cleared by software. The three least significant bits of the status register are always zero. If the status code is used as a vector to service routines, then the routines are displaced by eight address locations. Eight bytes of code is sufficient for most of the service routines (see the software example in this section).

## 8. Register overview

Table 334. Register overview: I2C (base address 0xE008 2000 (I2C0) and 0xE008 3000 (I2C1))

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| CONSET | R/W | 0x00 | **I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I$^2$C control register is set. Writing a zero has no effect on the corresponding bit in the I$^2$C control register. | 0x00 |
| STAT | RO | 0x04 | **I2C Status Register.** During I$^2$C operation, this register provides detailed status codes that allow software to determine the next action needed. | 0xF8 |
| DAT | R/W | 0x08 | **I2C Data Register.** During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register. | 0x00 |

Table 334. Register overview: I2C (base address 0xE008 2000 (I2C0) and 0xE008 3000 (I2C1))

| Name | Access | Address offset | Description | Reset value[1] |
|---|---|---|---|---|
| ADR | R/W | 0x0C | **I2C Slave Address Register.** Contains the 7 bit slave address for operation of the I$^2$C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address. | 0x00 |
| SCLH | R/W | 0x10 | **SCH Duty Cycle Register High Half Word.** Determines the high time of the I$^2$C clock. | 0x04 |
| SCLL | R/W | 0x14 | **SCL Duty Cycle Register Low Half Word.** Determines the low time of the I$^2$C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I$^2$C master and certain times used in slave mode. | 0x04 |
| CONCLR | WO | 0x18 | **I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I$^2$C control register is cleared. Writing a zero has no effect on the corresponding bit in the I$^2$C control register. | NA |
| MMCTRL | R/W | 0x1C | **Monitor mode control register.** | 0x00 |
| ADR1 | R/W | 0x20 | **I$^2$C Slave Address Register 1.** Contains the 7-bit slave address for operation of the I$^2$C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address. | 0x00 |
| ADR2 | R/W | 0x24 | **I$^2$C Slave Address Register 2.** Contains the 7-bit slave address for operation of the I$^2$C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address. | 0x00 |
| ADR3 | R/W | 0x28 | **I$^2$C Slave Address Register 3.** Contains the 7-bit slave address for operation of the I$^2$C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address. | 0x00 |
| DATA_ BUFFER | RO | 0x2C | **Data buffer register.** The contents of the 8 msb's 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 |
| MASK0 | R/W | 0x30 | **I$^2$C 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 |
| MASK1 | R/W | 0x34 | **I$^2$C 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 |
| MASK2 | R/W | 0x38 | **I$^2$C 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 |
| MASK3 | R/W | 0x3C | **I$^2$C 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.

## 8.1 I$^2$C Control Set Register (I2C[0/1]CONSET: 0xE008 2000, 0xE008 3000)

The I2CONSET registers control setting of bits in the I2CON register that controls operation of the I$^2$C interface. Writing a one to a bit of this register causes the corresponding bit in the I$^2$C control register to be set. Writing a zero has no effect.

Table 335. I$^2$C Control Set Register (I2C[0/1]CONSET - addresses: 0xE008 2000, 0xE008 3000) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 6 | I2EN | I$^2$C interface enable. See the text below. | 0 |
| 5 | STA | START flag. See the text below. | 0 |
| 4 | STO | STOP flag. See the text below. | 0 |
| 3 | SI | I$^2$C interrupt flag. | 0 |
| 2 | AA | Assert acknowledge flag. See the text below. | |
| 1:0 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

**I2EN** I$^2$C Interface Enable. When I2EN is 1, the I$^2$C interface is enabled. I2EN can be cleared by writing 1 to the I2ENC bit in the I2CONCLR register. When I2EN is 0, the I$^2$C interface is disabled.

When I2EN is “0”, the SDA and SCL input signals are ignored, the I$^2$C block is in the “not addressed” slave state, and the STO bit is forced to “0”.

I2EN should not be used to temporarily release the I$^2$C bus since, when I2EN is reset, the I$^2$C bus status is lost. The AA flag should be used instead.

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

When STA is 1 and the I$^2$C interface is not already in master mode, it enters master mode, checks the bus and generates a START condition if the bus is free. If the bus is not free, it waits for a STOP condition (which will free the bus) and generates a START condition after a delay of a half clock period of the internal clock generator. If the I$^2$C interface is already in master mode and data has been transmitted or received, it transmits a repeated START condition. STA may be set at any time, including when the I$^2$C interface is in an addressed slave mode.

STA can be cleared by writing 1 to the STAC bit in the I2CONCLR register. When STA is 0, no START condition or repeated START condition will be generated.

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

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

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

**SI** is the I$^2$C Interrupt Flag. This bit is set when the I$^2$C state changes. However, entering state F8 does not set SI since there is nothing for an interrupt service routine to do in that case.

While SI is set, the low period of the serial clock on the SCL line is stretched, and the serial transfer is suspended. When SCL is high, it is unaffected by the state of the SI flag. SI must be reset by software, by writing a 1 to the SIC bit in I2CONCLR register.

**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 I2ADR is set.
3. A data byte has been received while the I$^2$C is in the master receiver mode.
4. A data byte has been received while the I$^2$C is in the addressed slave receiver mode.

The AA bit can be cleared by writing 1 to the AAC bit in the I2CONCLR register. When AA is 0, a not acknowledge (high level to SDA) will be returned during the acknowledge clock pulse on the SCL line on the following situations:

1. A data byte has been received while the I$^2$C is in the master receiver mode.
2. A data byte has been received while the I$^2$C is in the addressed slave receiver mode.

## 8.2 I$^2$C Status Register (I2C[0/1]STAT - 0xE008 2004, 0xE008 3004)

Each I$^2$C Status register reflects the condition of the corresponding I$^2$C interface. The I$^2$C Status register is Read-Only.

**Table 336. I$^2$C Status Register (I2C[0/1]STAT - addresses 0xE008 2004, 0xE008 3004) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:3 | Status | These bits give the actual status information about the I$^2$C interface. | 0x1F |
| 2:0 | - | These bits are unused and are always 0. | 0 |

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$^2$C states. When any of these states entered, the SI bit will be set. For a complete list of status codes, refer to tables from Table 23–351 to Table 23–354.

## 8.3 I$^2$C Data Register (I2C[0/1]DAT - 0xE008 2008, 0xE008 3008)

This register contains the data to be transmitted or the data just received. The CPU can read and write to this register only while it is not in the process of shifting a byte, when the SI bit is set. Data in I2DAT remains stable as long as the SI bit is set. Data in I2DAT is always shifted from right to left: the first bit to be transmitted is the MSB (bit 7), and after a byte has been received, the first bit of received data is located at the MSB of I2DAT.

Table 337. I$^2$C Data Register ( I2C[0/1]DAT - addresses 0xE008 2008, 0xE008 3008) 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 |

## 8.4 I$^2$C Slave Address Register (I2C[0/1]ADR - 0xE008 200C, 0xE008 300C)

These registers are readable and writable, and is only used when an I$^2$C interface is set to slave mode. In master mode, this register has no effect. The LSB of I2ADR is the general call bit. When this bit is set, the general call address (0x00) is recognized.

Table 338. I$^2$C Slave Address register (I2C[0/1]ADR - addresses 0xE008 200C, 0xE008 300C) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7:1 | Address | The I$^2$C device address for slave mode. | 0x00 |
| 0 | GC | General Call enable bit. | 0 |

## 8.5 I$^2$C SCL High Duty Cycle Register (I2C[0/1]SCLH - 0xE008 2010, 0xE008 3010)

Table 339. I$^2$C SCL High Duty Cycle register (I2C[0/1]SCLH - addresses 0xE008 2010, 0xE008 3010) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 15:0 | SCLH | Count for SCL HIGH time period selection. | 0x0004 |

## 8.6 I$^2$C SCL Low Duty Cycle Register (I2C[0/1]SCLL - 0xE008 2014, 0xE008 3014)

Table 340. I$^2$C SCL Low Duty Cycle register (I2C[0/1]SCLL - addresses 0xE008 2014, 0xE008 3014) bit description

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 15:0 | SCLL | Count for SCL LOW time period selection. | 0x0004 |

### 8.6.1 Selecting the appropriate I$^2$C data rate and duty cycle

Software must set values for the registers I2SCLH and I2SCLL to select the appropriate data rate and duty cycle. I2SCLH defines the number of PCLK cycles for the SCL high time, I2SCLL defines the number of PCLK cycles for the SCL low time. The peripheral I2C clock PCLK is the base clock BASE_IVNSS_CLK. The frequency is determined by the following formula ($f_{PCLK}$ being the frequency of PCLK):

(3)

$$I^2C_{bitfrequency} = \frac{f_{PCLK}}{I2CSCLH + I2CSCLL}$$

The values for I2SCLL and I2SCLH should not necessarily be the same. Software can set different duty cycles on SCL by setting these two registers. For example, the I$^2$C bus specification defines the SCL low time and high time at different values for a 400 kHz I$^2$C rate. The value of the register must ensure that the data rate is in the I$^2$C data rate range of 0 through 400 kHz. Each register value must be greater than or equal to 4. Table 23–341 gives some examples of I$^2$C bus rates based on PCLK frequency BASE_IVNSS_CLK and I2SCLL and I2SCLH values.

**Table 341. Example I$^2$C clock rates**

| I2SCLL + I2SCLH | I$^2$C Bit Frequency (kHz) at PCLK (MHz) | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1** | **5** | **10** | **16** | **20** | **40** | **60** |
| 8 | 125 | | | | | | |
| 10 | 100 | | | | | | |
| 25 | 40 | 200 | 400 | | | | |
| 50 | 20 | 100 | 200 | 320 | 400 | | |
| 100 | 10 | 50 | 100 | 160 | 200 | 400 | |
| 160 | 6.25 | 31.25 | 62.5 | 100 | 125 | 250 | 375 |
| 200 | 5 | 25 | 50 | 80 | 100 | 200 | 300 |
| 400 | 2.5 | 12.5 | 25 | 40 | 50 | 100 | 150 |
| 800 | 1.25 | 6.25 | 12.5 | 20 | 25 | 50 | 75 |

## 8.7 I$^2$C Control Clear Register (I2C[0/1]CONCLR: 0xE008 2018, 0xE008 3018)

The I2CONCLR registers control clearing of bits in the I2CON register that controls operation of the I$^2$C interface. Writing a one to a bit of this register causes the corresponding bit in the I$^2$C control register to be cleared. Writing a zero has no effect.

**Table 342. I$^2$C Control Set Register (I2C[0/1]CONCLR - addresses 0xE008 2018, 0xE008 3018) bit description**

| Bit | Symbol | Description | Reset Value |
|---|---|---|---|
| 7 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 6 | I2ENC | I$^2$C interface Disable bit. | 0 |
| 5 | STAC | START flag 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 |
| 3 | SIC | I$^2$C interrupt Clear bit. | 0 |
| 2 | AAC | Assert acknowledge Clear bit. | |
| 1:0 | - | Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

**AAC** is the Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the I2CONSET register. Writing 0 has no effect.

**SIC** is the I$^2$C Interrupt Clear bit. Writing a 1 to this bit clears the SI bit in the I2CONSET register. Writing 0 has no effect.

**STAC** is the Start flag Clear bit. Writing a 1 to this bit clears the STA bit in the I2CONSET register. Writing 0 has no effect.

**I2ENC** is the I$^2$C Interface Disable bit. Writing a 1 to this bit clears the I2EN bit in the I2CONSET register. Writing 0 has no effect.

## 8.8 I$^2$C Monitor mode control register (I2C[0/1]MMCTRL: 0xE008 201C, 0xE008 301C)

This register controls the Monitor mode which allows the I$^2$C module to monitor traffic on the I$^2$C bus without actually participating in traffic or interfering with the I$^2$C bus.

**Table 343. I$^2$C Monitor mode control register (I2C[0/1]MMCTRL- addresses 0xE008 201C, 0xE008 301C) bit description**

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 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. | |
| 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] | |
| 0 | MM_ENA | | Monitor mode enable. | 0 |
| | | 0 | Monitor mode disabled. | |
| | | 1 | The I2C module will enter monitor mode. In this mode the SDA output will be forced high. This will prevent the I2C module from outputting data of any kind (including ACK) onto the I2C data bus. Depending on the state of the ENA_SCL bit, the output may be also forced high, preventing the module from having control over the I2C clock line. | |

[1] 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 23–8.9) 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).

### 8.8.1 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.

### 8.8.2 Loss of arbitration in Monitor mode

In monitor mode, the I2C module will not be able to respond to a request for information by the bus master or issue an ACK). Some other slave on the bus will respond instead. This will most probably result in a lost-arbitration state as far as our module is concerned.

Software should be aware of the fact that the module is in monitor mode and should not respond to any loss of arbitration state that is detected. In addition, hardware may be designed into the module to block some/all loss of arbitration states from occurring if those state would either prevent a desired interrupt from occurring or cause an unwanted interrupt to occur. Whether any such hardware will be added is still to be determined.

## 8.9 I$^2$C Data buffer register (I2C[0/1]DATA_BUFFER: 0xE008 202C, 0xE008 302C)

In monitor mode, the I2C module may lose the ability to stretch the clock (stall the bus) if the ENA_SCL bit is not set. This means that the processor will have a limited amount of time to read the contents of the data received on the bus. If the processor reads the I2DAT shift register, as it ordinarily would, it could have only one bit-time to respond to the interrupt before the received data is overwritten by new data.

To give the processor more time to respond, a new 8-bit, read-only DATA_BUFFER register will be added. The contents of the 8 msb’s 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. 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 I2DAT directly, as usual, and the behavior of I2DAT 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 344. I$^2$C Data buffer register (I2C[0/1]DATA_BUFFER addresses - 0xE008 202C, 0xE008 302C) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:0 | Data | This register holds contents of the 8 msb's of the I2DAT shift register. | 0 |

## 8.10 I$^2$C Slave Address registers (I2C[0/1]ADR0 to 3: 0xE008 20[0C, 20, 24, 28], 0xE008 30[0C, 20, 24, 28])

These registers are readable and writable and are only used when an I$^2$C interface is set to slave mode. In master mode, this register has no effect. The LSB of I2ADR is the general call bit. When this bit is set, the general call address (0x00) is recognized.

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

**Table 345. I$^2$C Slave Address registers (I2C[0/1]ADR0 to 3 - addresses 0xE008 20[0C, 20, 24, 28], 0xE008 30[0C, 20, 24, 28]) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 7:1 | Address | The I$^2$C device address for slave mode. | 0x00 |
| 0 | GC | General Call enable bit. | 0 |

## 8.11 I$^2$C Mask registers (I2C[0/1]MASK0 to 3: 0xE008 20[30, 34, 38, 3C], 0xE008 30[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 I2ADDRn register associated with that mask register. In other words, bits in an I2ADDRn 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 (I2DAT) to determine what the received address was that actually caused the match.

**Table 346. I$^2$C Mask registers (I2C[0/1]MASK0 to 3 - addresses 0xE008 20[30, 34, 38, 3C], 0xE008 30[30, 34, 38, 3C]) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 31:8 | - | Reserved. User software should not write ones to reserved bits. These bits read always back as 0's. | 0 |
| 7:1 | MASK | Mask bits. | 0x00 |
| 0 | - | Reserved. User software should not write ones to reserved bits. This bit reads always back as 0. | 0 |

## 9. Details of I$^2$C operating modes

The four operating modes are:

- Master Transmitter
- Master Receiver
- Slave Receiver
- Slave Transmitter

Data transfers in each mode of operation are shown in Figures 102 to 106. Table 23–347 lists abbreviations used in these figures when describing the I$^2$C operating modes.

**Table 347. Abbreviations used to describe an I$^2$C operation**

| Abbreviation | Explanation |
|---|---|
| S | Start Condition |
| SLA | 7 bit slave address |
| R | Read bit (high level at SDA) |
| W | Write bit (low level at SDA) |
| A | Acknowledge bit (low level at SDA) |
| $\overline{A}$ | Not acknowledge bit (high level at SDA) |
| Data | 8 bit data byte |
| P | Stop condition |

In Figures 102 to 106, circles are used to indicate when the serial interrupt flag is set. The numbers in the circles show the status code held in the I2STAT register. At these points, a service routine must be executed to continue or complete the serial transfer. These service routines are not critical since the serial transfer is suspended until the serial interrupt flag is cleared by software.

When a serial interrupt routine is entered, the status code in I2STAT is used to branch to the appropriate service routine. For each status code, the required software action and details of the following serial transfer are given in tables from Table 23–351 to Table 23–355.

### 9.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 23–102). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

**Table 348. I2CONSET used to initialize Master Transmitter mode**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Symbol | - | I2EN | STA | STO | SI | AA | - | - |
| Value | - | 1 | 0 | 0 | 0 | x | - | - |

The I$^2$C rate must also be configured in the I2SCLL and I2SCLH registers. I2EN must be set to logic 1 to enable the I$^2$C block. If the AA bit is reset, the I$^2$C block will not acknowledge its own slave address or the general call address in the event of another device becoming master of the bus. In other words, if AA is reset, the I$^2$C interface cannot enter a slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The I$^2$C logic will now test the I$^2$C bus and generate a start condition as soon as the bus becomes free. When a START condition is transmitted, the serial interrupt flag (SI) is set, and the status code in the status register (I2STAT) will be 0x08. This status code is used by the interrupt service routine to enter the appropriate state service routine that loads I2DAT with the slave address and the data direction bit (SLA+W). The SI bit in I2CON must then be reset before the serial transfer can continue.

When the slave address and the direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in I2STAT are possible. There are 0x18, 0x20, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = logic 1). The appropriate action to be taken for each of these status codes is detailed in Table 23–351. After a repeated start condition (state 0x10). The I$^2$C block may switch to the master receiver mode by loading I2DAT with SLA+R).

## 9.2 Master Receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see Figure 23–103). The transfer is initialized as in the master transmitter mode. When the start condition has been transmitted, the interrupt service routine must load I2DAT with the 7 bit slave address and the data direction bit (SLA+R). The SI bit in I2CON must then be cleared before the serial transfer can continue.

When the slave address and the data direction bit have been transmitted and an acknowledgment bit has been received, the serial interrupt flag (SI) is set again, and a number of status codes in I2STAT are possible. These are 0x40, 0x48, or 0x38 for the master mode and also 0x68, 0x78, or 0xB0 if the slave mode was enabled (AA = 1). The appropriate action to be taken for each of these status codes is detailed in Table 23–352. After a repeated start condition (state 0x10), the I$^2$C block may switch to the master transmitter mode by loading I2DAT with SLA+W.

## 9.3 Slave Receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 23–104). To initiate the slave receiver mode, I2ADR and I2CON must be loaded as follows:

**Table 349. 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$^2$C block will respond when addressed by a master. If the LSB (GC) is set, the I$^2$C block will respond to the general call address (0x00); otherwise it ignores the general call address.

**Table 350. 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$^2$C bus rate settings do not affect the I$^2$C block in the slave mode. I2EN must be set to logic 1 to enable the I$^2$C block. The AA bit must be set to enable the I$^2$C block to acknowledge its own slave address or the general call address. STA, STO, and SI must be reset.

When I2ADR and I2CON have been initialized, the I$^2$C block waits until it is addressed by its own slave address followed by the data direction bit which must be “0” (W) for the I$^2$C block to operate in the slave receiver mode. After its own slave address and the W bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from I2STAT. This status code is used to vector to a state service routine. The appropriate action to be taken for each of these status codes is detailed in Table 23–353. The slave receiver mode may also be entered if arbitration is lost while the I$^2$C block is in the master mode (see status 0x68 and 0x78).

If the AA bit is reset during a transfer, the I$^2$C block will return a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the I$^2$C block does not respond to its own slave address or a general call address. However, the I$^2$C bus is still monitored and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the I$^2$C block from the I$^2$C bus.

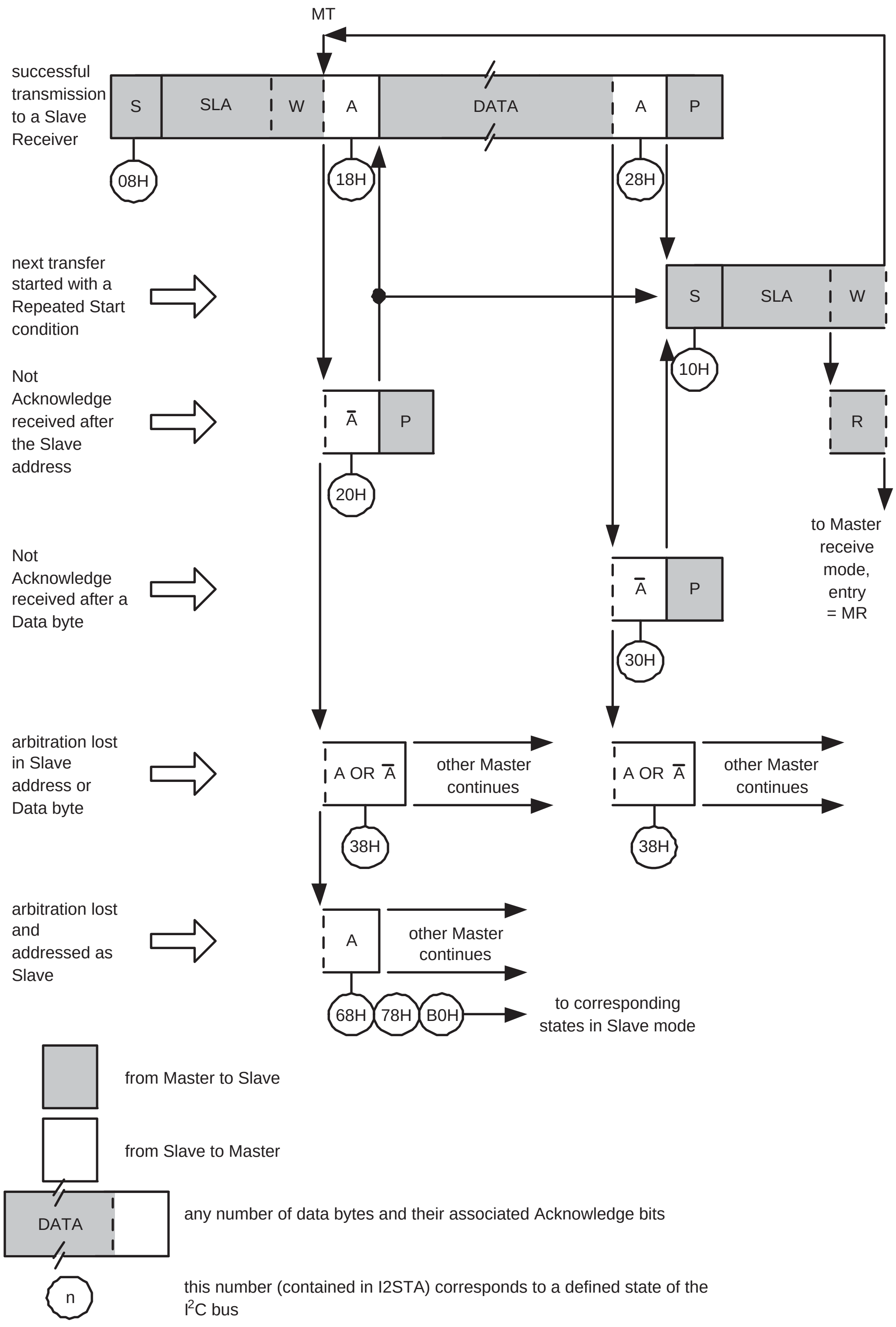

MT
successful transmission to a Slave Receiver
S
SLA
W
A
DATA
A
P
08H
18H
28H
next transfer started with a Repeated Start condition
S
SLA
W
10H
Not Acknowledge received after the Slave address
$\bar{A}$
P
20H
R
to Master receive mode, entry = MR
Not Acknowledge received after a Data byte
$\bar{A}$
P
30H
arbitration lost in Slave address or Data byte
A OR $\bar{A}$
other Master continues
38H
A OR $\bar{A}$
other Master continues
38H
arbitration lost and addressed as Slave
A
other Master continues
68H
78H
B0H
to corresponding states in Slave mode
from Master to Slave
from Slave to Master
DATA
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I²C bus

**Fig 102. Format and States in the Master Transmitter mode**

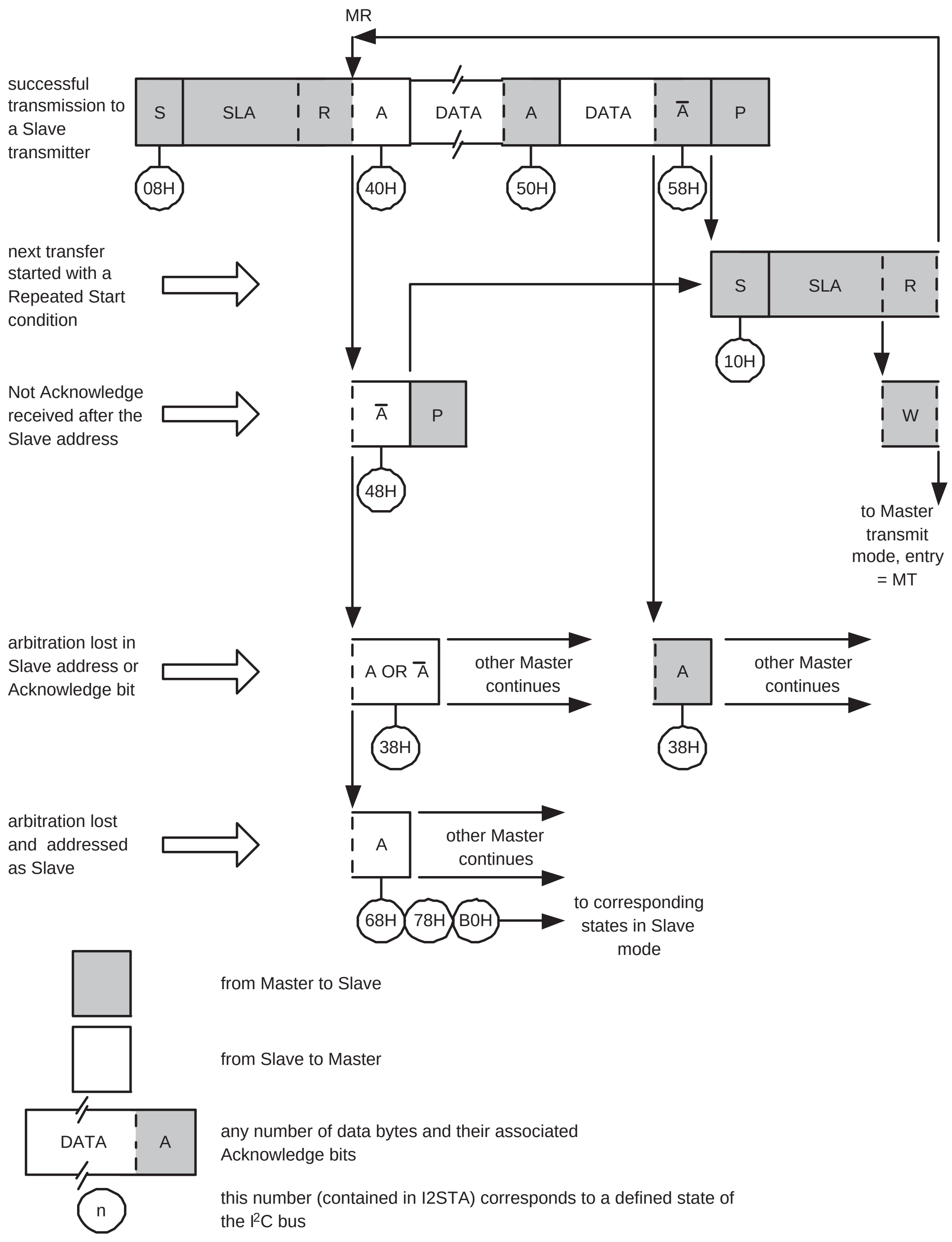

MR
successful transmission to a Slave transmitter
S
SLA
R
A
DATA
A
DATA
$\overline{A}$
P
08H
40H
50H
58H
next transfer started with a Repeated Start condition
S
SLA
R
10H
W
to Master transmit mode, entry = MT
Not Acknowledge received after the Slave address
$\overline{A}$
P
48H
arbitration lost in Slave address or Acknowledge bit
A OR $\overline{A}$
other Master continues
A
other Master continues
38H
38H
arbitration lost and addressed as Slave
A
other Master continues
68H
78H
B0H
to corresponding states in Slave mode
from Master to Slave
from Slave to Master
DATA
A
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I$^2$C bus

**Fig 103. Format and States in the Master Receiver mode**

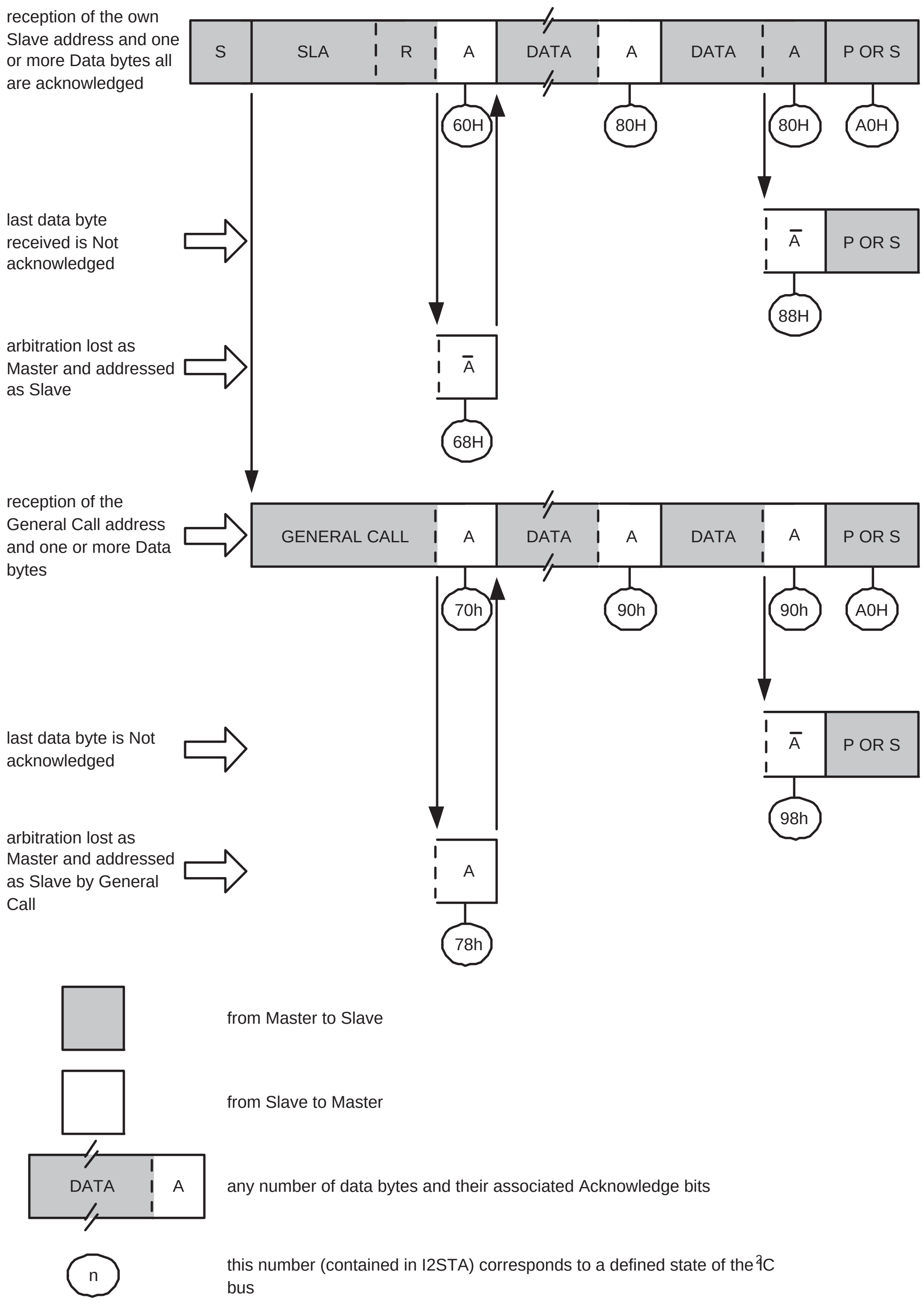

reception of the own Slave address and one or more Data bytes all are acknowledged
S
SLA
R
A
DATA
A
DATA
A
P OR S
60H
80H
80H
A0H
last data byte received is Not acknowledged
$\overline{A}$
P OR S
88H
arbitration lost as Master and addressed as Slave
$\overline{A}$
68H
reception of the General Call address and one or more Data bytes
GENERAL CALL
A
DATA
A
DATA
A
P OR S
70h
90h
90h
A0H
last data byte is Not acknowledged
$\overline{A}$
P OR S
98h
arbitration lost as Master and addressed as Slave by General Call
A
78h
from Master to Slave
from Slave to Master
DATA
A
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I²C bus

Fig 104. Format and States in the Slave Receiver mode

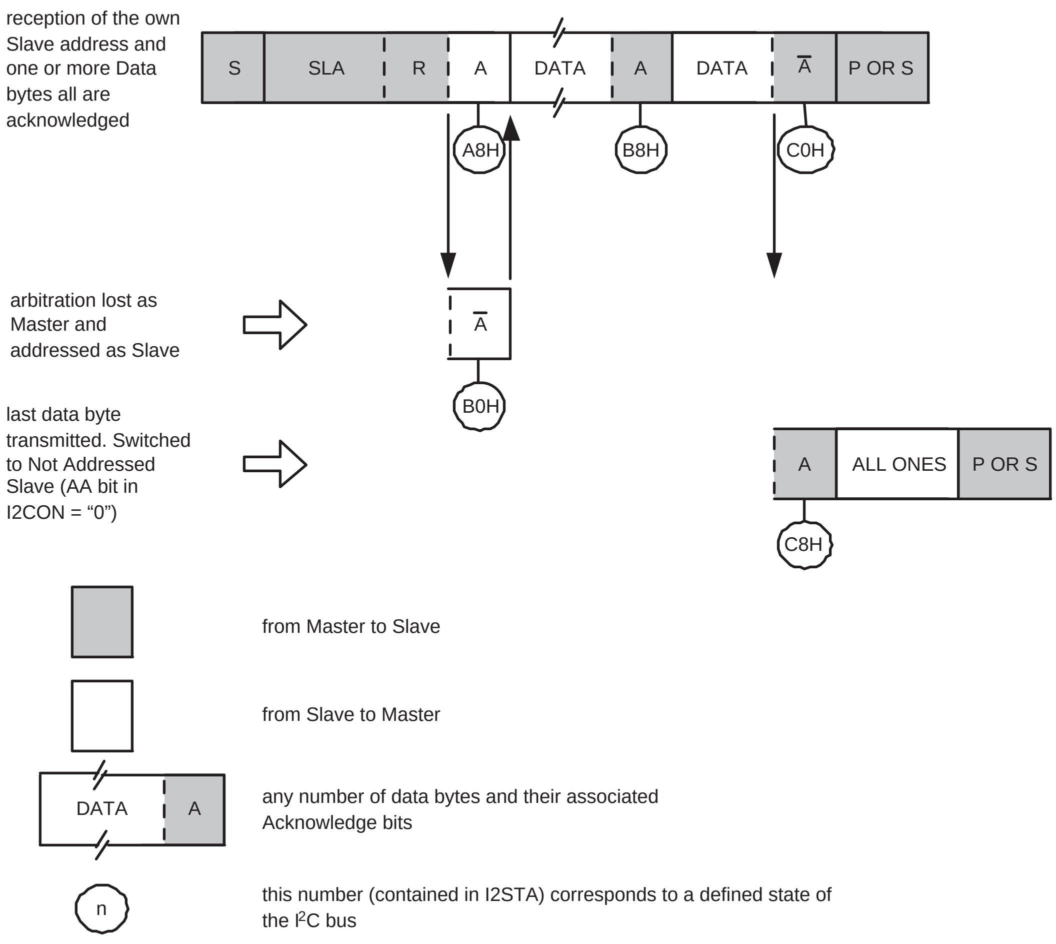

reception of the own Slave address and one or more Data bytes all are acknowledged
S
SLA
R
A
DATA
A
DATA
Ā
P OR S
A8H
B8H
C0H
arbitration lost as Master and addressed as Slave
Ā
B0H
last data byte transmitted. Switched to Not Addressed Slave (AA bit in I2CON = "0")
A
ALL ONES
P OR S
C8H
from Master to Slave
from Slave to Master
DATA
A
any number of data bytes and their associated Acknowledge bits
n
this number (contained in I2STA) corresponds to a defined state of the I²C bus

**Fig 105. Format and States in the Slave Transmitter mode**

## 9.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 23–105). Data transfer is initialized as in the slave receiver mode. When I2ADR and I2CON have been initialized, the I$^2$C block waits until it is addressed by its own slave address followed by the data direction bit which must be "1" (R) for the I$^2$C block to operate in the slave transmitter mode. After its own slave address and the R bit have been received, the serial interrupt flag (SI) is set and a valid status code can be read from I2STAT. This status code is used to vector to a state service routine, and the appropriate action to be taken for each of these status codes is detailed in Table 23–354. The slave transmitter mode may also be entered if arbitration is lost while the I$^2$C block is in the master mode (see state 0xB0).

If the AA bit is reset during a transfer, the I$^2$C block will transmit the last byte of the transfer and enter state 0xC0 or 0xC8. The I$^2$C block is switched to the not addressed slave mode and will ignore the master receiver if it continues the transfer. Thus the master receiver receives all 1s as serial data. While AA is reset, the I$^2$C block does not respond to its own slave address or a general call address. However, the I$^2$C bus is still monitored, and address recognition may be resumed at any time by setting AA. This means that the AA bit may be used to temporarily isolate the I$^2$C block from the I$^2$C bus.

**Table 351. Master Transmitter mode**

| Status Code (I2CSTAT) | Status of the I$^2$C bus and hardware | Application software response | | | | | Next action taken by I$^2$C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From I2DAT | To I2CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x08 | A START condition has been transmitted. | Load SLA+W Clear STA | X | 0 | 0 | X | SLA+W will be transmitted; ACK bit will be received. |
| 0x10 | A repeated START condition has been transmitted. | Load SLA+W or | X | 0 | 0 | X | As above. |
| | | Load SLA+R Clear STA | X | 0 | 0 | X | SLA+W will be transmitted; the I$^2$C block will be switched to MST/REC mode. |
| 0x18 | SLA+W has been transmitted; ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No I2DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No I2DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No I2DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x20 | SLA+W has been transmitted; NOT ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No I2DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No I2DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No I2DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x28 | Data byte in I2DAT has been transmitted; ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No I2DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No I2DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No I2DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x30 | Data byte in I2DAT has been transmitted; NOT ACK has been received. | Load data byte or | 0 | 0 | 0 | X | Data byte will be transmitted; ACK bit will be received. |
| | | No I2DAT action or | 1 | 0 | 0 | X | Repeated START will be transmitted. |
| | | No I2DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No I2DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x38 | Arbitration lost in SLA+R/W or Data bytes. | No I2DAT action or | 0 | 0 | 0 | X | I$^2$C bus will be released; not addressed slave will be entered. |
| | | No I2DAT action | 1 | 0 | 0 | X | A START condition will be transmitted when the bus becomes free. |

Table 352. Master Receiver mode

| Status Code (I2CSTAT) | Status of the I$^2$C bus and hardware | Application software response | | | | | Next action taken by I$^2$C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From I2DAT | To I2CON | | | | |
| | | | 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$^2$C block will be switched to MST/TRX mode. |
| 0x38 | Arbitration lost in NOT ACK bit. | No I2DAT action or | 0 | 0 | 0 | X | I$^2$C bus will be released; the I$^2$C block will enter a slave mode. |
| | | No I2DAT action | 1 | 0 | 0 | X | A START condition will be transmitted when the bus becomes free. |
| 0x40 | SLA+R has been transmitted; ACK has been received. | No I2DAT action or | 0 | 0 | 0 | 0 | Data byte will be received; NOT ACK bit will be returned. |
| | | No I2DAT action | 0 | 0 | 0 | 1 | Data byte will be received; ACK bit will be returned. |
| 0x48 | SLA+R has been transmitted; NOT ACK has been received. | No I2DAT action or | 1 | 0 | 0 | X | Repeated START condition will be transmitted. |
| | | No I2DAT action or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | No I2DAT action | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |
| 0x50 | Data byte has been received; ACK has been returned. | Read data byte or | 0 | 0 | 0 | 0 | Data byte will be received; NOT ACK bit will be returned. |
| | | Read data byte | 0 | 0 | 0 | 1 | Data byte will be received; ACK bit will be returned. |
| 0x58 | Data byte has been received; NOT ACK has been returned. | Read data byte or | 1 | 0 | 0 | X | Repeated START condition will be transmitted. |
| | | Read data byte or | 0 | 1 | 0 | X | STOP condition will be transmitted; STO flag will be reset. |
| | | Read data byte | 1 | 1 | 0 | X | STOP condition followed by a START condition will be transmitted; STO flag will be reset. |

**Table 353. Slave Receiver Mode**

| Status Code (I2CSTAT) | Status of the $I^2C$ bus and hardware | Application software response | | | | | Next action taken by $I^2C$ hardware |
|---|---|---|---|---|---|---|---|
| | | To/From I2DAT | To I2CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x60 | Own SLA+W has been received; ACK has been returned. | No I2DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No I2DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x68 | Arbitration lost in SLA+R/W as master; Own SLA+W has been received, ACK returned. | No I2DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No I2DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x70 | General call address (0x00) has been received; ACK has been returned. | No I2DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No I2DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x78 | Arbitration lost in SLA+R/W as master; General call address has been received, ACK has been returned. | No I2DAT action or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | No I2DAT action | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |
| 0x80 | Previously addressed with own 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 I2ADR[0] = logic 1. |
| | | Read data byte or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | Read data byte | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0x90 | Previously addressed with General Call; DATA byte has been received; ACK has been returned. | Read data byte or | X | 0 | 0 | 0 | Data byte will be received and NOT ACK will be returned. |
| | | Read data byte | X | 0 | 0 | 1 | Data byte will be received and ACK will be returned. |

Table 353. Slave Receiver Mode

| Status Code (I2CSTAT) | Status of the I$^2$C bus and hardware | Application software response | | | | | Next action taken by I$^2$C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From I2DAT | To I2CON | | | | |
| | | | STA | STO | SI | AA | |
| 0x98 | Previously addressed with General Call; DATA byte has been received; NOT ACK has been returned. | Read data byte or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | Read data byte or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. |
| | | Read data byte or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | Read data byte | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0xA0 | A STOP condition or repeated START condition has been received while still addressed as 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 I2ADR[0] = logic 1. |
| | | No STDAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No STDAT action | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |

**Table 354. Tad_105: Slave Transmitter mode**

| Status Code (I2CSTAT) | Status of the I$^2$C bus and hardware | Application software response | | | | | Next action taken by I$^2$C hardware |
|---|---|---|---|---|---|---|---|
| | | To/From I2DAT | To I2CON | | | | |
| | | | STA | STO | SI | AA | |
| 0xA8 | Own SLA+R has been received; ACK has been returned. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK will be received. |
| 0xB0 | Arbitration lost in SLA+R/W as master; Own SLA+R has been received, ACK has been returned. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK bit will be received. |
| 0xB8 | Data byte in I2DAT has been transmitted; ACK has been received. | Load data byte or | X | 0 | 0 | 0 | Last data byte will be transmitted and ACK bit will be received. |
| | | Load data byte | X | 0 | 0 | 1 | Data byte will be transmitted; ACK bit will be received. |
| 0xC0 | Data byte in I2DAT has been transmitted; NOT ACK has been received. | No I2DAT action or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | No I2DAT action or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. |
| | | No I2DAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No I2DAT action | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. A START condition will be transmitted when the bus becomes free. |
| 0xC8 | Last data byte in I2DAT has been transmitted (AA = 0); ACK has been received. | No I2DAT action or | 0 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. |
| | | No I2DAT action or | 0 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR[0] = logic 1. |
| | | No I2DAT action or | 1 | 0 | 0 | 0 | Switched to not addressed SLV mode; no recognition of own SLA or General call address. A START condition will be transmitted when the bus becomes free. |
| | | No I2DAT action | 1 | 0 | 0 | 1 | Switched to not addressed SLV mode; Own SLA will be recognized; General call address will be recognized if I2ADR.0 = logic 1. A START condition will be transmitted when the bus becomes free. |

## 9.5 Miscellaneous states

There are two I2STAT codes that do not correspond to a defined I$^2$C hardware state (see Table 23–355). These are discussed below.

### 23.9.5.1 I2STAT = 0xF8

This status code indicates that no relevant information is available because the serial interrupt flag, SI, is not yet set. This occurs between other states and when the I$^2$C block is not involved in a serial transfer.

### 23.9.5.2 I2STAT = 0x00

This status code indicates that a bus error has occurred during an I$^2$C 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$^2$C block signals. When a bus error occurs, SI is set. To recover from a bus error, the STO flag must be set and SI must be cleared. This causes the I$^2$C block to enter the “not addressed” slave mode (a defined state) and to clear the STO flag (no other bits in I2CON are affected). The SDA and SCL lines are released (a STOP condition is not transmitted).

Table 355. Miscellaneous states

| Status Code (I2CSTAT) | Status of the I$^2$C bus and hardware | Application software response |  |  |  |  | Next action taken by I$^2$C hardware |
|---|---|---|---|---|---|---|---|
|  |  | To/From I2DAT | To I2CON |  |  |  |  |
|  |  |  | STA | STO | SI | AA |  |
| 0xF8 | No relevant state information available; SI = 0. | No I2DAT action | No I2CON action |  |  |  | Wait or proceed current transfer. |
| 0x00 | Bus error during MST or selected slave modes, due to an illegal START or STOP condition. State 0x00 can also occur when interference causes the I$^2$C block to enter an undefined state. | No I2DAT action | 0 | 1 | 0 | X | Only the internal hardware is affected in the MST or addressed SLV modes. In all cases, the bus is released and the I$^2$C block is switched to the not addressed SLV mode. STO is reset. |

## 9.6 Some special cases

The I$^2$C hardware has facilities to handle the following special cases that may occur during a serial transfer:

## 9.7 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 23–106). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

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

## 9.8 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see Figure 23–100). Loss of arbitration is indicated by the following states in I2STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 23–102 and Figure 23–103).

If the STA flag in I2CON is set by the routines which service these states, then, if the bus is free again, a START condition (state 0x08) is transmitted without intervention by the CPU, and a retry of the total serial transfer can commence.

## 9.9 Forced access to the I$^2$C bus

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

If an uncontrolled source generates a superfluous START or masks a STOP condition, then the I$^2$C bus stays busy indefinitely. If the STA flag is set and bus access is not obtained within a reasonable amount of time, then a forced access to the I$^2$C bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I$^2$C hardware behaves as if a STOP condition was received and is able to transmit a START condition. The STO flag is cleared by hardware (see Figure 23–107).

## 9.10 I$^2$C bus obstructed by a LOW level on SCL or SDA

An I$^2$C bus hang-up can occur if either the SDA or SCL line is held LOW by any device on the bus. If the SCL line is obstructed (pulled LOW) by a device on the bus, no further serial transfer is possible, and the problem must be resolved by the device that is pulling the SCL bus line LOW.

Typically, the SDA line may be obstructed by another device on the bus that has become out of synchronization with the current bus master by either missing a clock, or by sensing a noise pulse as a clock. In this case, the problem can be solved by transmitting additional clock pulses on the SCL line (see Figure 23–108). The I$^2$C interface does not include a dedicated timeout timer to detect an obstructed bus, but this can be implemented using another timer in the system. When detected, software can force clocks (up to 9 may be required) on SCL until SDA is released by the offending device. At that point, the slave may still be out of synchronization, so a START should be generated to insure that all I$^2$C peripherals are synchronized.

## 9.11 Bus error

A bus error occurs when a START or STOP condition is present at an illegal position in the format frame. Examples of illegal positions are during the serial transfer of an address byte, a data bit, or an acknowledge bit.

The I$^2$C hardware only reacts to a bus error when it is involved in a serial transfer either as a master or an addressed slave. When a bus error is detected, the I$^2$C block immediately switches to the not addressed slave mode, releases the SDA and SCL lines, sets the interrupt flag, and loads the status register with 0x00. This status code may be used to vector to a state service routine which either attempts the aborted serial transfer again or simply recovers from the error condition as shown in Table 23–355.

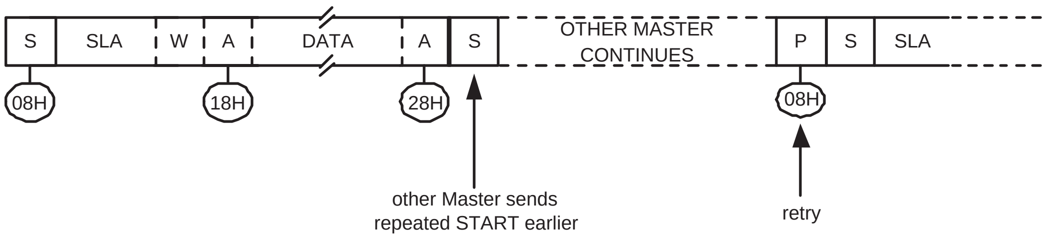


**Fig 106. Simultaneous repeated START conditions from 2 masters**

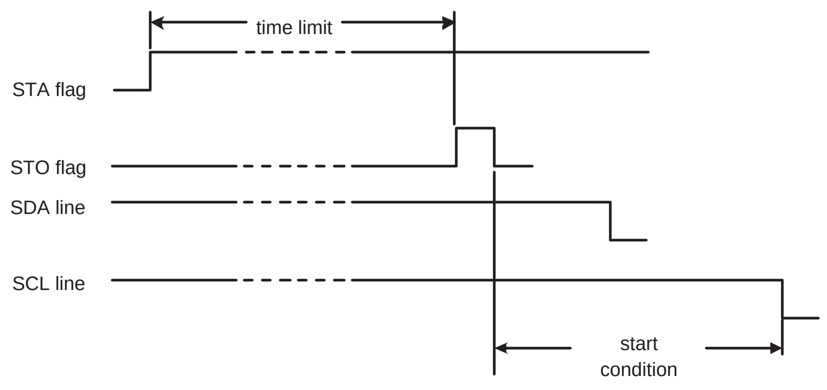
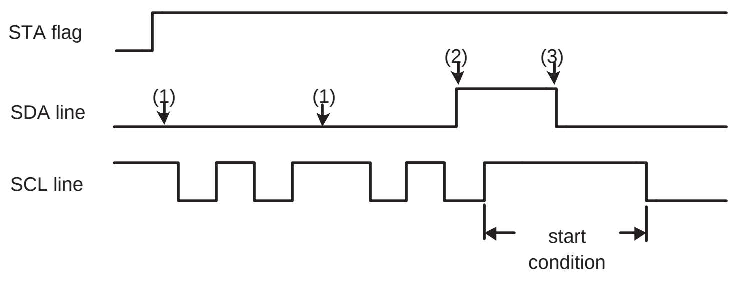


**Fig 107. Forced access to a busy I$^2$C bus**



(1) Unsuccessful attempt to send a start condition.

(2) SDA line is released.

(3) Successful attempt to send a start condition. State 08H is entered.

**Fig 108. Recovering from a bus obstruction caused by a low level on SDA**

## 9.12 I$^2$C State service routines

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

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

### 9.12.1 Initialization

In the initialization example, the I$^2$C block is enabled for both master and slave modes. For each mode, a buffer is used for transmission and reception. The initialization routine performs the following functions:

- I2ADR is loaded with the part's own slave address and the general call bit (GC).
- The I$^2$C interrupt enable and interrupt priority bits are set.
- The slave mode is enabled by simultaneously setting the I2EN and AA bits in I2CON and the serial clock frequency (for master modes) is defined by loading CR0 and CR1 in I2CON. 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 I2STAT is loaded with the appropriate state information.

### 9.12.2 $I^2C$ interrupt service

When the $I^2C$ interrupt is entered, I2STAT contains a status code which identifies one of the 26 state services to be executed.

### 9.12.3 The state service routines

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

### 9.12.4 Adapting state services to an application

The state service examples show the typical actions that must be performed in response to the 26 $I^2C$ 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.

## 10. Software example

### 10.1 Initialization routine

Example to initialize $I^2C$ Interface as a Slave and/or Master.

1. Load I2ADR with own Slave Address, enable general call recognition if needed.
2. Enable $I^2C$ interrupt.
3. Write 0x44 to I2CONSET to set the I2EN and AA bits, enabling Slave functions. For Master only functions, write 0x40 to I2CONSET.

### 10.2 Start master transmit function

Begin a Master Transmit operation by setting up the buffer, pointer, and data count, then initiating a Start.

1. Initialize Master data counter.
2. Set up the Slave Address to which data will be transmitted, and add the Write bit.
3. Write 0x20 to I2CONSET to set the STA bit.
4. Set up data to be transmitted in Master Transmit buffer.
5. Initialize the Master data counter to match the length of the message being sent.
6. Exit

### 10.3 Start master receive function

Begin a Master Receive operation by setting up the buffer, pointer, and data count, then initiating a Start.

1. Initialize Master data counter.

2. Set up the Slave Address to which data will be transmitted, and add the Read bit.
3. Write 0x20 to I2CONSET to set the STA bit.
4. Set up the Master Receive buffer.
5. Initialize the Master data counter to match the length of the message to be received.
6. Exit

## 10.4 I$^2$C interrupt routine

Determine the I$^2$C state and which state routine will be used to handle it.

1. Read the I$^2$C status from I2STA.
2. Use the status value to branch to one of 26 possible state routines.

## 10.5 Non mode specific states

### 10.5.1 State : 0x00

Bus Error. Enter not addressed Slave mode and release bus.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

## 10.6 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.

### 10.6.1 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 I2DAT.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Master Transmit mode data buffer.
5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

### 10.6.2 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 I2DAT.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.

4. Set up Master Transmit mode data buffer.
5. Set up Master Receive mode data buffer.
6. Initialize Master data counter.
7. Exit

## 10.7 Master Transmitter states

### 10.7.1 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 I2DAT with first data byte from Master Transmit buffer.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment Master Transmit buffer pointer.
5. Exit

### 10.7.2 State : 0x20

Slave Address + Write has been transmitted, NOT ACK has been received. A Stop condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 10.7.3 State : 0x28

Data has been transmitted, ACK has been received. If the transmitted data was the last data byte then transmit a Stop condition, otherwise transmit the next data byte.

1. Decrement the Master data counter, skip to step 5 if not the last data byte.
2. Write 0x14 to I2CONSET to set the STO and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Exit
5. Load I2DAT with next data byte from Master Transmit buffer.
6. Write 0x04 to I2CONSET to set the AA bit.
7. Write 0x08 to I2CONCLR to clear the SI flag.
8. Increment Master Transmit buffer pointer
9. Exit

### 10.7.4 State : 0x30

Data has been transmitted, NOT ACK received. A Stop condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 10.7.5 State : 0x38

Arbitration has been lost during Slave Address + Write or data. The bus has been released and not addressed Slave mode is entered. A new Start condition will be transmitted when the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 10.8 Master Receive states

#### 10.8.1 State : 0x40

Previous state was State 08 or State 10. Slave Address + Read has been transmitted, ACK has been received. Data will be

received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 10.8.2 State : 0x48

Slave Address + Read has been transmitted, NOT ACK has been received. A Stop condition will be transmitted.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

#### 10.8.3 State : 0x50

Data has been received, ACK has been returned. Data will be read from I2DAT. Additional data will be received. If this is the last data byte then NOT ACK will be returned, otherwise ACK will be returned.

1. Read data byte from I2DAT into Master Receive buffer.
2. Decrement the Master data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
4. Exit
5. Write 0x04 to I2CONSET to set the AA bit.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment Master Receive buffer pointer
8. Exit

#### 10.8.4 State : 0x58

Data has been received, NOT ACK has been returned. Data will be read from I2DAT. A Stop condition will be transmitted.

1. Read data byte from I2DAT into Master Receive buffer.
2. Write 0x14 to I2CONSET to set the STO and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Exit

## 10.9 Slave Receiver states

### 10.9.1 State : 0x60

Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 10.9.2 State : 0x68

Arbitration has been lost in Slave Address and R/W bit as bus Master. Own Slave Address + Write has been received, ACK has been returned. Data will be received and ACK will be returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit.

### 10.9.3 State : 0x70

General call has been received, ACK has been returned. Data will be received and ACK returned.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 10.9.4 State : 0x78

Arbitration has been lost in Slave Address + R/W bit as bus Master. General call has been received and ACK has been returned. Data will be received and ACK returned. STA is set to restart Master mode after the bus is free again.

1. Write 0x24 to I2CONSET to set the STA and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.

3. Set up Slave Receive mode data buffer.
4. Initialize Slave data counter.
5. Exit

### 10.9.5 State : 0x80

Previously addressed with own Slave Address. Data has been received and ACK has been returned. Additional data will be read.

1. Read data byte from I2DAT into the Slave Receive buffer.
2. Decrement the Slave data counter, skip to step 5 if not the last data byte.
3. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
4. Exit.
5. Write 0x04 to I2CONSET to set the AA bit.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment Slave Receive buffer pointer.
8. Exit

### 10.9.6 State : 0x88

Previously addressed with own Slave Address . Data has been received and NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 10.9.7 State : 0x90

Previously addressed with general call. Data has been received, ACK has been returned. Received data will be saved. Only the first data byte will be received with ACK. Additional data will be received with NOT ACK.

1. Read data byte from I2DAT into the Slave Receive buffer.
2. Write 0x0C to I2CONCLR to clear the SI flag and the AA bit.
3. Exit

### 10.9.8 State : 0x98

Previously addressed with general call. Data has been received, NOT ACK has been returned. Received data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

### 10.9.9 State : 0xA0

A Stop condition or repeated Start has been received, while still addressed as a Slave. Data will not be saved. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

## 10.10 Slave Transmitter States

### 10.10.1 State : 0xA8

Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received.

1. Load I2DAT from Slave Transmit buffer with first data byte.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

### 10.10.2 State : 0xB0

Arbitration lost in Slave Address and R/W bit as bus Master. Own Slave Address + Read has been received, ACK has been returned. Data will be transmitted, ACK bit will be received. STA is set to restart Master mode after the bus is free again.

1. Load I2DAT from Slave Transmit buffer with first data byte.
2. Write 0x24 to I2CONSET to set the STA and AA bits.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Set up Slave Transmit mode data buffer.
5. Increment Slave Transmit buffer pointer.
6. Exit

### 10.10.3 State : 0xB8

Data has been transmitted, ACK has been received. Data will be transmitted, ACK bit will be received.

1. Load I2DAT from Slave Transmit buffer with data byte.
2. Write 0x04 to I2CONSET to set the AA bit.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment Slave Transmit buffer pointer.
5. Exit

### 10.10.4 State : 0xC0

Data has been transmitted, NOT ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.

3. Exit

### 10.10.5 State : 0xC8

The last data byte has been transmitted, ACK has been received. Not addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set the AA bit.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit

# UM10316

# Chapter 24: LPC29xx Modulation and Sampling Control Subsystem (MSCSS)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. Note that the ADC0 is available on LPC2926/27/29, LPC2930, and LPC2939 only. External start inputs for ADC1 are not pinned out on the LPC2927/29 and LPC2921/23/25.

## 2. MSCSS functional description

The modulation and sampling control subsystem (MSCSS) is a module provided with ADCs and PWMs. The ADCs can be used to measure voltages while the PWMs can be used to create various square waveforms or to capture strobes. The combination of the blocks can be used to control motors.

Three functional blocks in the MSCSS (ADC, PWM, QEI) are described in the following chapters. The MSCSS timers are functionally identical to the general timers and are described in Section 17–2.

The ADCs and PWMs are provided with several trigger inputs and outputs. Two timers are available for synchronization of all actions.

A complete overview of the synchronization and trigger mechanism is shown in Figure 24–109 below:

### 2.1 Synchronization and trigger features of the MSCSS

The MSCSS contains two internal timers to generate synchronization and carrier pulses for the ADCs and PWMs. Figure 24–109 shows how the timers are connected to the ADC and PWM modules.

Each ADC module has four start inputs. An ADC conversion is triggered when one of the start ADC conditions is valid:

- start 0: ADC external start input pin; can be triggered at a positive or negative edge. Note that this signal is captured in the ADC clock domain
- start 1: If the ‘preceding’ ADC conversion is ended, the sync_out signal starts an ADC conversion. This signal is captured in the MSCSS subsystem clock domain. As can be seen in Figure 24–109, the sync_out of ADC1 is connected to the start 1 input of ADC2 and the sync_out of ADC2 is connected to the start 1 input of ADC1.
- start 2: The PWM sync_out can start an ADC conversion. The sync_out signal is synchronized to the ADC clock in the ADC module. This signal is captured in the MSCSS subsystem clock domain.
- start 3: The match outputs from MSCSS timer 0 are connected to the start 3 inputs of the ADCs. This signal is captured in the ADC clock domain.

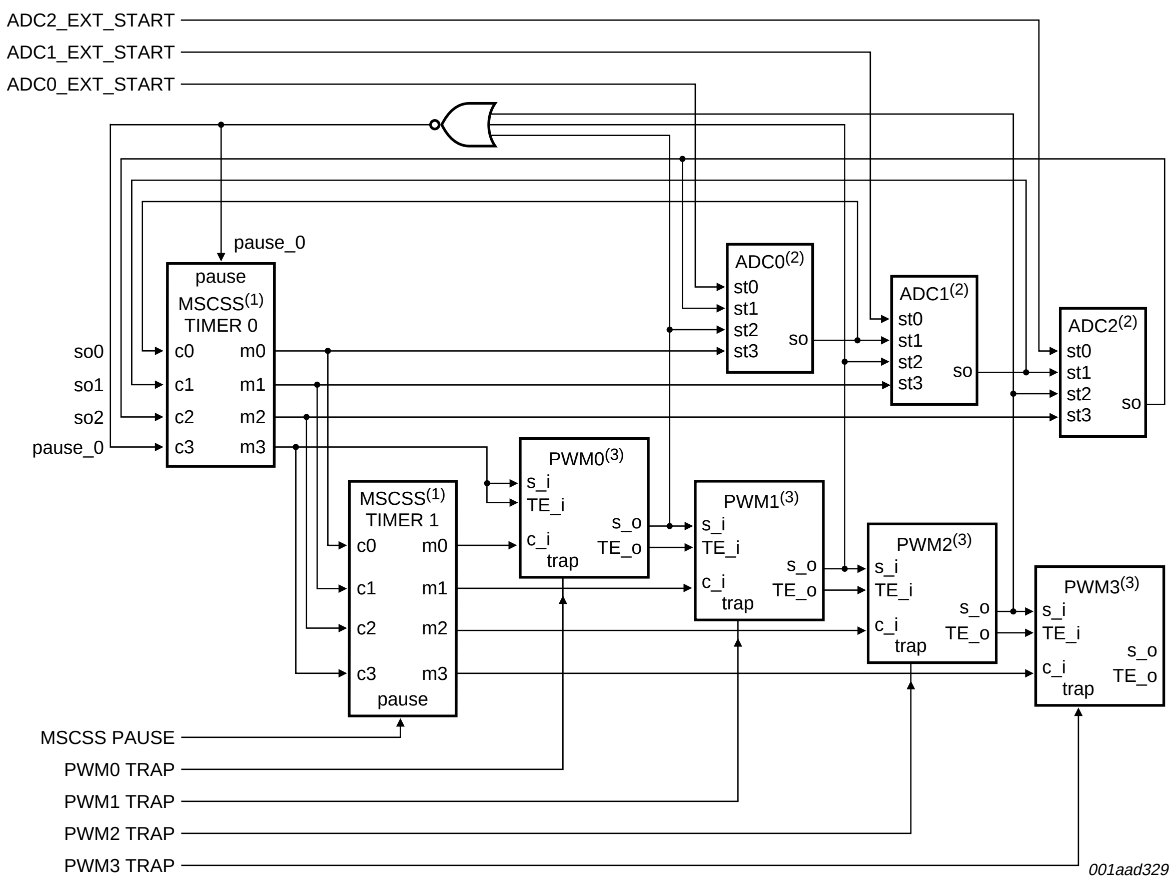
The PWM_sync and trans_enable_in of PWM 0 are connected to the 4th match output of MSCSS timer 0 to start the PWM after a pre-programmed delay. This sync signal is cascaded through all PWMs, allowing a programmable delay offset between subsequent PWMs. The sync delay of each PWM can be programmed synchronously or with a different phase for spreading the power load.

The match outputs of MSCSS timer 1 (PWM control) are connected to the corresponding carrier inputs of the PWM modules. The carrier signal is modulated with the PWM-generated waveforms.

The pause input of MSCSS timer 1 (PWM Control) is connected to an external input pin. Generation of the carrier signal is stopped by asserting the pause of this timer.

The pause input of MSCSS timer 0 (ADC Control) is connected to a ‘NOR’ of the PWM_sync outputs (start 2 input on the ADCs). If the pause feature of this timer is enabled the timer only counts when one of the PWM_sync outputs is active HIGH. This feature can be used to start the ADC once every x PWM cycles, where x corresponds to the value in the match register of the timer. In this case the start 3 input of the ADC should be enabled (start on match output of MSCSS timer 0).

The signals connected to the capture inputs of the timers (both MSCSS timer 0 and MSCSS timer 1) are intended for debugging.



(1) Timers:

c0 to c3 = capture in 0 to capture in 3

m0 to m3 = match out 0 to match out 3

(2) ADCs:

st0 to st3 = start 0 to start 3 inputs

s0 to s3 = sync_out 0 to sync_out 3

(3) PWMs:

c_i = carrier in

s_i = sync_in

s_o = sync_out

TE_i = trans_enable_in

TE_o = trans_enable_out

**Fig 109. MSCSS block diagram**

## 3. MSCSS miscellaneous operations

The MSCSS is equipped with several functions which are useful in a wide range of applications:

- Voltage monitoring
- Autonomous voltage-threshold monitoring
- PWM generation on a configurable number of outputs

- Capture inputs to measure time between events
- Synchronization between several PWM signals
- Synchronization between ADCs
- Synchronization between PWMs and ADCs
- Timed and synchronized renewal of settings
- Interrupt generation for a wide range of events

In the next paragraphs, a number of possible applications are described.

## 3.1 Continuous level measurement

To measure voltages with constant intervals, MSCSS_Timer0 can be configured to generate pulses with constant intervals. Using the match outputs and trigger inputs of the ADC, these pulses can be used to start the ADC. If the ADC is configured to run once on the trigger it delivers the output value after conversion and waits for the next trigger.

**Remark:** There is no built-in mechanism to avoid over-triggering of the ADC, so the user has to configure the triggers in such a way that the triggering rate does not exceed the capabilities of the ADC. Note also that is possible to start the second ADC with the sync_out of the first ADC, but since sync_out is triggered on scan-complete this is not the recommended method.

## 3.2 Comparator functionality

Since the ADC has a compare functionality it can run continuously without reading the sampled values, but as soon as the sampled value exceeds a certain threshold an interrupt is generated to the processor. This reduces load on the processor since no polling loop needed.

The sample speed can be set by adjusting the parameter adc_clk or by using MSCSS_Timer0 to trigger the ADC. With MSCSS_Timer0 it is possible to set up the sample speed for each ADC individually: adc_clk influences all ADCs.

**Remark:** The above comparison is done on unfiltered data, so it can be inaccurate if no external filtering is done on the analog signal.

## 3.3 Dimmer using PWMs

The PWM can be used as a dimmer for lighting for example. Since the voltage produced by the generator of the car varies with the load of the engine and load of the generator, the illumination varies accordingly. Voltage on the power lines for lighting can thus be steered using the PWM to have a constant illumination, or adjusted to certain levels.

Not applying the full power of the generator to the lighting bulbs but a constant 12 V instead considerably extends bulb lifetime.

For checking the voltage the ADCs can be used, as in the level-monitoring described in Section 24–3.2.

## 3.4 Generating sine waves

For several applications it can be useful to generate a sine wave without having a high processor load.

To generate a sine wave, the sine period can be divided into N periods. In each of these a fixed voltage is generated via PWM0, having a much shorter period compared to the sine itself. Output PWM0_0 steers positive voltage; PWM0_1 steers the negative voltage.

Along with the high-frequency PWM0, MSCSS_Timer0 defines the period for the N intervals. On the match output of MSCSS_Timer0, the trans_en_in and sync of PWM0 are triggered and the new pre-loaded value for PWM0 is activated.

Pre-loading new values into the registers can be done on the interrupts indicating that the values are activated. The pre-loading must be finished before the next pulse. To avoid switching noise, the period of MSCSS_Timer0 has to be an interval times the period of PWM0.

If a capacitor is attached to the outputs, a sine can be generated.

## 4. Register overview

See Table 24–356 for timer, ADC, PWM, and QEI registers.

Table 356. MSCSS register base addresses

| Functional block | Base address | Reference |
|---|---|---|
| MSCSS timer 0 | 0xE00C 0000 | Section 17–5 |
| MSCSS timer 1 | 0xE00C 1000 | Section 17–5 |
| ADC0 | 0xE00C 2000 | Section 26–4 |
| ADC1 | 0xE00C 3000 | Section 26–4 |
| ADC2 | 0xE00C 4000 | Section 26–4 |
| PWM0 | 0xE00C 5000 | Section 25–5 |
| PWM1 | 0xE00C 6000 | Section 25–5 |
| PWM2 | 0xE00C 7000 | Section 25–5 |
| PWM3 | 0xE00C 8000 | Section 25–5 |
| QEI | 0xE00C 9000 | Section 27–6 |

# UM10316

# Chapter 25: LPC29xx Pulse Width Modulator (PWM)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

The MSCSS contains four PWM blocks to allow generation and capture of all kinds of square waveforms. The main features of a PWM block are:

- Six pulse-width modulated output signals
- Optional interrupt generation on match (each edge)
- Different operation modes: continuous and run-once
- 16-bit PWM counter and 16-bit prescale counter allowing large range of PWM periods
- A protective mode (TRAP) holding the output in a software-controllable state and with optional interrupt generation on a trap event
- Four capture registers and three capture trigger pins with optional interrupt generation on a capture event
- Interrupt generation on a match event, capture event, PWM counter overflow or trap event
- A burst mode mixing an external carrier signal with an internally generated PWM
- Programmable sync delay output to trigger other PWM modules (master/slave behavior)

**Remark:** Capture channels two and three share an external pin.

## 3. Shadow registers and related update mechanism

It is possible to reconfigure the PWM outputs ‘on the fly’ changing frequency, cycle period or duty cycles by enabling new sets of parameter values. A mechanism is provided to allow an atomic change of the PWM configuration without disturbing the currently generated PWM outputs. Therefore two sets are available for each register. The software uses one set while the other set - the shadow registers - is used by the PWM. This mechanism also controls the moment at which the updated configuration is transferred to the shadow registers.

Software update of shadow registers is done by configuration of UPD_ENA (Update_Enable), TRANS_ENA (Transfer_Enable) and TRANS_ENA_SEL (Trans_Enable_Select) bits of the MODECTL registers. The MODECTL, TRPCTL, CAPTCTL and CAPTSRC registers are also updated using the UPD_ENA bit. This avoids a large synchronization task.

Registers that are not updated by the UPD_ENA bit can be updated through software or hardware. This is achieved via the TRANS_ENA and TRANS_ENA_SEL bits of the MODECTL register. If TRANS_ENA_SEL is high, shadowing is controlled via hardware, the TRANS_ENA bit is not taken into account and shadowing occurs only if TRANS_EN_IN pin is high (update triggered by MSCSS Timer0 match3 output of TRANS_EN_OUT of previous PWM block). If TRANS_ENA_SEL is low shadowing is controlled via software.

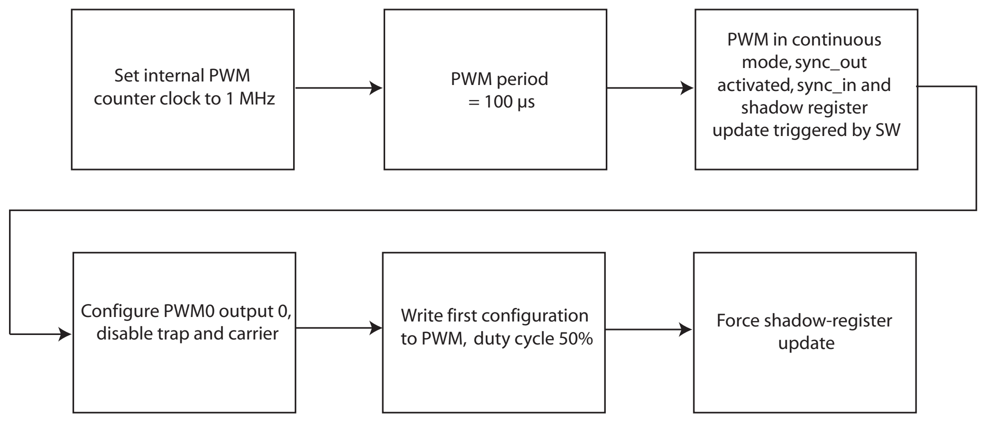

Set internal PWM counter clock to 1 MHz
PWM period = 100 µs
PWM in continuous mode, sync_out activated, sync_in and shadow register update triggered by SW
Configure PWM0 output 0, disable trap and carrier
Write first configuration to PWM, duty cycle 50%
Force shadow-register update

**Fig 110. Update configuration flowchart**

## 4. Functional description

The ability to provide flexible waveforms allows PWM blocks to be used in multiple applications, e.g. dimmer/lamp control and motor control. Pulse-width modulation is the preferred method to regulate power because no additional heat is generated and it is energy efficient when compared to linear regulating voltage control networks.

PWM is used to deliver the waveforms/pulses of desired duty cycles and cycle periods. The very basic application of these pulses can be in controlling the amount of power transferred to a load. As the duty cycle of the pulses can be controlled, the desired amount of power can be transferred for a controlled duration.

Figure 25–111 illustrates the operation of a PWM in continuous mode. Each PWM consists of an internal 16-bit counter (CNT register). This counter is clocked with the prescaled system clock (PRSC register). When the counter reaches the threshold defined by the PRD register it resets and starts counting from the beginning.

The rising and falling edges of the PWM signal are freely configurable. The MTCHACT register defines the position of the rising edge while the MTCHDEACT register defines the falling edge of the waveform. The PWM output changes when the internal PWM counter matches the values defined in the related registers (MTCHACT and MTCHDEACT).

The sync_in input of each PWM timer is used to reset (resynchronize) the PWM block. The sync_out can be asserted immediately after the sync_in or can be delayed via the SYNDEL register. Different PWM blocks can be triggered or synchronized under control of the MODECTL register. In Figure 25–111 the PWM0 is synchronized with PWM1. Each time a sync event is provided to PWM0, PWM1 starts or restarts after the sync delay programmed in the SYNDEL register.

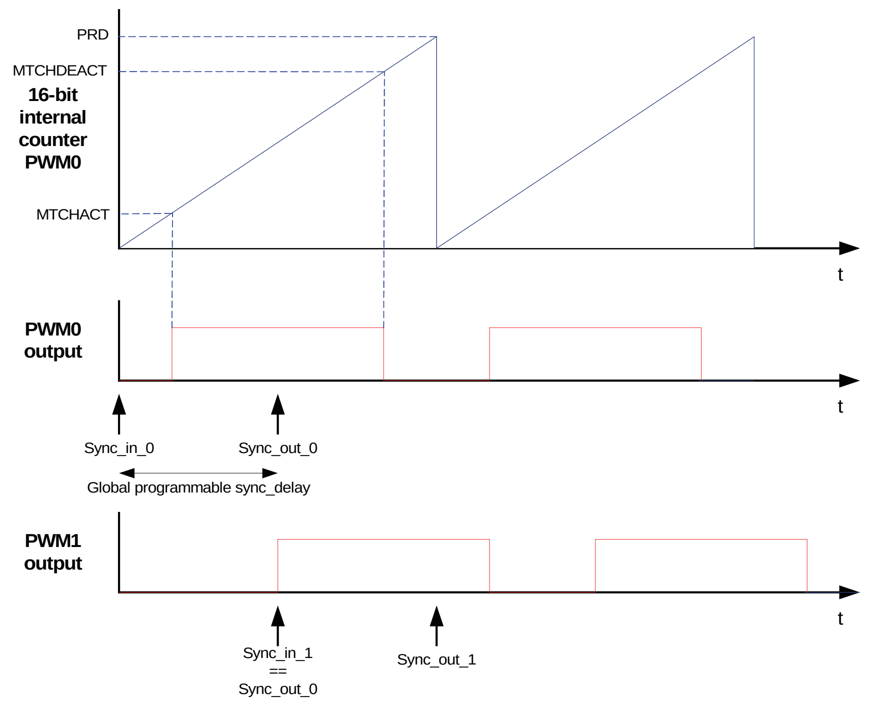


Fig 111.PWM operation

## 4.1 PWM counter synchronization

Several PWMs can be synchronized using TRANS_ENABLE_IN and TRANS_ENABLE_OUT (see Section 25–5.1 ) and the SYNC_IN and SYNC_OUT ports. A PWM module can also provide synchronization signals to other modules. The signal SYNC_OUT is a pulse of one clock cycle's duration generated when the internal PWM counter starts or restarts. The signal TRANS_ENABLE_OUT is a pulse synchronous with SYNC_OUT, but generated if a shadow register update occurs when the PWM counter restarts. By using the SYNDEL register a delay can be inserted between the counter start and the generation of TRANS_ENABLE_OUT and SYNC_OUT.

## 4.2 Delayed register update triggered by timer 0

The update of the PWM configuration registers (CTRL, PRD, PRSC, SYNDEL, CNT, MTCHACT, MTCHDEACT) can also be triggered by hardware. This is done through the trans_enable_in signal of each PWM block. The match3 output of the MSCSS Timer0 (Timer_ADC) can be used to trigger the register update.

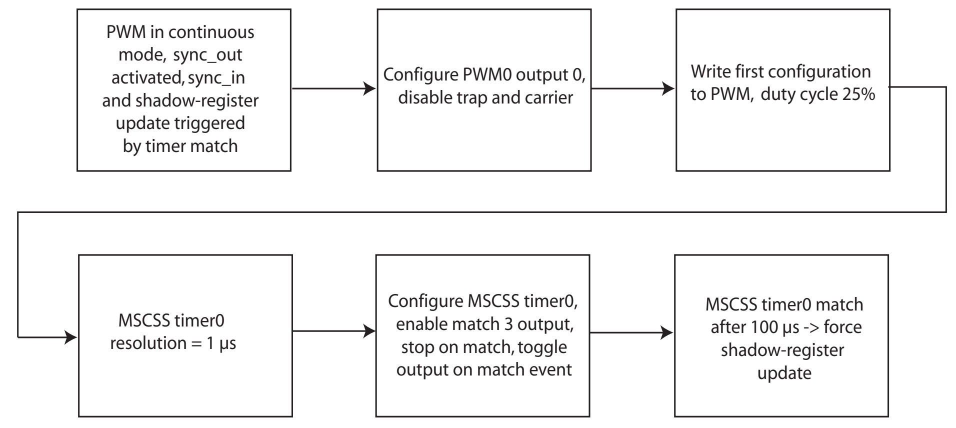

PWM in continuous mode, sync_out activated, sync_in and shadow-register update triggered by timer match
Configure PWM0 output 0, disable trap and carrier
Write first configuration to PWM, duty cycle 25%
MSCSS timer0 resolution = 1 µs
Configure MSCSS timer0, enable match 3 output, stop on match, toggle output on match event
MSCSS timer0 match after 100 µs -> force shadow-register update

**Fig 112. Delayed-update configuration flowchart**

## 4.3 Center-aligned PWM

Center-aligned PWM can be easily achieved via software by using the MTCHACT and MTCHDEACT registers.

## 4.4 Input capturing

Each PWM has four capture channels, with channels 2 and 3 on the same external pin so that in effect there are only three external capture sources per PWM. The capture source for each channel can be selected from the external PWM capture source (capture pin), TRAP signal, the sync_in signal and the trans_enable_in signal. In this mode, the counter's content is latched into the respective capture registers in response to an external event. This event can be programmed to be effective on a negative, a positive or both a negative and a positive transition at the corresponding external input pin. Interrupts can be generated at each transition of the external-capture input pin.

## 4.5 Modulation of PWM and timer carrier

The PWM block has a carrier input pin. This means that active phases of the PWM outputs can be further modulated, e.g. to influence the speed or torque of a motor.

Using the burst method (BURST_ENA), the active phases of the pulse-width modulated outputs are modulated by the output of the MSCSS Timer1 (Timer_PWM). The modulating signal typically has a higher frequency than the modulated output signals. The BURST_ENA bits of the CTRL register are used to enable modulation of the carrier and a particular PWM output. The carrier signal is derived from the match output of the MSCSS Timer 1 (Timer_PWM).

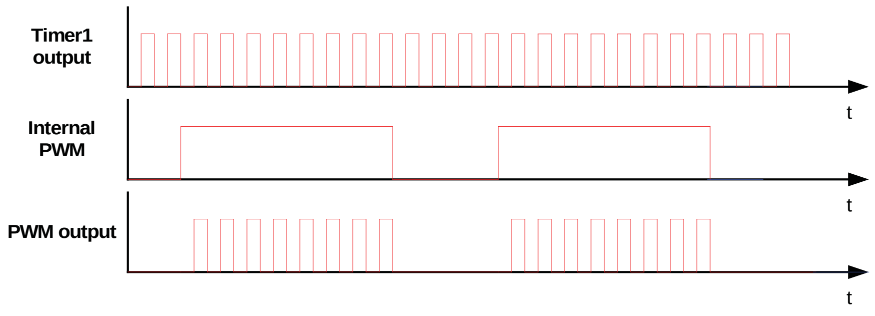


Fig 113. Modulation of PWM and timer carrier

Figure 25–113 illustrates the carrier signal (Timer1, 50% duty cycle) the internal PWM signal and the modulated output signal of the PWM. The flowchart below shows how to configure the PWM carrier input to achieve this result:

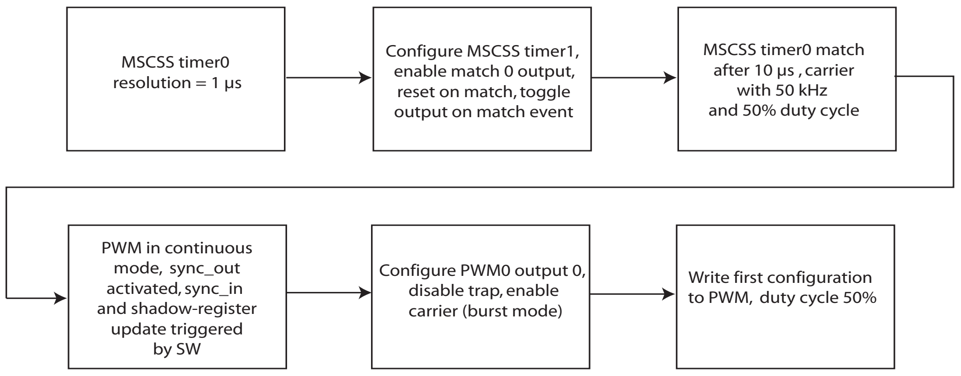

MSCSS timer0 resolution = 1 μs
Configure MSCSS timer1, enable match 0 output, reset on match, toggle output on match event
MSCSS timer0 match after 10 μs , carrier with 50 kHz and 50% duty cycle
PWM in continuous mode, sync_out activated, sync_in and shadow-register update triggered by SW
Configure PWM0 output 0, disable trap, enable carrier (burst mode)
Write first configuration to PWM, duty cycle 50%

Fig 114. Carrrier-input configuration flowchart

## 5. Register overview

Table 357. Register overview: PWM (base address: 0xE00C 5000 (PWM0), 0xE00C 6000 (PWM1), 0xE00C 7000 (PWM2), 0xE00C 8000 (PWM3))

| Name | Access | Address | Description | Reset Value | Reference |
|---|---|---|---|---|---|
| MODECTL | R/W | 0x000 | Main control register | 0x0000 0000 | Table 25–358 |
| TRPCTL | R/W | 0x004 | Controls the behavior of PWM outputs when there is an event on the PWMx TRAP input pin | 0x0001 003F | Table 25–360 |
| CAPTCTL | R/W | 0x008 | Controls the behavior of the Capture_Registers and associated pins | 0x0000 0000 | Table 25–361 |
| CAPTSRC | R/W | 0x00C | Controls the source of the capture events | 0x0000 0000 | Table 25–362 |
| CTRL | R/W | 0x010 | Controls the PWM output behavior. | 0x0000 003F | Table 25–363 |
| PRD | R/W | 0x014 | The cycle period (minus 1) of all PWM output | 0x0000 FFFF | Table 25–364 |

Table 357. Register overview: PWM (base address: 0xE00C 5000 (PWM0), 0xE00C 6000 (PWM1), 0xE00C 7000 (PWM2), 0xE00C 8000 (PWM3)) *…continued*

| Name | Access | Address | Description | Reset Value | Reference |
|---|---|---|---|---|---|
| PRSC | R/W | 0x018 | The prescale register defines the number of system clock cycles to be counted (PR+1) before the PWM counter increments | 0x0000 FFFF | Table 25–365 |
| SYNDEL | R/W | 0x01C | Holds the delay between input and output trigger signals of the synchronization port | 0x0000 0000 | Table 25–366 |
| CNT | R | 0x020 | The PWM counter increments every Prescale+1 system clock cycles. When no pre scaling is required the PWM_Prescale should be kept at its reset value. Single system clock cycles are then counted | 0x0000 0000 | Table 25–367 |
| MTCHACT(0) | R/W | 0x100 | Holds the first (activation) match value related to PWM 0 output | 0x0000 0000 | Table 25–368 |
| MTCHACT(1) | R/W | 0x104 | Holds the first (activation) match value related to PWM 1 output | 0x0000 0000 | Table 25–368 |
| MTCHACT(2) | R/W | 0x108 | Holds the first (activation) match value related to PWM 2 output | 0x0000 0000 | Table 25–368 |
| MTCHACT(3) | R/W | 0x10C | Holds the first (activation) match value related to PWM 3 output | 0x0000 0000 | Table 25–368 |
| MTCHACT(4) | R/W | 0x110 | Holds the first (activation) match value related to PWM 4 output | 0x0000 0000 | Table 25–368 |
| MTCHACT(5) | R/W | 0x114 | Holds the first (activation) match value related to PWM 5 output | 0x0000 0000 | Table 25–368 |
| MTCHDEACT(0) | R/W | 0x200 | Holds the second (de-activation) match value related to PWM 0 output | 0x0000 0000 | Table 25–369 |
| MTCHDEACT(1) | R/W | 0x204 | Holds the second (de-activation) match value related to PWM 1 output | 0x0000 0000 | Table 25–369 |
| MTCHDEACT(2) | R/W | 0x208 | Holds the second (de-activation) match value related to PWM 2 output | 0x0000 0000 | Table 25–369 |
| MTCHDEACT(3) | R/W | 0x20C | Holds the second (de-activation) match value related to PWM 3 output | 0x0000 0000 | Table 25–369 |
| MTCHDEACT(4) | R/W | 0x210 | Holds the second (de-activation) match value related to PWM 4 output | 0x0000 0000 | Table 25–369 |
| MTCHDEACT(5) | R/W | 0x214 | Holds the second (de-activation) match value related to PWM 5 output | 0x0000 0000 | Table 25–369 |
| CAPT(0) | R | 0x300 | Holds the captured value on the selected event of capture channel 0 | 0x0000 0000 | Table 25–370 |
| CAPT(1) | R | 0x304 | Holds the captured value on the selected event of capture channel 1 | 0x0000 0000 | Table 25–370 |
| CAPT(2) | R | 0x308 | Holds the captured value on the selected event of capture channel 2 | 0x0000 0000 | Table 25–370 |
| CAPT(3) | R | 0x30C | Holds the captured value on the selected event of capture channel 3 | 0x0000 0000 | Table 25–370 |
| MODECTLS | R | 0x800 | Mirror the synchronized MODECTL register from PWM domain. Stable only if UPD_ENA is 0 | 0x0000 0000 | Table 25–371 |

Table 357. Register overview: PWM (base address: 0xE00C 5000 (PWM0), 0xE00C 6000 (PWM1), 0xE00C 7000 (PWM2), 0xE00C 8000 (PWM3)) *…continued*

| Name | Access | Address | Description | Reset Value | Reference |
|---|---|---|---|---|---|
| TRPCTLS | R | 0x804 | Mirror the synchronized TRPCTL register from PWM domain. Stable only if UPD_ENA is 0 | 0x0000 0000 | Table 25–372 |
| CAPTCTLS | R | 0x808 | Mirror the synchronized CAPTCTL register from PWM domain. Stable only if UPD_ENA is 0 | 0x0000 0000 | Table 25–373 |
| CAPTSRCS | R | 0x80C | Mirror the synchronized CAPTSRC register from PWM domain. Stable only if UPD_ENA is 0 | 0x0000 0000 | Table 25–374 |
| CTRLS | R | 0x810 | Mirror the shadowed CTRL register from PWM domain. Stable only if TRANS_ENA is low | 0x0000 003F | Table 25–375 |
| PRDS | R | 0x814 | Mirror the shadowed PRD register from PWM domain. Stable only if TRANS_ENA is 0 | 0x0000 FFFF | Table 25–376 |
| PRSCS | R | 0x818 | Mirror the shadowed PRSC register from PWM domain. Stable only if TRANS_ENA is 0 | 0x0000 FFFF | Table 25–377 |
| SYNDELS | R | 0x81C | Mirror the shadowed SYNDEL register from PWM domain. Stable only if TRANS_ENA is 0 | 0x0000 0000 | Table 25–378 |
| MTCHACTS(0) | R | 0x900 | Mirror the activation match shadowed value related to PWM 0 output. | 0x0000 0000 | Table 25–379 |
| MTCHACTS(1) | R | 0x904 | Mirror the activation match shadowed value related to PWM 1 output | 0x0000 0000 | Table 25–379 |
| MTCHACTS(2) | R | 0x908 | Mirror the activation match shadowed value related to PWM 2 output | 0x0000 0000 | Table 25–379 |
| MTCHACTS(3) | R | 0x90C | Mirror the activation match shadowed value related to PWM 3 output | 0x0000 0000 | Table 25–379 |
| MTCHACTS(4) | R | 0x910 | Mirror the activation match shadowed value related to PWM 4 output | 0x0000 0000 | Table 25–379 |
| MTCHACTS(5) | R | 0x914 | Mirror the activation match shadowed value related to PWM 5 output | 0x0000 0000 | Table 25–379 |
| MTCHDEACTS(0) | R | 0xA00 | Mirror the de-activation match shadowed value related to PWM 0 output | 0x0000 0000 | Table 25–380 |
| MTCHDEACTS(1) | R | 0xA04 | Mirror the de-activation match shadowed value related to PWM 1 output | 0x0000 0000 | Table 25–380 |
| MTCHDEACTS(2) | R | 0xA08 | Mirror the de-activation match shadowed value related to PWM 2 output | 0x0000 0000 | Table 25–380 |
| MTCHDEACTS(3) | R | 0xA0C | Mirror the de-activation match shadowed value related to PWM 3 output | 0x0000 0000 | Table 25–380 |
| MTCHDEACTS(4) | R | 0xA10 | Mirror the de-activation match shadowed value related to PWM 4 output | 0x0000 0000 | Table 25–380 |
| MTCHDEACTS(5) | R | 0xA14 | Mirror the de-activation match shadowed value related to PWM 5 output | 0x0000 0000 | Table 25–380 |
| INT_CLR_ENABLE | W | 0xF90 | PWM interrupt clear-enable register | - | Table 10–91 |
| INT_SET_ENABLE | W | 0xF94 | PWM interrupt set-enable register | - | Table 10–92 |

**Table 357. Register overview: PWM (base address: 0xE00C 5000 (PWM0), 0xE00C 6000 (PWM1), 0xE00C 7000 (PWM2), 0xE00C 8000 (PWM3))** *…continued*

| Name | Access | Address | Description | Reset Value | Reference |
|---|---|---|---|---|---|
| INT_STATUS | R | 0xF98 | PWM interrupt status register | 0x0000 0000 | Table 10–93 |
| INT_ENABLE | R | 0xF9C | PWM interrupt enable register | 0x0000 0000 | Table 10–94 |
| INT_CLR_STATUS | W | 0xFA0 | PWM interrupt clear-status register | - | Table 10–95 |
| INT_SET_STATUS | W | 0xFA4 | PWM interrupt set-status register | - | Table 10–96 |
| INT_MTCH_CLR_ENABLE | W | 0xFA8 | Match interrupt clear-enable register | - | Table 10–91 |
| INT_MTCH_SET_ENABLE | W | 0xFAC | Match interrupt set enable register | - | Table 10–92 |
| INT_MTCH_STATUS | R | 0xFB0 | Match interrupt status register | 0x0000 0000 | Table 10–93 |
| INT_MTCH_ENABLE | R | 0xFB4 | Match interrupt enable register | 0x0000 0000 | Table 10–94 |
| INT_MTCH_CLR_STATUS | W | 0xFB8 | Match interrupt clear-status register | - | Table 10–95 |
| INT_MTCH_SET_STATUS | W | 0xFBC | Match interrupt set-status register | - | Table 10–96 |
| INT_CAPT_CLR_ENABLE | W | 0xFC0 | Capture interrupt clear-enable register | - | Table 10–91 |
| INT_CAPT_SET_ENABLE | W | 0xFC4 | Capture interrupt set-enable register | - | Table 10–92 |
| INT_CAPT_STATUS | R | 0xFC8 | Capture interrupt status register | 0x0000 0000 | Table 10–93 |
| INT_CAPT_ENABLE | R | 0xFCC | Capture interrupt enable register | 0x0000 0000 | Table 10–94 |
| INT_CAPT_CLR_STATUS | W | 0xFD0 | Capture interrupt clear-status register | - | Table 10–95 |
| INT_CAPT_SET_STATUS | W | 0xFD4 | Capture interrupt set-status register | - | Table 10–96 |

## 5.1 PWM shadow registers

Shadow registers are configured in the system domain but duplicated in the PWM domain to allow reconfiguration of the PWM ‘on the fly’. A mechanism is provided to modify configuration of the PWM and control the moment at which the updated configuration is transferred to the shadow registers in the PWM domain.

The actual moment the PWM shadow registers are updated can be configured to take place under software control or hardware control (trans_enable_in signal).

## 5.2 PWM mode control register

The MODECTL register is used to enable or reset the PWM counter and the internal prescale counter. It also contains the bit fields to control the update of the shadow registers and bit fields to control synchronization.

Table 25–358 shows the bit assignment of the MODECTL register.

The counting process starts once the CNT_ENA bit is set. The counting process can be reset by setting the CNT_RESET bit. The PWM counter and prescale counter remain in the reset state as long as the CNT_RESET bit is active.

When the update enable (UPD_ENA) bit register is set, an update of the shadow registers (see Section 25–5.1) is initiated. Software should not modify the contents of the registers until the UPDATE_ENABLE bit is cleared by the device, indicating that the update of the shadow registers is done by the hardware.

The actual moment the PWM shadow registers are updated is controlled by two other bit fields in the MODECTL register; TRANS_ENA (transfer enable) and TRANS_ENA_SEL. If TRANS_ENA_SEL is set updating of the shadow registers is done by the hardware on an active HIGH-level on the trans_enable_in pin of the PWM. If TRANS_ENA_SEL is cleared, the trans_enable_in pin is ignored and update of the shadow registers takes place when the TRANS_ENA bit is set AND the PWM counter starts a new cycle. This happens when the counter overflows, or on an active HIGH signal on the sync_in input pin (if enabled via the SYNC_SEL bit of the MODECTL register).

When shadow registers are stable in the PWM domain the TRANS_ENA bit field is automatically reset, and an interrupt might then be generated.

**Table 358. MODECTL register bit description (0xE00C 5000 (PWM0), 0xE00C 6000 (PWM1), 0xE00C 7000 (PWM2), 0xE00C 8000 (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 | reserved | R | 0 | Reserved |
| 30-8 | reserved | R | - | Reserved |
| 7 | UPD_ENA | R/W | 1 | Enables synchronization to the PWM domain. This bit is automatically reset when synchronization is finished |
| | | | 0* | |
| 6 | TRANS_ENA | R/W | 1 | Enables transfer to the compare registers. Effective transfer is done when the PWM counter overflows if TRANS_ENA is logic 1. See Table 25–359. This bit is automatically reset when shadowing is finished[1] |
| | | | 0* | |
| 5 | TRANS_ENA_SEL | R/W | | Selection of the enable signal which allows the transfer of the new set of values; see Table 25–359 |
| | | | 1 | active HIGH-level on trans_enable_in pin |
| | | | 0* | TRANS_ENA bit of the MODECTL register |
| 4 | SYNC_SEL | R/W | | Selection of the synchronization source; see Table 25–359 |
| | | | 1 | Sync_in pin |
| | | | 0* | Internal |
| 3 | SYNC_OUT_ENA | R/W | | Sync_out enable |
| | | | 1 | Sync_out is active |
| | | | 0* | Sync_out is stuck LOW |
| 2 | RUN_ONCE | R/W | 1 | PWM counter stops at the end of current cycle |
| | | | 0* | |
| 1 | CNT_RESET | R/W | 1 | Synchronously reset PWM counter and prescale counter. Counters remain reset when this bit is active |
| | | | 0* | |
| 0 | CNT_ENA | R/W | 1 | Enables the PWM counter and prescale counter |
| | | | 0* | |

[1] Set this bit when an interrupt should be enabled on Transfer_done.

**Table 359. Bits involved in shadow register update**

| SYNC_SEL | TRANS_ENA_SEL | Update of registers |
|---|---|---|
| 0 | x | TRANS_ENA bit |
| 1 | 0 | TRANS_ENA bit (master mode) |
| 1 | 1 | trans_enable_in signal (slave mode) |

## 5.3 PWM trap control register

The APB PWM has an external asynchronous input which is used to switch the selected PWM outputs to the not-active level. The polarity of the active level is defined by bit ACT_LVL, see Table 25–363. The possibility of enabling this feature (TRAP_ENA) can be defined for each output. The trap-signal polarity that triggers the emergency is also contained in the trap control register.

Table 25–360 shows the bit assignment of the TRPCTL register.

**Table 360. TRPCTL register bit description (0xE00C 5004 (PWM0), 0xE00C 6004 (PWM1), 0xE00C 7004 (PWM2), 0xE00C 8004 (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 17 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 16 | TRAP_POL | R/W | 1* | The asynchronous trap input is active on HIGH level (rising edge is detected to switch the PWM outputs to the not-active level as defined by ACT_LVL) |
| | | | 0 | the asynchronous trap input is active on LOW level (falling edge is detected to switch the PWM outputs to the not-active level as defined by ACT_LVL) |
| 15 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 | TRAP_ENA[5] | R/W | 1* | Trap function is enabled for the corresponding PWM 5 output |
| : | : | : | : | : |
| 0 | TRAP_ENA[0] | R/W | 1* | Trap function is enabled for the corresponding PWM 0 output |

## 5.4 PWM capture control register

The PWM has four capture registers, and the capture behavior of the associated CAPT registers and pins is controlled by the CAPCTL register. The CNT register value is captured synchronously with the system clock.

Table 25–361 shows the bit assignment of the CAPCTL register.

**Remark:** Capture channels two and three are associated with the same pin.

**Table 361. CAPCTL register bit description (0xE00C 5008 (PWM0), 0xE00C 6008 (PWM1), 0xE00C 7008 (PWM2), 0xE00C 8008 (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 and 6 | CAPT_EDGE3[1:0] | R/W | | Select the edge of the capture channel 3 to trigger capture of the PWM |
| | | | 00* | No triggering |
| | | | 01 | Triggering on rising edge |
| | | | 10 | Triggering on falling edge |
| | | | 11 | Triggering on both edges |
| : | : | : | : | : |
| 1 and 0 | CAPT_EDGE0[1:0] | R/W | | Select the edge of the capture channel 0 to trigger capture of the PWM |
| | | | 00* | No triggering |
| | | | 01 | Triggering on rising edge |
| | | | 10 | Triggering on falling edge |
| | | | 11 | Triggering on both edges |

## 5.5 PWM capture source register

Each PWM has four capture channels, channels 2 and 3 being on the same external pin so that in effect there are only three external capture sources per PWM. The capture source for each channel can be selected between the external PWMx CAPTy and PWMx TRAP pins, and the internal sync_in and the trans_enable_in signals.

Table 25–362 shows the bit assignment of the CAPSRC register.

**Table 362. CAPSRC register bit description (0xE00C 500C (PWM0), 0xE00C 600C (PWM1), 0xE00C 700C (PWM2), 0xE00C 800C (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 and 6 | CAPT_SRC3[1:0] | R/W | | Select the source of capture channel 3 to trigger capture of the PWM |
| | | | 00* | PWMx CAPT3 signal, x is index of PWM[1] |
| | | | 01 | sync_in signal |
| | | | 10 | PWMx TRAP signal, x is index of PWM |
| | | | 11 | Trans_enable_in signal |
| : | : | : | : | : |
| 1 and 0 | CAPT_SRC0[1:0] | R/W | | Select the source of capture channel 0 to trigger capture of the PWM |
| | | | 00* | PWMx CAPT0 signal, x is index of PWM[1] |
| | | | 01 | Sync_in signal |
| | | | 10 | PWMx TRAP signal, x is index of PWM |
| | | | 11 | Trans_enable_in signal |

[1] There are only three external capture inputs per PWM. PWMx CAPT2 is also connected to CAPT3.

## 5.6 PWM control register

The CTRL register selects the active level for each output. It also allows mixing of the external carrier signal with the internal PWM generated signals.

Table 25–363 shows the bit assignment of the CTRL register.

**Table 363. CTRL register bit description (0xE00C 5010 (PWM0), 0xE00C 6010 (PWM1), 0xE00C 7010 (PWM2), 0xE00C 8010 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 22 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 21 | BURST_ENA[5] | R/W | 1 | PWM 5 is mixed with the external carrier input |
| | | | 0* | |
| : | : | : | : | : |
| 16 | BURST_ENA[0] | R/W | 1 | PWM 0 is mixed with the external carrier input |
| | | | 0* | |
| 15 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 | ACT_LVL[5] | R/W | 1* | PWM 5 output is at a HIGH level for the active state. [1] |
| | | | 0 | PWM 5 output is at a LOW level for the active state. [1] |
| : | : | : | : | : |
| 0 | ACT_LVL[0] | R/W | 1* | PWM 0 output is at a HIGH level for the active state.[1] |
| | | | 0 | PWM 0 output is at a LOW level for the active state.[1] |

[1] Changing the ACT_LVL bits may cause the outputs to change directly if the PWM outputs are enabled through the SCU.

## 5.7 PWM period register

The PRD register contains the cycle period value minus 1. PWM output period is:

$(PRD + 1) \times (PRSC + 1)$ system clock cycles

Given the desired PWM period, values for PRD and PRSC can be derived from:

$PRD \times PRSC = t_{PWM} / t_{clk(sys)} - 1$

Table 25–364 shows the bit assignment of the PRD register.

**Table 364. PRD register bit description (0xE00C 5014 (PWM0), 0xE00C 6014 (PWM1), 0xE00C 7014 (PWM2), 0xE00C 8014 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | PRD | R/W | | Period cycle minus 1 |
| | | | 0xFFFF* | |

## 5.8 PWM prescale register

The PWM has a prescale register. The PWM counter increments after ‘PRSC + 1’ PWM clock cycles are counted.

Table 25–365 shows the bit assignment of the PRSC register.

**Table 365. PRSC register bit description (0xE00C 5018 (PWM0), 0xE00C 6018 (PWM1), 0xE00C 7018 (PWM2), 0xE00C 8018 (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | PRSC | R/W | | Prescaler value |
| | | | 0xFFFF* | |

## 5.9 PWM synchronization delay register

The SYNDEL register allows delay of the trigger sync_out pin. Sync_out is generated when the internal PWM counter matches the SYNDEL register and the prescale counter overflows.

Table 25–366 shows the bit assignment of the SYNDEL register.

**Table 366. SYNDEL register bit description (0xE00C 501C (PWM0), 0xE00C 601C (PWM1), 0xE00C 701C (PWM2), 0xE00C 801C (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | DLY | R/W | | Value in system clock cycles of the delay between the sync_in and sync_out pins |
| | | | 0x0000* | |

## 5.10 PWM count register

The CNT register contains the current PWM value. When the PRSC register is zero the PWM counter increments every system clock cycle: when the PRSC register value is unequal to zero an internal prescale counter first counts the number of system clock cycles as defined in this register plus one, then increments the PWM value.

Table 25–367 shows the bit assignment of the CNT register.

**Table 367. CNT register bit description (0xE00C 5020 (PWM0), 0xE00C 6020 (PWM1), 0xE00C 7020 (PWM2), 0xE00C 8020 (PWM3))**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | CNT | R | | PWM counter value |
| | | | 0x0000* | |

## 5.11 PWM match active registers

There are six MTCHACT registers per PWM; one for each PWM output. Each MTCHACT register can be programmed to contain the first value which is compared with the PWM counter to generate the corresponding PWM output and interrupt. By making use of the shadow register concept, updating the MTCHACT register while the PWM counter is running is possible without affecting the current PWM outputs, see Section 25–5.1. Reading this register returns the last written value, but not the value actually used by the PWM counter comparator.

Table 25–368 shows the bit assignment of the MTCHACT(0) to MTCHACT(5) registers.

**Table 368. MTCHACT(0 - 5) register bit description (0xE00C 5100 - 114 (PWM0), 0xE00C 6100 - 114 (PWM1), 0xE00C 7100 - 114 (PWM2), 0xE00C 8100 - 114 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | MTCHACT | R/W | | The first (activation) match value which is compared with the PWM counter to generate the PWM(m) output and an interrupt |
| | | | 0x0000* | |

## 5.12 PWM match deactive registers

There are six MTCHDEACT registers per PWM, one for each PWM output. Each MTCHACT register can be programmed to contain the second value which is compared to the PWM counter to generate the corresponding PWM output and interrupt. The use of this register is similar to the MTCHACT register, see Section 25–5.11.

Table 25–369 shows the bit assignment of the MTCHDEACT(0) to MTCHDEACT(5) registers.

**Table 369. MTDECHACT(0 - 5) register bit description (0xE00C 5200 - 214 (PWM0), 0xE00C 6200 - 214 (PWM1), 0xE00C 7200 - 214 (PWM2), 0xE00C 8200 - 214 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | MTDECHACT | R/W | | The second (deactivation) match value which is compared with the PWM counter to generate the PWM(m) output and an interrupt |
| | | | 0x0000* | |

## 5.13 PWM capture registers

There are four CAPT registers per PWM, one for each external PWM capture channel. See Section 25–5.5 for selecting the source for each capture register and the limitations of capture channel 3. The CAPT register contains the captured value triggered by the corresponding channel.

Table 25–370 shows the bit assignment of the CAPT0 to CAPT3 registers.

Table 370. CAPT0 - 3 register bit description (0xE00C 5300 - 30C (PWM0), 0xE00C 6300 - 30C (PWM1), 0xE00C 7300 - 30C (PWM2), 0xE00C 8300 - 30C (PWM3))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | CAPT | R | | The PWM counter value captured at the event programmed on the capture channel pin |
| | | | 0x0000* | |

## 5.14 PWM mode control shadow register

The MODECTLS register is the shadow register of the MODECTL register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–371 shows the bit assignment of the MODECTLS register.

Table 371. MODECTLS register bit description (0xE00C 5800 (PWM0), 0xE00C 6800 (PWM1), 0xE00C 7800 (PWM2), 0xE00C 8800 (PWM3))
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 30 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0. |
| 5 | TRANS_ENA_SEL_SYNC | R | | Mirrors the synchronized TRANS_ENA_SEL bit field |
| | | | 0* | |
| 4 | SYNC_SEL_SYNC | R | | Mirrors the synchronized SYNC_SEL bit field |
| | | | 0* | |
| 3 | reserved | R | - | Reserved; do not modify. Read as logic 0. |
| 2 | RUN_ONCE_SYNC | R | | Mirrors the synchronized RUN_ONCE bit field |
| | | | 0* | |
| 1 | CNT_RESET_SYNC | R | | Mirrors the synchronized CNT_RESET bit field |
| | | | 0* | |
| 0 | CNT_ENA_SYNC | R | | Mirrors the synchronized CNT_ENA bit field |
| | | | 0* | |

## 5.15 PWM trap control shadow register

The TRPCTLS register is the shadow register of the TRPCTL register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–372 shows the bit assignment of the TRPCTLS register.

**Table 372. TRPCTLS register bit description (0xE00C 5804 (PWM0), 0xE00C 6804 (PWM1), 0xE00C 7804 (PWM2), 0xE00C 8804 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 30 to 17 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 16 | TRAP_POL_SYNC | R | | Mirrors the synchronized TRAP_POL bit field |
| | | | 0* | |
| 15 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 to 0 | TRAP_ENA_SYNC | R | | Mirrors the synchronized TRAP_ENA bit field |
| | | | 0x00* | |

## 5.16 PWM capture control shadow register

The CAPTCTLS register is the shadow register of the CAPTCTL register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–373 shows the bit assignment of the CAPTCTLS register.

Table 25–361 shows the explanation of the values for the CAPT_CTL_SYNC bit fields.

**Table 373. CAPTCTLS register bit description (0xE00C 5808 (PWM0), 0xE00C 6808 (PWM1), 0xE00C 7808 (PWM2), 0xE00C 8808 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 30 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 and 6 | CAPT_EDGE_SYNC3[1:0] | R | | Mirrors the synchronized CAPT_EDGE3 bit field |
| | | | 0x0* | |
| : | : | : | : | : |
| 1 and 0 | CAPT_EDGE_SYNC0[1:0] | R | | Mirrors the synchronized CAPT_EDGE0 bit field |
| | | | 0x0* | |

## 5.17 PWM capture source shadow register

The CAPTSRCS register is the shadow register of the CAPTSRC register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information about the principle of shadow registers.

Table 25–374 shows the bit assignment of the CAPTSRCS register.

Table 25–362 shows the explanation of the values for the CAPT_SRC_SYNC bit fields.

**Table 374. CAPTSRCS register bit description (0xE00C 580C (PWM0), 0xE00C 680C (PWM1), 0xE00C 780C (PWM2), 0xE00C 880C (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 30 to 6 | reserved | R | - | reserved; do not modify. Read as logic 0 |
| 7 and 6 | CAPT_SRC_SYNC3[1:0] | R | | Mirrors the synchronized CAPT_SRC3 bit field |
| | | | 0x0* | |
| : | : | : | : | : |
| 1 and 0 | CAPT_SRC_SYNC0[1:0] | R | | Mirrors the synchronized CAPT_SRC0 bit field |
| | | | 0x0* | |

## 5.18 PWM control shadow register

The CTRLS register is the shadow register of the CAPTSRC register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–375 shows the bit assignment of the CTRLS register.

**Table 375. CTRLS register bit description (0xE00C 5810 (PWM0), 0xE00C 6810 (PWM1), 0xE00C 7810 (PWM2), 0xE00C 8810 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 22 | reserved | R | - | reserved; do not modify. Read as logic 0 |
| 21 to 16 | BURST_ENA_SHAD[5:0] | R | | Mirrors the shadowed BURST_ENA bit field |
| | | | 0x00* | |
| 15 to 6 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 5 to 0 | ACT_LVL_SHAD[5:0] | R | | Mirrors the shadowed ACT_LVL bit field |
| | | | 0x3F* | |

## 5.19 PWM period shadow register

The PRDS register is the shadow register of the PRD register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–376 shows the bit assignment of the PRDS register.

**Table 376. PRDS register bit description (0xE00C 5814 (PWM0), 0xE00C 6814 (PWM1), 0xE00C 7814 (PWM2), 0xE00C 8814 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | PRD_SHAD | R | | Mirrors the shadowed PRD bit field |
| | | | 0xFFFF* | |

## 5.20 PWM prescale shadow register

The PRSCS register is the shadow register of the PRSC register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–377 shows the bit assignment of the PRSCS register.

**Table 377. PRSCS register bit description (0xE00C 5818 (PWM0), 0xE00C 6818 (PWM1), 0xE00C 7818 (PWM2), 0xE00C 8818 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | PRSC_SHAD | R | | Mirrors the shadowed PRSC bit field |
| | | | 0xFFFF* | |

## 5.21 PWM synchronization delay shadow register

The SYNDELS register is the shadow register of the SYNDEL register. It mirrors the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–378 shows the bit assignment of the SYNDELS register.

**Table 378. SYNDELS register bit description (0xE00C 581C (PWM0), 0xE00C 681C (PWM1), 0xE00C 781C (PWM2), 0xE00C 881C (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | DLY_SHAD | R | | Mirrors the shadowed DLY bit field |
| | | | 0x0000* | |

## 5.22 PWM match active shadow registers

The MTCHACTS registers are the shadow registers of the MTCHACT registers. They mirror the values used in the PWM domain. See Section 25–5.1 for more information on the principle of shadow registers.

Table 25–379 shows the bit assignment of the MTCHACTS(0) to MTCHACTS(5) registers.

**Table 379. MTCHACTS(0 - 5) registers bit description (0xE00C 5900 - 914 (PWM0), 0xE00C 6900 - 914 (PWM1), 0xE00C 7900 - 914 (PWM2), 0xE00C 8900 - 914 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | MTCHACT_SHAD | R | | Mirrors the first (activation) value which is compared with the PWM counter to generate the PWM(m) output and an interrupt |
| | | | 0x0000* | |

## 5.23 PWM match deactive shadow registers

The MTCHDEACTS registers are the shadow registers of the MTCHDEACT registers. They mirror the values used in the PWM domain. See Section 25–5.1 for more information about the principle of the shadow registers.

Table 25–380 shows the bit assignment of each MTCHDEACTS(0) to MTCHDEACTS(5) registers.

**Table 380. MTCHDEACTS(0 - 5) register bit description (0xE00C 5A00 - A14 (PWM0), 0xE00C 6A00 - A14 (PWM1), 0xE00C 7A00 - A14 (PWM2), 0xE00C 8A00 - A14 (PWM3))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 to 0 | MTDECHACT_SHAD | R | | Mirrors the second (deactivation) match value which is compared with the PWM counter to generate the PWM(m) output and an interrupt |
| | | | 0x0000* | |

## 5.24 PWM interrupt bit description

Each PWM has two separate active-HIGH interrupt request pins: intreq_capt_match and intreq_pwm. These interrupt requests are routed to the Vectored Interrupt Controller VIC, see (Section 9–2). The interrupt process has a maximum of 19 possible sources:

- 12 match events for intreq_capt_match
- 3 capture events for intreq_capt_match
- 1 trap event for intreq_pwm
- 1 PWM counter overflow event for intreq_pwm
- 1 transfer event for intreq_pwm
- 1 update event for intreq_pwm

Table 25–381 gives the interrupts for the PWM. The first column gives the bit number in the interrupt registers. For a general explanation of the interrupt concept and a description of the registers see Section 10–5.

**Table 381. PWM interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 4 | unused | Unused |
| 3 | EMGY | Trap emergency event |
| 2 | UD | Update done |
| 1 | TD | Transfer done |
| 0 | CO | PWM counter overflow |

Table 382. PWM Match interrupt sources

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 12 | unused | Unused |
| 11 | MTCHDEACT5 | PWM counter value matching MTCHDEACT5 |
| : | : | : |
| 6 | MTCHDEACT0 | PWM counter value matching MTCHDEACT0 |
| 5 | MTCHACT5 | PWM counter value matching MTCHACT5 |
| : | : | : |
| 0 | MTCHACT0 | PWM counter value matching MTCHACT0 |

Table 383. PWM Capture interrupt sources

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 4 | unused | Unused |
| 3 | CAPT3 | Capture channel 3 |
| 2 | CAPT2 | Capture channel 2 |
| 1 | CAPT1 | Capture channel 1 |
| 0 | CAPT0 | Capture channel 0 |

# UM10316

# Chapter 26: LPC29xx Analog-to-Digital Converter (ADC)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts. ADC0 is available in LPC2926/27/29, LPC2930, and LPC2939 only.

## 2. ADC functional description

The ADC block contains two conversion channels for each input. Therefore, each ADC contains a total of 16 channels connected to 8 input pins. Each input is connected to two channels: input 0 to channels 0 and 8, input 1 to channels 1 and 9, and so forth - see Figure 26–115. Each of these channels can be selected for the analog-to-digital conversion. All selected channels are converted sequentially at each conversion scan.

The conversion can have a resolution of between two and 10 bits, configurable per channel, and it can be either a single or a continuous scan. The conversion scan is started either immediately by software or waits for an external trigger (a timer event or transition on an external input).

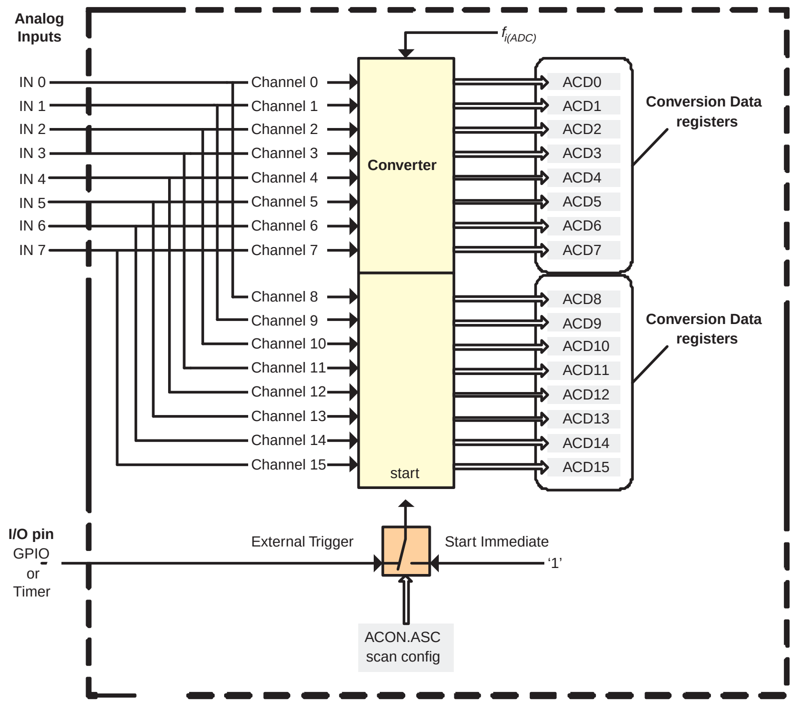


**Fig 115. Schematic representation of the analog to digital converter**

The main control of the ADC is via the ADC control register (ACON). This register allows enabling or disabling the ADC, defining the scan mode – single-shot or continuous – and the channel trigger mode, and it also contains the notification of whether a conversion is running or finished. The channel configuration register (ACC) is used to define the resolution of the individual channels and to enable them. Once the conversion scan is finished the resulting conversion data can be read from the ACD registers.

The sampling rate of the ADC depends on the programmed resolution of the channels. The relation between the two is as follows:

$$f_S = \frac{f_{i(ADC)}}{resolution + 1}$$

## 2.1 Clock distribution

The ADC clock is limited to 4.5 MHz maximum frequency and should always be lower than or equal to the system clock frequency. The CGU provides a programmable fractional system-clock divider dedicated to the ADC clock to fulfill this constraint or to select the desired lower sampling frequency. The conversion rate is determined by the ADC clock frequency divided by the number of resolution bits plus one. Accessing ADC registers requires an enabled ADC clock which is controllable via the CGU.

## 2.2 Compare conversion results with predefined threshold

The ADC provides a feature that reduces the interrupt load of the system, in that an interrupt is only generated when a certain voltage level is greater than or less than the predefined threshold. Comparison of conversion results and the threshold is performed in hardware and an interrupt is requested when the compare condition is true, otherwise the next conversion is started without notification.

## 2.3 Trigger ADC conversion with MSCSS timer 0

Each ADC provides four different options to start a conversion. Each start input is sensitive on either rising or falling edges of the applied trigger (start) signal. The four start inputs are:

- External input
- Timer0 match output
- PWM sync_out signal
- Previous ADC

## 2.4 Interrupt handling

The ADC can be configured to generate an interrupt after a conversion scan. The interrupt control in this case is via the registers Interrupt Enable (AIE), Interrupt Status (AIS) and Interrupt Clear (AIC).

## 3. Pin description

**Table 384. ADC pins**

| Symbol | Pin name | Direction | Description |
|---|---|---|---|
| ADC0 IN[7:0] | IN0[7:0] | IN | analog input for 5.0 V ADC0, channel 7 to channel 0. |
| ADC1/2 IN[7:0] | IN1/2[7:0] | IN | analog input for 3.3 V ADC1/2, channel 7 to channel 0. |
| ADC2_EXT_START | CAP1[2] | IN | ADC external start-trigger input. |
| VREFN | VREFN | IN | ADC LOW reference level. |
| VREFP | VREFP | IN | ADC HIGH reference level. |
| $V_{DDA(ADC5V0)}$ | $V_{DDA(ADC5V0)}$ | IN | 5 V high-power supply and HIGH reference for ADC0. Connect to clean 5 V as HIGH reference. May also be connected to 3.3 V if 3.3 V measurement range for ADC0 is needed.[1][2] |
| $V_{DDA(ADC3V3)}$ | $V_{DDA(ADC3V3)}$ | IN | ADC1 and ADC2 3.3 V supply (also used for ADC0).[2] |

[1] The analog inputs of ADC0 are internally multiplied by a factor of 3.3 / 5. If $V_{DDA(ADC5V0)}$ is connected to 3.3 V, the maximum digital result is 1024 × 3.3 / 5.

[2] $V_{DDA(ADC5V0)}$ and $V_{DDA(ADC3V3)}$ must be set as follows: $V_{DDA(ADC5V0)} = V_{DDA(ADC3V3)} \times 1.5$.

**Remark:** Note that the ADC1 and ADC2 accept an input voltage up to of 3.6 V on the ADC1/2 IN pins. If the ADC is not used, the pins are 5 V tolerant. The ADC0 pins are 5 V tolerant.

Voltage variations on VREFP (i.e. those that deviate from voltage variations on the $V_{DDA(ADC5V5)}$ pin) are visible as variations in the measurement result. The following formula is used to determine the conversion result of an input voltage $V_I$ on ADC0:

$$\left(\frac{2}{3}\left(V_I - \frac{1}{2}V_{DDA(ADC5V0)}\right) + \frac{1}{2}V_{DDA(ADC3V3)}\right) \times \frac{1024}{V_{VREFP} - V_{VREFN}} \quad (4)$$

**Remark:** The above formula only applies to ADC0.

## 4. Register overview

**Table 385. Register overview: ADC (base address: 0xE00C 2000 (ADC0), 0xE00C 3000 (ADC1), 0xE00C 4000 (ADC2))**

| Name | Access | Address offset | Description[1] | Reset value | Reference |
|---|---|---|---|---|---|
| ACC0 | R/W | 0x000 | ADC channel 0 configuration register | 0x0000 | see Table 26–386 |
| ACC1 | R/W | 0x004 | ADC channel 1 configuration register | 0x0000 | see Table 26–386 |
| ACC2 | R/W | 0x008 | ADC channel 2 configuration register | 0x0000 | see Table 26–386 |
| ACC3 | R/W | 0x00C | ADC channel 3 configuration register | 0x0000 | see Table 26–386 |
| ACC4 | R/W | 0x010 | ADC channel 4 configuration register | 0x0000 | see Table 26–386 |
| ACC5 | R/W | 0x014 | ADC channel 5 configuration register | 0x0000 | see Table 26–386 |
| ACC6 | R/W | 0x018 | ADC channel 6 configuration register | 0x0000 | see Table 26–386 |
| ACC7 | R/W | 0x01C | ADC channel 7 configuration register | 0x0000 | see Table 26–386 |

Table 385. Register overview: ADC (base address: 0xE00C 2000 (ADC0), 0xE00C 3000 (ADC1), 0xE00C 4000 (ADC2)) *…continued*

| Name | Access | Address offset | Description[1] | Reset value | Reference |
|---|---|---|---|---|---|
| ACC8 | R/W | 0x020 | ADC channel 8 configuration register | 0x0000 | see Table 26–386 |
| ACC9 | R/W | 0x024 | ADC channel 9 configuration register | 0x0000 | see Table 26–386 |
| ACC10 | R/W | 0x028 | ADC channel 10 configuration register | 0x0000 | see Table 26–386 |
| ACC11 | R/W | 0x02C | ADC channel 11 configuration register | 0x0000 | see Table 26–386 |
| ACC12 | R/W | 0x030 | ADC channel 12 configuration register | 0x0000 | see Table 26–386 |
| ACC13 | R/W | 0x034 | ADC channel 13 configuration register | 0x0000 | see Table 26–386 |
| ACC14 | R/W | 0x038 | ADC channel 14 configuration register | 0x0000 | see Table 26–386 |
| ACC15 | R/W | 0x03C | ADC channel 15 configuration register | 0x0000 | see Table 26–386 |
| COMP0 | R/W | 0x100 | ADC channel 0 compare register | 0x0000 | see Table 26–387 |
| COMP1 | R/W | 0x104 | ADC channel 1 compare register | 0x0000 | see Table 26–387 |
| COMP2 | R/W | 0x108 | ADC channel 2 compare register | 0x0000 | see Table 26–387 |
| COMP3 | R/W | 0x10C | ADC channel 3 compare register | 0x0000 | see Table 26–387 |
| COMP4 | R/W | 0x110 | ADC channel 4 compare register | 0x0000 | see Table 26–387 |
| COMP5 | R/W | 0x114 | ADC channel 5 compare register | 0x0000 | see Table 26–387 |
| COMP6 | R/W | 0x118 | ADC channel 6 compare register | 0x0000 | see Table 26–387 |
| COMP7 | R/W | 0x11C | ADC channel 7 compare register | 0x0000 | see Table 26–387 |
| COMP8 | R/W | 0x120 | ADC channel 8 compare register | 0x0000 | see Table 26–387 |
| COMP9 | R/W | 0x124 | ADC channel 9 compare register | 0x0000 | see Table 26–387 |
| COMP10 | R/W | 0x128 | ADC channel 10 compare register | 0x0000 | see Table 26–387 |
| COMP11 | R/W | 0x12C | ADC channel 11 compare register | 0x0000 | see Table 26–387 |
| COMP12 | R/W | 0x130 | ADC channel 12 compare register | 0x0000 | see Table 26–387 |
| COMP13 | R/W | 0x134 | ADC channel 13 compare register | 0x0000 | see Table 26–387 |
| COMP14 | R/W | 0x138 | ADC channel 14 compare register | 0x0000 | see Table 26–387 |
| COMP15 | R/W | 0x13C | ADC channel 15 compare register | 0x0000 | see Table 26–387 |
| ACD0 | R | 0x200 | ADC channel 0 conversion data register | 0x0000 | see Table 26–388 |
| ACD1 | R | 0x204 | ADC channel 1 conversion data register | 0x0000 | see Table 26–388 |
| ACD2 | R | 0x208 | ADC channel 2 conversion data register | 0x0000 | see Table 26–388 |
| ACD3 | R | 0x20C | ADC channel 3 conversion data register | 0x0000 | see Table 26–388 |
| ACD4 | R | 0x210 | ADC channel 4 conversion data register | 0x0000 | see Table 26–388 |
| ACD5 | R | 0x214 | ADC channel 5 conversion data register | 0x0000 | see Table 26–388 |
| ACD6 | R | 0x218 | ADC channel 6 conversion data register | 0x0000 | see Table 26–388 |
| ACD7 | R | 0x21C | ADC channel 7 conversion data register | 0x0000 | see Table 26–388 |
| ACD8 | R | 0x220 | ADC channel 8 conversion data register | 0x0000 | see Table 26–388 |
| ACD9 | R | 0x224 | ADC channel 9 conversion data register | 0x0000 | see Table 26–388 |
| ACD10 | R | 0x228 | ADC channel 10 conversion data register | 0x0000 | see Table 26–388 |
| ACD11 | R | 0x22C | ADC channel 11 conversion data register | 0x0000 | see Table 26–388 |
| ACD12 | R | 0x230 | ADC channel 12 conversion data register | 0x0000 | see Table 26–388 |
| ACD13 | R | 0x234 | ADC channel 13 conversion data register | 0x0000 | see Table 26–388 |
| ACD14 | R | 0x238 | ADC channel 14 conversion data register | 0x0000 | see Table 26–388 |

**Table 385. Register overview: ADC (base address: 0xE00C 2000 (ADC0), 0xE00C 3000 (ADC1), 0xE00C 4000 (ADC2))** *…continued*

| Name | Access | Address offset | Description[1] | Reset value | Reference |
|---|---|---|---|---|---|
| ACD15 | R | 0x23C | ADC channel 15 conversion data register | 0x0000 | see Table 26–388 |
| COMP_STATUS | R | 0x300 | Compare-status register | 0x0000 | see Table 26–389 |
| COMP_STATUS_CLR | W | 0x304 | Compare-status clear register | - | see Table 26–390 |
| ADC_CONFIG | R/W | 0x400 | ADC configuration register | 0x0000 | see Table 26–391 |
| ADC_CONTROL | R/W | 0x404 | ADC control register | 0x0000 | see Table 26–392 |
| ADC_STATUS | R | 0x408 | ADC status register | 0x0000 | see Table 26–393 |
| INT_CLR_ENABLE | W | 0xFD8 | Interrupt clear-enable register | - | see Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Interrupt set-enable register | - | see Table 10–92 |
| INT_STATUS | R | 0xFE0 | Interrupt status register | 0x0000 | see Table 10–93 |
| INT_ENABLE | R | 0xFE4 | Interrupt enable register | 0x0000 | see Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | interrupt clear-status register | - | see Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | interrupt set-status register | - | see Table 10–96 |

[1] ADC0, ADC1 and ADC2 have eight analog input pins each and each input has two compare, configuration, and conversion data registers (see Figure 26–115).

## 4.1 ADC channel configuration register

The LPC29xx contains a channel configuration register for each of the 16 ADC channel inputs. These registers define the resolution per channel from 2-bit to 10-bit.

Table 26–386 shows the bit assignment of the ACC0 to ACC15 registers.

**Table 386. ACCn register bit description (ACC0 to 15, addresses 0xE00C 2000 to 0xE00C203C (ADC0), 0xE00C 3000 to 0xE00C303C (ADC1), 0xE00C 4000 to 0xE00C403C (ADC2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 4 | reserved | R | - | Reserved; do not modify. Read as logic 0 |

**Table 386. ACCn register bit description (ACC0 to 15, addresses 0xE00C 2000 to 0xE00C203C (ADC0), 0xE00C 3000 to 0xE00C303C (ADC1), 0xE00C 4000 to 0xE00C403C (ADC2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 3 to 0 | ACC[3:0] | R/W | | Set the resolution for a channel |
| | | | 0000* | Channel not selected |
| | | | 0001 | Reserved |
| | | | 0010 | 2-bit resolution |
| | | | 0011 | 3-bit resolution |
| | | | 0100 | 4-bit resolution |
| | | | 0101 | 5-bit resolution |
| | | | 0110 | 6-bit resolution |
| | | | 0111 | 7-bit resolution |
| | | | 1000 | 8-bit resolution |
| | | | 1001 | 9-bit resolution |
| | | | 1010 | 10-bit resolution |
| | | | 1011 | Reserved |
| | | | : | : |
| | | | 1111 | Reserved |

## 4.2 ADC channel-compare register

The LPC29xx contains a compare register for each of the ADC channel inputs. These registers contain the value with which the ADC_R_x data is to be compared. The comparison can be done for values ‘less than’ or ‘greater than or equal to’ the compare value as defined in the match part of the register. The COMP registers can be updated ‘on the fly’. The comparison check is done at the end of a scan when new ADC_R_x data is available.

Table 26–387 shows the bit assignment of the COMP0 to COMP15 registers.

**Table 387. COMPn register bit description (COMP0 to 15, addresses 0xE00C 2100 to 0xE00C213C (ADC0), 0xE00C 3100 to 0xE00C313C (ADC1), 0xE00C 4100 to 0xE00C413C (ADC2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 18 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 17 and16 | MATCH[1:0] | R/W | | Compare for values ‘less than’ or ‘greater than or equal to’ the compare value |
| | | | 00* | No comparison is done |
| | | | 01 | Unused |
| | | | 10 | Interrupt is generated when ADC data is less than compare data |
| | | | 11 | Interrupt is generated when ADC data is greater than or equal to compare data |

**Table 387. COMPn register bit description (COMP0 to 15, addresses 0xE00C 2100 to 0xE00C213C (ADC0), 0xE00C 3100 to 0xE00C313C (ADC1), 0xE00C 4100 to 0xE00C413C (ADC2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 15 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 to 0 | COMP_R | R/W | | Compare data with respect to analog input channel |
| | | | 0x00* | |

## 4.3 ADC channel conversion data register

The LPC29xx contains a conversion data register for each of the ADC channel inputs. These registers store the result of an analog-to-digital conversion scan. The selected bit resolution in the ADC channel configuration register simultaneously defines the number of valid most-significant conversion data bits in the ADC channel conversion data register. The remaining conversion data bits become logic 0 accordingly.

The ACD register is read only. Table 26–388 shows the bit assignment of the 16 ACD registers.

**Table 388. ACD register bit description addresses 0xE00C 2200 to 0xE00C223C (ADC0), 0xE00C 3200 to 0xE00C323C (ADC1), 0xE00C 4200 to 0xE00C423C (ADC2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 10 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 9 to 0 | ACD[9:0] | R | | 10-bit conversion data. The value represents the voltage on the corresponding channel input pin, divided by the voltage on the $V_{DDA(ADC5V)}$ pin (for ADC0) or $V_{DDA(ADC3V3)}$ pin (for ADC1 and ADC2). Figure 26–115 shows the assignment of an input pin to each of its two channels. |
| | | | 0x000* | |

## 4.4 Compare status register

The compare status register indicates which channels had a compare match (logic 1) and which not (logic 0). Note that the compare function is located in the system domain. The COMP_STATUS register is updated after each scan one MSCSS subsystem clock cycle after the ADC scan has finished.

Table 26–389 shows the bit assignment of the COMP_STATUS register.

**Table 389. COMP_STATUS register bit description (COMP_STATUS addresses 0xE00C 2300 (ADC0), 0xE00C 3300 (ADC1), 0xE00C 4300 (ADC2))**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify, read as logic 0 |
| 15 | COMP_STATUS_15 | R | 1 | Compare match of channel 15 |
| | | | 0* | No compare match of channel 15 |

**Table 389. COMP_STATUS register bit description (COMP_STATUS addresses 0xE00C 2300 (ADC0), 0xE00C 3300 (ADC1), 0xE00C 4300 (ADC2))** *…continued*

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| : | : | : | : | : |
| 0 | COMP_STATUS_0 | R | 1 | Compare match of channel 0 |
| | | | 0* | No compare match of channel 0 |

## 4.5 Compare-status clear register

Writing a 1 to the compare-status clear register clears that specific bit in the COMP_STATUS Register.

Table 26–390 shows the bit assignment of the COMP_STATUS_CLR register.

**Table 390. COMP_STATUS_CLR register bit description COMP_STATUS_CLR, addresses 0xE00C 2304 (ADC0), 0xE00C 3304 (ADC1), 0xE00C 4304 (ADC2)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 | COMP_STATUS_CLR_15 | W | 1 | Clears the compare match of channel 15 |
| : | : | : | : | : |
| 0 | COMP_STATUS_CLR_0 | W | 1 | Clears the compare match of channel 0 |

## 4.6 ADC configuration register

The ADC configuration register configures the ADC operation modes.

Bit 0 configures the operation mode. This can be either single or continuous. In single mode (0), one scan of all the selected channels is performed. In continuous mode (1), the next scan for all selected inputs is started once the previous scan has been completed.

Bit 1 configures the power-down mode: when set to 0, the ADC is internally put into Power-down mode when no conversion is being done: when set to 1 the ADC is never put into power-down mode.

Bits 7 to 15 configure the enabling of the several start inputs per ADC: start 0 to start 3. Setting to 1 enables the start. When enabled, start 0 and start 2 need a pulse which takes at least one system clock cycle; start 1 and start 3 need a pulse which takes at least one ADC clock cycle.

Transfer of configuration to the ADC domain starts as soon as the update bit in the ADC_CONTROL register is set to 1 (see Section 26–4.7). The update bit being zero again indicates that the transfer is ready. When the update bit is set it is not possible to write to the ADC_CONTROL registers.

Table 26–391 shows the bit assignment of the ADC_CONFIG register.

**Table 391. ADC_CONFIG register bit description (ADC_CONFIG addresses, 0xE00C 2400 (ADC0), 0xE00C 3400 (ADC1), 0xE00C 4400 (ADC2)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 | NEGEDGE_START_3 | R/W | 1 | Enable ADC starting on the negative edge of start 3. The match output x of MSCSS timer 0, x is equal to ADC number |
| | | | 0* | |
| 14 | POSEDGE_START_3 | R/W | 1 | Enable ADC starting on the positive edge of start 3. The match output x of MSCSS timer 0, x is equal to ADC number[1] |
| | | | 0* | |
| 13 | NEGEDGE_START_2 | R/W | 1 | Enable ADC starting on the negative edge of start 2. The sync output of PWM x, x is equal to ADC number[2] |
| | | | 0* | |
| 12 | POSEDGE_START_2 | R/W | 1 | Enable ADC starting on the positive edge of start 2. The sync output of PWM x, x is equal to ADC number[2] |
| | | | 0* | |
| 11 | NEGEDGE_START_1 | R/W | 1 | Enable ADC starting on the negative edge of start 1: sync_out signal from preceding ADC converter[1] |
| | | | 0* | |
| 10 | POSEDGE_START_1 | R/W | 1 | Enable ADC starting on the positive edge of start 1: sync_out signal from preceding ADC converter[3] |
| | | | 0* | |
| 9 | NEGEDGE_START_0 | R/W | 1 | enable ADC starting on the negative edge of start 0: ADCx_EXT_START input pin[2] |
| | | | 0* | |
| 8 | POSEDGE_START_0 | R/W | 1 | Enable ADC starting on the positive edge of start 0: ADCx_EXT_START input pin[2] |
| | | | 0* | |
| 7 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | ADC_PD | R/W | 1 | The ADC is **never** put into Power-down mode. |
| | | | 0* | The ADC is internally put into Power-down mode when no conversion is executed. |
| 0 | ADC_CSCAN | R/W | | ADC continuous scan |
| | | | 1 | Continuous scan |
| | | | 0* | Single scan |

[1] Start 1 and start 3 are captured in the ADC clock domain, minimum pulse width is two ADC clock periods.

[2] Start 0 and start 2 are captured in the system clock domain, minimum pulse width is two system clock periods.

[3] Only for ADC0: ADC1 and ADC2 do not have a calibration mode.

## 4.7 ADC control register

The ADC_CONTROL register controls the ADC operation modes. It contains three bits.

- The start bit (0): when set to 1 the ADC conversion is started.
- The stop bit (1): when set to 1 the conversion is stopped at the end of the next scan. The stop bit being 0 again indicates that the ADC conversion has been stopped at the end of a scan.
- The update bit (2): when set to 1 the configuration and resolution are copied to the ADC clock domain. This is done immediately when the ADC is in IDLE mode. In continuous mode, copying of the configuration is done at the end of the scan. The update bit being 0 again indicates that the configuration has been copied to the ADC domain.

Table 26–392 shows the bit assignment of the ADC_CONTROL register.

**Table 392. ADC_CONTROL register bit description (ADC_CONTROL addresses, 0xE00C 2404 (ADC0), 0xE00C 3404 (ADC1), 0xE00C 4404 (ADC2)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 3 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 2 | UPDATE | R/W | 1 | Copy the configuration |
| | | | 0* | |
| 1 | STOP | R/W | 1 | Stop ADC conversion |
| | | | 0* | |
| 0 | START | R/W | 1 | Start ADC conversion |
| | | | 0* | |

## 4.8 ADC status register

The ADC status register consists of two bits.

- The ADC_STATUS bit (bit 0): this bit indicates whether an ADC scan is in progress (bit is set to 1) or not (bit is set to 0). When the configuration is set to internal trigger (ADC_CONFIG[3:0] = 0000) the ADC_STATUS bit will be asserted when the start bit is set. When the configuration is set to external start triggering, the ADC_STATUS bit will be asserted when the ADC conversion is started as a result of the selected external start event. The ADC_STATUS bit is set to logic 0 when the conversion is stopped and the results are available in the ACD registers.
- The ADC_CONFIG bit (bit 1): this bit indicates whether the data in the ACD register corresponds to the loaded configuration (bit is set to 0) or to an old configuration (bit is set to 1). A configuration is assumed to be loaded as soon as the update bit in the ADC_CONTROL register is set.

Table 26–393 shows the bit assignment of the ADC_STATUS register.

**Table 393. ADC_STATUS register bit description (ADC_STATUS addresses, 0xE00C 2408 (ADC0), 0xE00C 3408 (ADC1), 0xE00C 4408 (ADC2)**

** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 2 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 1 | ADC_CONFIG | R | 1 | Indicates that the ACD register corresponds to the loaded configuration |
| | | | 0* | Indicates that the ACD register corresponds to an older configuration |
| 0 | ADC_STATUS | R | 1 | ADC conversion in progress |
| | | | 0* | ADC conversion not in progress |

## 4.9 ADC interrupt bit description

Table 26–394 gives the interrupts for the analog-to-digital converter. The first column gives the bit number in the interrupt registers. For a general explanation of the interrupt concept and a description of the registers see Section 10–6.

**Table 394. ADC interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 2 | unused | Unused |
| 1 | COMPARE | Compare match |
| 0 | SCAN | End of scan |

# UM10316

# Chapter 27: LPC29xx Quadrature Encoder Interface (QEI)

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

The MSCSS includes a quadrature encoder interface (QEI) with the following features:

- Tracks encoder position.
- Increments/ decrements depending on direction.
- Programmable for 2x or 4x position counting.
- Velocity capture using built-in timer.
- Velocity compare function with less than interrupt.
- Uses 32-bit registers for position and velocity.
- Three position compare registers with interrupts.
- Index counter for revolution counting.
- Index compare register with interrupts.
- Can combine index and position interrupts to produce an interrupt for whole and partial revolution displacement.
- Digital filter with programmable delays for encoder input signals.
- Can accept decoded signal inputs (clk and direction).
- Connected to APB.

## 3. Introduction

A quadrature encoder, also known as a 2-channel incremental encoder, converts angular displacement into two pulse signals. By monitoring both the number of pulses and the relative phase of the two signals, you can track the position, direction of rotation, and velocity. In addition, a third channel, or index signal, can be used to reset the position counter. This quadrature encoder interface module decodes the digital pulses from a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.

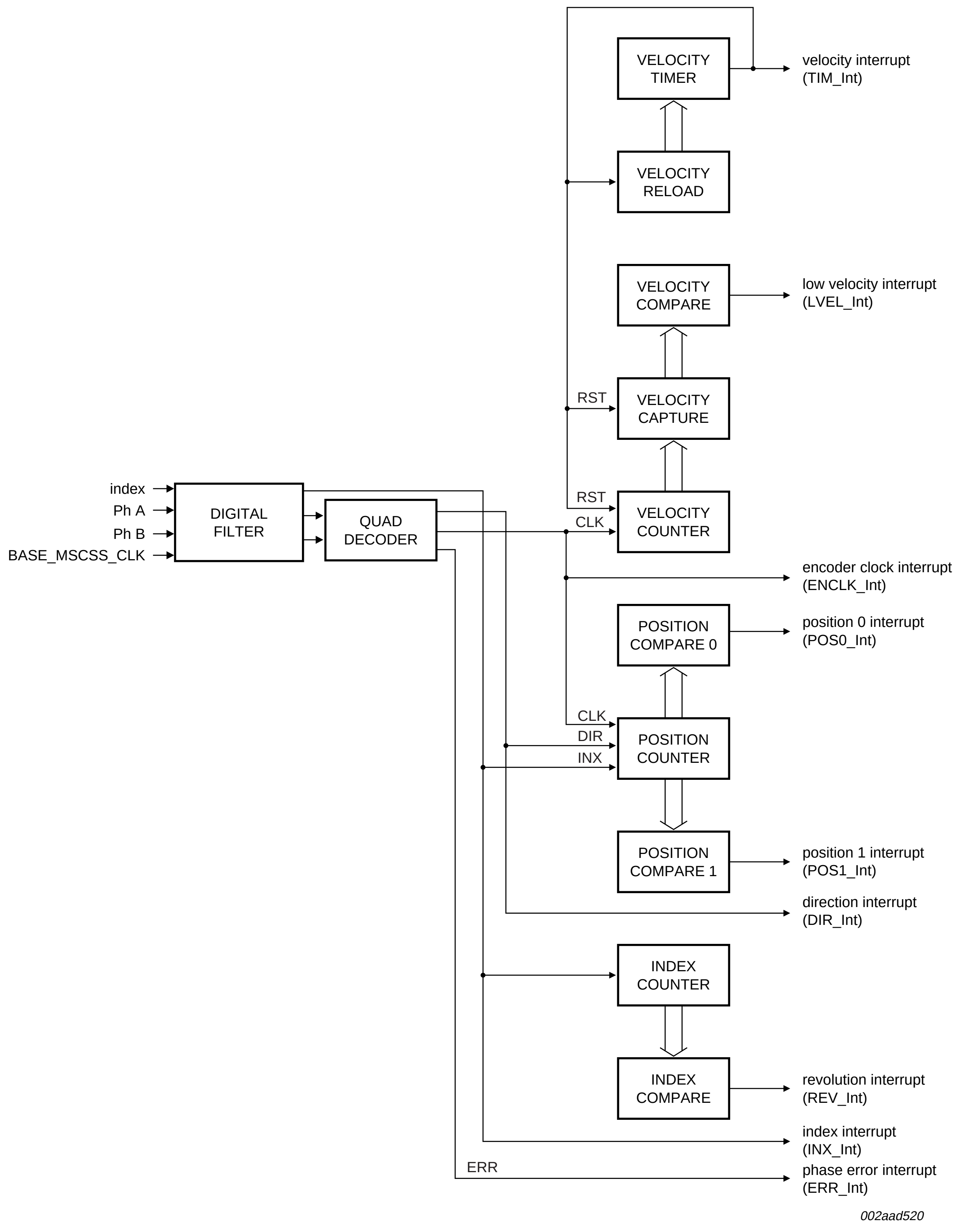

VELOCITY TIMER
velocity interrupt (TIM_Int)
VELOCITY RELOAD
VELOCITY COMPARE
low velocity interrupt (LVEL_Int)
RST
VELOCITY CAPTURE
index
Ph A
Ph B
BASE_MSCSS_CLK
DIGITAL FILTER
QUAD DECODER
RST
CLK
VELOCITY COUNTER
encoder clock interrupt (ENCLK_Int)
POSITION COMPARE 0
position 0 interrupt (POS0_Int)
CLK
DIR
INX
POSITION COUNTER
POSITION COMPARE 1
position 1 interrupt (POS1_Int)
direction interrupt (DIR_Int)
INDEX COUNTER
INDEX COMPARE
revolution interrupt (REV_Int)
index interrupt (INX_Int)
ERR
phase error interrupt (ERR_Int)
002aad520

**Fig 116.Encoder interface block diagram**

## 4. Functional description

The QEI module interprets the two-bit gray code produced by a quadrature encoder wheel to integrate position over time and determine direction of rotation. In addition, it can capture the velocity of the encoder wheel.

### 4.1 Input signals

The QEI module supports two modes of signal operation: quadrature phase mode and clock/direction mode. In quadrature phase mode, the encoder produces two clocks that are 90 degrees out of phase; the edge relationship is used to determine the direction of rotation. In clock/direction mode, the encoder produces a clock signal to indicate steps and a direction signal to indicate the direction of rotation. ).

This mode is determined by the SigMode bit of the QEI Control (QEICON) register (See Table 27–400). When the SigMode bit = 1, the quadrature decoder is bypassed and the PhA pin functions as the direction signal and PhB pin functions as the clock signal for the counters, etc. When the SigMode bit = 0, the PhA pin and PhB pins are decoded by the quadrature decoder. In this mode the quadrature decoder produces the direction and clock signals for the counters, etc. In both modes the direction signal is subject to the effects of the direction invert (DIRINV) bit.

#### 4.1.1 Quadrature input signals

When edges on PhA lead edges on PhB , the position counter is incremented. When edges on PhB lead edges on PhA , the position counter is decremented. When a rising and falling edge pair is seen on one of the phases without any edges on the other, the direction of rotation has changed.

**Table 395. Encoder states**

| Phase A | Phase B | state |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 1 | 2 |
| 0 | 1 | 3 |
| 0 | 0 | 4 |

**Table 396. Encoder state transitions[1]**

| from state | to state | Direction |
|---|---|---|
| 1 | 2 | positive |
| 2 | 3 | |
| 3 | 4 | |
| 4 | 1 | |
| 4 | 3 | negative |
| 3 | 2 | |
| 2 | 1 | |
| 1 | 4 | |

[1] All other state transitions are illegal and should set the ERR bit.

Interchanging of the PhA and PhB input signals are compensated by complementing the DIR bit. When set = 1, the direction invertion bit (DIRINV) complements the DIR bit.

Table 397. Encoder direction

| DIR bit | DIRINV bit | direction |
|---|---|---|
| 0 | 0 | forward |
| 1 | 0 | reverse |
| 0 | 1 | reverse |
| 1 | 1 | forward |

### 4.1.2 Digital input filtering

All three encoder inputs (PhA, PhB, and index) require digital filtering. The number of sample clocks is user programmable from 1 to 4,294,967,295. In order for a transition to be accepted, the input signal must remain in new state for the programmed number of sample clocks.

## 4.2 Position capture

The capture mode for the position integrator can be set to update the position counter on every edge of the PhA signal or to update on every edge of both PhA and PhB. Updating the position counter on every PhA and PhB provides more positional resolution at the cost of less range in the positional counter.

The position integrator and velocity capture can be independently enabled. Alternatively, the phase signals can be interpreted as a clock and direction signal as output by some encoders.

The position counter is automatically reset on one of two conditions. Incrementing past the maximum position value (QEIMAXPOS) will reset the position counter to zero. If the reset on index bit (RESPI) is set, sensing the index pulse will reset the position counter to zero.

## 4.3 Velocity capture

The velocity capture has a programmable timer and a capture register. It counts the number of phase edges (using the same configuration as for the position integrator) in a given time period. When the velocity timer (QEITIME) overflows the contents of the velocity counter (QEIVEL) are transfered to the capture (QEICAP) register. The velocity counter is then cleared. The velocity timer is loaded with the contents of the velocity reload register (QEILOAD). Finally, the velocity interrupt (TIM_Int) is asserted. The number of edges counted in a given time period is directly proportional to the velocity of the encoder. Setting the reset velocity bit (RESV) has the same effect as an overflow of the velocity timer, except that the setting the RESV bit will not generate a velocity interrupt.

Figure 27–117 shows how the quadrature encoder converts the phase input signals into clock pulses, the direction signal, and the encoder clock (in 4x mode).

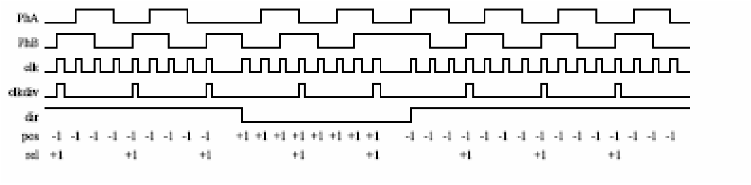

PhA
PhB
clk
clkdiv
dir
pos -1 -1 -1 -1 -1 -1 -1 -1 -1 +1 +1 +1 +1 +1 +1 +1 +1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
vel +1 +1 +1 +1 +1 +1 +1 +1

**Fig 117.Encoder and velcoity divider operation**

The following equation converts the velocity counter value into an rpm value:

```
rpm = (clock * (2 ^ VelDiv) * Speed * 60) ÷ (Load * ppr * edges)
```

where:

- clock is the controller clock rate
- ppr is the number of pulses per revolution of the physical encoder
- edges is 2 or 4, based on the capture mode set in the QEICON register (2 for CapMode set to 0 and 4 for CapMode set to 1)

For example, consider a motor running at 600 rpm. A 2048 pulse per revolution quadrature encoder is attached to the motor, producing 8192 phase edges per revolution. With a velocity predivider of ÷1 (VelDiv set to 0) and clocking on both PhA and PhB edges, this results in 81,920 pulses per second (the motor turns 10 times per second). If the timer were clocked at 10,000 Hz, and the loadvalue was 2,500 (¼ of a second), it would count 20,480 pulses per update. Using the above equation:

rpm = (10000 * 1 * 20480 * 60) ÷ (2500 * 2048 * 4) = 600 rpm

Now, consider that the motor is sped up to 3000 rpm. This results in 409,600 pulses per second, or 102,400 every ¼ of a second. Again, the above equation gives:

rpm = (10000 * 1 * 102400 * 60) ÷ (2500 * 2048 * 4) = 3000 rpm

## 4.4 Velocity compare

In addition to velocity capture, the velocity measurement system includes a programmable velocity compare register. After every velocity capture event the contents of the velocity capture register (QEICAP) is compared with the contents of the velocity compare register (VELCOMP). If the captured velocity is less than the compare value an interrupt is asserted provided that the velocity compare interrupt enable bit is set. This can be used to determine if a motor shaft is either stalled or moving too slow.

## 5. Pin description

**Table 398. QEI pin description**

| Pin name | I/O | Description |
|---|---|---|
| PHA | I | |
| PHB | I | |
| IDX | I | |

## 6. Register overview

**Table 399. Register overview: QEI (base address 0xE00C 9000)**

| Symbol | Access | Address offset | Description |
|---|---|---|---|
| **Control registers** | | | |
| QEICON | W | 0x00 | Control register |
| QEISTAT | R | 0x04 | Encoder status register |
| QEICONF | R/W | 0x08 | Configuration register |
| **Position, index, and timer registers** | | | |
| QEIPOS | R | 0x0C | Position register |
| QEIMAXPSOS | R/W | 0x10 | Maximum position register |
| CMPOS0 | R/W | 0x14 | position compare register 0 |
| CMPOS1 | R/W | 0x18 | position compare register 1 |
| CMPOS2 | R/W | 0x1C | position compare register 2 |
| INXCNT | R | 0x20 | Index count register |
| INXCMP | R/W | 0x24 | Index compare register |
| QEILOAD | R/W | 0x28 | Velocity timer reload register |
| QEITIME | R | 0x2C | Velocity timer register |
| QEIVEL | R | 0x30 | Velocity counter register |
| QEICAP | R | 0x34 | Velocity capture register |
| VELCOMP | R/W | 0x38 | Velocity compare register |
| FILTER | R/W | 0x3C | Digital filter register |
| **Interrupt registers** | | | |
| QEIIES | W | 0xFDC | Interrupt enable set register |
| QEIIEC | W | 0xFD8 | Interrupt enable clear register |
| QEIINTSTAT | R | 0xFE0 | Interrupt status register |
| QEIIE | R | 0xFE4 | Interrupt enable register |
| QEICLR | W | 0xFE8 | Interrupt status clear register |
| QEISET | W | 0xFEC | Interrupt status set register |

### 6.1 QEI Control (QEICON)

This register contains bits which control the operation of the position and velocity counters of the QEI module. This register can be set by software, but only hardware can reset the register bits.

Table 400. QEI Control Register (QEICON - address 0xE00C 9000)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 4:31 | - | reserved | 0 |
| 3 | RESI | Reset index counter. When set = 1, resets the index counter to all zeros. Autoclears when the index counter is cleared. | 0 |
| 2 | RESV | Reset velocity. When set = 1, resets the the velocity counter to all zeros and reloads the velocity timer. Autoclears when the velocity counter is cleared. | 0 |
| 1 | RESPI | Reset position counter on index. When set = 1, resets the the position counter to all zeros when an index pulse occurs. Autoclears when the position counter is cleared. | 0 |
| 0 | RESP | Reset position counter. When set = 1, resets the position counter to all zeros. Autoclears when the position counter is cleared. | 0 |

## 6.2 QEI Configuration (QEICONF)

This register contains the configuration of the QEI module.

Table 401. QEI Contfiguration Register (QEICONF - address 0xE00C 9008)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 4:31 | - | reserved | 0 |
| 3 | INVINX | Invert Index. When set, inverts the sense of the index input. | 0 |
| 2 | CAPMODE | Capture Mode. When = 0, only PhA edges are counted (2X). When = 1, BOTH PhA and PhB edges are counted (4X), increasing resolution but decreasing range. | 0 |
| 1 | SIGMODE | Signal Mode. When = 0, PhA and PhB function as quadrature encoder inputs. When = 1, PhA functions as the direction signal and PhB functions as the clock signal. | 0 |
| 0 | DIRINV | Direction invert. When = 1, complements the DIR bit. | 0 |

## 6.3 QEI Status (QEISTAT)

This register provides the status of the encoder interface.

Table 402. QEI Interrupt Status Register (QEISTAT - address 0xE00C 9004)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 1:31 | - | reserved | 0 |
| 0 | DIR | Direction bit. In combination with DIRINV bit indicates forward or reverse direction. See Table 27–397. | |

## 6.4 QEI Position (QEIPOS)

This register contains the current value of the encoder position. Increments or decrements when encoder counts occur, depending on the direction of rotation.

Table 403. QEI Position Register (QEIPOS - address 0xE00C 900C)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

## 6.5 QEI Maximum Position (QEIMAXPOS)

This register contains the maximum value of the encoder position. In forward rotation the position register resets to zero when the position register exceeds this value. In reverse rotation the position register resets to this value when the position register decrements from zero.

Table 404. QEI Maximum Position Register (QEIMAXPOS - address 0xE00C 9010)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current maximum position value. | 0 |

## 6.6 QEI Position Compare 0 (CMPOS0)

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 405. QEI Position Compare Register 0 (CMPOS0 - address 0xE00C 9014)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

## 6.7 QEI Position Compare 1 (CMPOS1)

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 406. QEI Position Compare Register 1 (CMPOS1 - address 0xE00C 9018)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

## 6.8 QEI Position Compare 2 (CMPOS2)

This register contains a position compare value. This value is compared against the current value of the position register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the position register.

Table 407. QEI Position Compare Register 2 (CMPOS2 - address 0xE00C 901C)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

### 6.9 QEI Index Count (INXCNT)

This register contains the current value of the encoder position. Increments or decrements when encoder counts occur, depending on the direction of rotation.

**Table 408. QEI Index Count Register (CMPOS - address 0xE00C 9020)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

### 6.10 QEI Index Compare (INXCMP)

This register contains an index compare value. This value is compared against the current value of the index count register. Interrupts can be enabled to interrupt when the compare value is less than, equal to, or greater than the current value of the index count register.

**Table 409. QEI Index Compare Register (CMPOS - address 0xE00C 9024)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current position value. | 0 |

### 6.11 QEI Timer Reload (QEILOAD)

This register contains the reload value of the velocity timer. When the timer (QEITIME) overflows or the RESV bit is asserted, this value is loaded into the timer (QEITIME).

**Table 410. QEI Timer Load Register (QEILOAD - address 0xE00C 9028)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current velocity timer load value. | 0 |

### 6.12 QEI Timer (QEITIME)

This register contains the current value of the velocity timer. When this timer overflows the value of velocity counter (QEIVEL) is stored in the velocity capture register (QEICAP), the velocity counter is reset to zero, the timer is reloaded with the value stored in the velocity reload register (QEILOAD), and the velocity interrupt (TIM_Int) is asserted.

**Table 411. QEI Timer Register (QEITIME - address 0xE00C 902C)**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current velocity timer value. | 0 |

### 6.13 QEI Velocity (QEIVEL)

This register contains the running count of velocity pulses for the current time period. When the velocity timer (QEITIME) overflows the contents of this register is captured in the velocity capture register (QEICAP). After capture, this register is set to zero. This register is also reset when the velocity reset bit (RESV) is asserted.

Table 412. QEI Velocity Register (QEIVEL - address 0xE00C 9030)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current velocity pulse count. | 0 |

## 6.14 QEI Velocity Capture (QEICAP)

This register contains the most recently measured velocity of the encoder. This corresponds to the number of velocity pulses counted in the previous velocity timer period.The current velocity count is latched into this register when the velocity timer overflows.

Table 413. QEI Velocity Capture Register (QEICAP - address 0xE00C 9034)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current velocity pulse count. | 0 |

## 6.15 QEI Velocity Compare (VELCOMP)

This register contains a velocity compare value. This value is compared against the captured velocity in the velocity capture register. If the capture velocity is less than the value in this compare register, a velocity compare interrupt (VELC_Int) will be asserted, if enabled.

Table 414. QEI Velocity Compare Register (VELCOMP - address 0xE00C 9038)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Current velocity pulse count. | 0 |

## 6.16 QEI Digital Filter (FILTER)

This register contains the sampling count for the digital filter. A sampling count of zero bypasses the filter.

Table 415. QEI Digital Filter Register (FILTER - address 0xE00C 903C)

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0:31 | - | Digital filter sampling delay | 0x0 |

## 6.17 Interrupt registers

### 6.17.1 QEI interrupt bit description

Table 27–416 contains all QEI interrupts which are combined into one interrupt in the VIC. See Section 10–6.1 for a description of the interrupt control registers. The interrupt bits listed in Table 27–416 apply to all interrupt registers.

Table 416. QEI Interrupt bits

| Bit | Symbol | Description |
|---|---|---|
| 13:31 | - | reserved |
| 12 | POS2REV_Int | Combined position 2 and revolution count interrupt. Set when both the POS2_Int bit is set and the REV_Int is set. |
| 11 | POS1REV_Int | Combined position 1 and revolution count interrupt. Set when both the POS1_Int bit is set and the REV_Int is set. |
| 10 | POS0REV_Int | Combined position 0 and revolution count interrupt. Set when both the POS0_Int bit is set and the REV_Int is set. |
| 9 | REV_Int | Indicates that the index compare value is equal to the current index count. |
| 8 | POS2_Int | Indicates that the position 2 compare value is equal to the current position. |
| 7 | POS1_Int | Indicates that the position 1 compare value is equal to the current position. |
| 6 | POS0_Int | Indicates that the position 0 compare value is equal to the current position. |
| 5 | ENCLK_Int | Indicates that and encoder clock pulse was detected. |
| 4 | ERR_Int | Indicates that an encoder phase error was detected. |
| 3 | DIR_Int | Indicates that a change of direction was detected. |
| 2 | VELC_Int | Indicates that captured velocity is less than compare velocity. |
| 1 | TIM_Int | Indicates that a velocity timer overflow occured |
| 0 | INX_Int | Indicates that an index pulse was detected. |

# UM10316
# Chapter 28: LPC29xx Flash/EEPROM

Rev. 3 — 19 October 2010 User manual

## 1. How to read this chapter

Flash configurations vary for the different LPC29xx parts.

Table 417. Feature overview

| Part | Flash size | EEPROM size | Notes |
|---|---|---|---|
| LPC2917/01 | 512 kB | 16 kB | - |
| LPC2919/01 | 768 kB | 16 kB | - |
| LPC2921 | 128 kB | 16 kB | - |
| LPC2923 | 256 kB | 16 kB | - |
| LPC2925 | 512 kB | 16 kB | - |
| LPC2926 | 256 kB | 16 kB | - |
| LPC2927 | 512 kB | 16 kB | - |
| LPC2929 | 768 kB | 16 kB | - |
| LPC2930 | - | - | To reduce power consumption use the FS_PD bit in the FCTR register to power down boot ROM after the boot process has finished. |
| LPC2939 | 768 kB | 16 kB | - |

## 2. Flash memory/EEPROM controller functional description

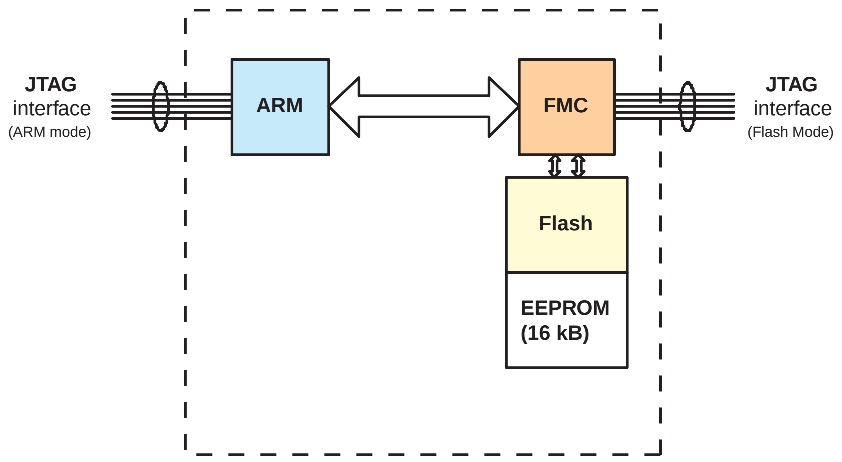


Fig 118. Schematic representation of the FMC

The flash memory consists of the embedded flash memory (flash), EEPROM, and a controller (the FMC) to control access to both. The EEPROM for the LPC29xx contains one 16 kB EEPROM block. The controller can be accessed in two ways: either by register access in software, running on the ARM core, or directly via the JTAG interface Figure 28–118.

In the following sections access to the Flash Memory Controller via software is described. Access via the JTAG interface is described in Section 29–1.

## 2.1 Flash memory layout

The flash memory is arranged into sectors, pages, and flash-words (Figure 28–119). For writing (erase/burn) the following issues are relevant:

- Erasing is performed per sector.
- Protection against erase/burn is arranged per sector.
- Burning - the actual write into flash memory - is performed per page.
- The smallest part that can be written at a time is a flash-word (16 bytes).

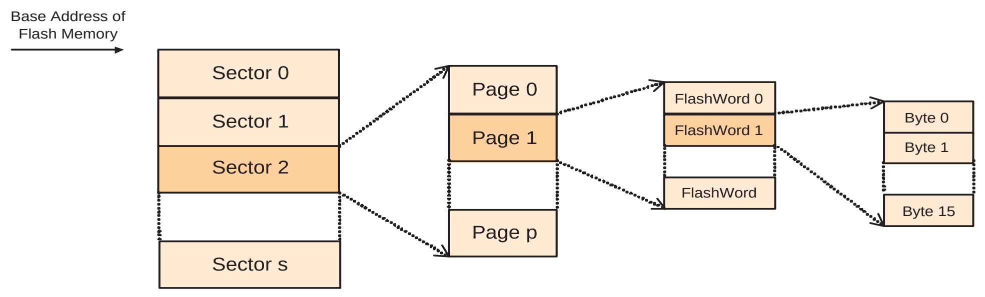


**Fig 119. Flash memory layout**

Table 28–418 lists the various parameters of the flash memory.

**Table 418. Flash memory layout**

| Type number | Flash size | Sector | | Page (per sector) | | Flash-word (per page) | |
|---|---|---|---|---|---|---|---|
| | | # small/ large | Size small/large | # small/ large | Size | # | Size |
| LPC2919/01, LPC2929, LPC2939 | 768 kB | 8/11 | 8192/65536 | 16/128 | 512 byte | 32 | 16 byte |
| LPC2917/01, LPC2925, LPC2927 | 512 kB | 8/7 | 8192/65536 | 16/128 | 512 byte | 32 | 16 byte |
| LPC2923 | 256 kB | 8/3 | 8192/65536 | 16/128 | 512 byte | 32 | 16 byte |
| LPC2921 | 128 kB | 8/1 | 8192/65536 | 16/128 | 512 byte | 32 | 16 byte |

Table 419. Flash sector overview

| Sector number | Sector size (kB) | Sector base address | LPC2921 | LPC2923 | LPC2925 | LPC2926 | LPC2927 | LPC2929 | LPC2939 | LPC2917/01 | LPC2919/01 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 8 | 0x2000 0000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 12 | 8 | 0x2000 2000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 13 | 8 | 0x2000 4000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 14 | 8 | 0x2000 6000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 15 | 8 | 0x2000 8000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 16 | 8 | 0x2000 A000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 17 | 8 | 0x2000 C000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 18 | 8 | 0x2000 E000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 0 | 64 | 0x2001 0000 | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| 1 | 64 | 0x2002 0000 | no | yes | yes | yes | yes | yes | yes | yes | yes |
| 2 | 64 | 0x2003 0000 | no | yes | yes | yes | yes | yes | yes | yes | yes |
| 3 | 64 | 0x2004 0000 | no | no | yes | no | yes | yes | yes | yes | yes |
| 4 | 64 | 0x2005 0000 | no | no | yes | no | yes | yes | yes | yes | yes |
| 5 | 64 | 0x2006 0000 | no | no | yes | no | yes | yes | yes | yes | yes |
| 6 | 64 | 0x2007 0000 | no | no | yes | no | yes | yes | yes | yes | yes |
| 7 | 64 | 0x2008 0000 | no | no | no | no | no | yes | yes | no | yes |
| 8 | 64 | 0x2009 0000 | no | no | no | no | no | yes | yes | no | yes |
| 9 | 64 | 0x200A 0000 | no | no | no | no | no | yes | yes | no | yes |
| 10 | 64 | 0x200B 0000 | no | no | no | no | no | yes | yes | no | yes |

## 2.2 Flash memory reading

During a read (e.g. read-only data or program execution) no special actions are required. The address space of flash memory can simply be accessed like normal ROM with word, half-word, or byte access. It is possible however to modify or optimize the read settings of the flash memory.

For optimal read performance the flash memory contains two internal 128-bit buffers. The configuration of these buffers and the number of wait-states for unbuffered reads can be set in the FMC, see Ref. 31–1. For a detailed description of the flash bridge wait-states register see Table 28–427.

## 2.3 Flash memory writing

In the following section the typical write (erase and burn) sequences are listed. See Section 28–4 for details.

Writing can be split into two parts, erasing and burning. Both operations are asynchronous; i.e. after initiating the operation it takes some time to complete. Erasing is a relatively time-consuming process, see Ref. 31–1. During this process any access to the flash memory results in wait-states. To serve interrupts or perform other actions, this critical code must be present outside the flash memory (e.g. internal RAM). The code that initiates the erase/burn operation must also be present outside the flash memory.

Normally the sectors are protected against write actions. Before a write is started, the corresponding sector(s) must be unprotected, after which protection can be enabled again. Protection is automatically enabled on a reset. During a write (erase/burn) operation the internal clock of the flash must be enabled. After completion the clock can be disabled again.

### 2.3.1 Erase sequence (for one or more sectors)

In the following section the typical erase sequences are listed. See Section 28–4 for details.

- Unprotect sector(s) to be erased.
- Mark sector(s) to be erased.
- Initiate the erase process.
- Wait until erasing is finished, see Section 28–2.5.
- Protect sector(s) (optional).

**Remark:** During the erase process the internal clock of the flash module must be enabled.

### 2.3.2 Burn sequence (for one or more pages)

Burning data into the flash memory is a two-stage process. First, the data for a page is written into data latches, and afterwards the contents of these data latches (single page) are burned into memory. If only a part of a page has to be burned, the contents of the data latches must be preset with logical 1s to avoid changing the remainder of the page. Presetting these latches is done via the FMC (see Section 28–2.6).

In the following section the typical burn sequences are shown. See Section 28–4 for details.

- Unprotect the sectors containing the pages to be burned.
- For each page:
  - Preset the data latches of the flash module (only required if a part of a page has to be programmed; otherwise optional).
  - Write data for the page into the data latches (ordinary 32-bit word writes to the address space of the flash memory).

    **Remark:** Data must be written from flash-word boundaries onwards and must be a multiple of a flash-word.
  - Initiate the burn process.
  - Wait until burning is finished, see Section 28–2.5.
- Protect sectors (optional).

**Remark:** During the burn process the internal clock of the flash module must be enabled.

**Remark:** Only erased flash-word locations can be written to.

**Remark:** A complete page should be burned at one time. Before burning it again the corresponding sector should be erased.

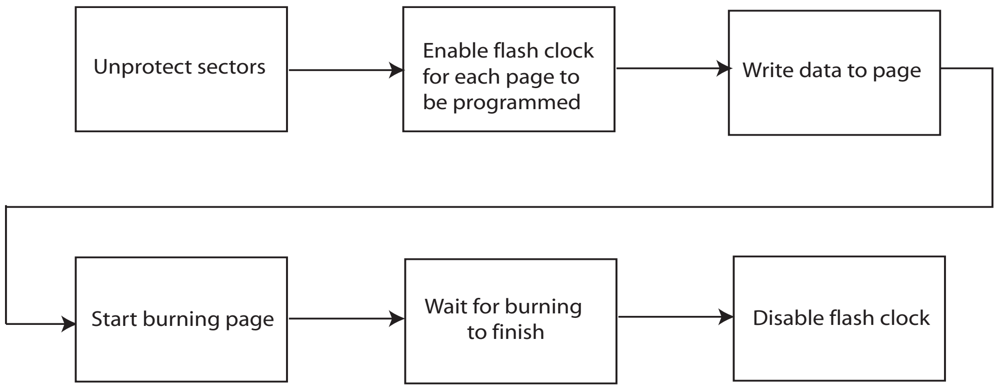

Unprotect sectors
Enable flash clock for each page to be programmed
Write data to page
Start burning page
Wait for burning to finish
Disable flash clock

Fig 120. Flash-memory burn sequence

## 2.4 Flash signature generation

The flash module contains a built-in signature generator. This generator can produce a 128-bit signature (MISR) from a range of the flash memory. A typical usage is to verify the flashed contents against a calculated signature (e.g. during programming). See Section 28–4 for details.

**Remark:** The address range for generating a signature must be aligned on flash-word boundaries.

**Remark:** Like erasing a sector or burning a page, the generation of a signature is also an asynchronous action; i.e. after starting generation the module begins calculating the signature, and during this process any access to the flash results in wait-states (see Section 28–2.1). To serve interrupts or perform other actions this critical code must be present outside flash memory (e.g. internal RAM). The code that initiates the signature generation must also be present outside flash memory.

## 2.5 Flash interrupts

Burn, erase and signature generation (MISR) are asynchronous operations; i.e. after initiating them it takes some time before they complete. During this period access to the flash memory results in wait-states.

Completion of these operations is checked via the interrupt status register (INT_STATUS). This can be done either by polling the corresponding interrupt status or by enabling the generation of an interrupt via the interrupt enable register (INT_SET_ENABLE).

The following interrupt sources are available (see Section 28–3.18):

- END_OF_BURN; indicates the completion of burning a page.
- END_OF_ERASE; indicates the completion of erasing one or more sectors.
- END_OF_MISR; indicates the completion of signature generation.

Generation of an interrupt can be enabled (INT_SET_ENABLE register) or disabled (INT_CLR_ENABLE register) for each of these interrupt sources. The interrupt status is always available even if the corresponding interrupt is disabled. INT_STATUS indicates the raw, unmasked interrupt status.

**Remark:** The interrupt status of an operation should be cleared via the INT_CLR_STATUS register before starting the operation, otherwise the status might indicate completion of a previous operation.

**Remark:** Access to flash memory is blocked during asynchronous operations and results in wait-states. Any interrupt service routine that needs to be serviced during this period must be stored entirely outside the flash memory (e.g. in internal RAM).

**Remark:** To detect the completion of an operation (e.g. erase or burn) it is also possible to poll the interrupt status register. This register indicates the raw interrupt status, i.e. the status is independent of whether an interrupt is enabled or not. In this case the interrupts of the Flash Memory Controller must be disabled (default value after reset).

Polling is the easiest way to detect completion of an operation. This method is also used in the previous examples.

## 2.6 Flash memory index-sector features

The flash memory has a special index sector. This is normally invisible from the address space. By setting the FS_ISS bit in the FCTR register the index sector becomes visible at the flash base address and replaces all regular sectors. The layout Figure 28–121 and burn procedure are similar to those for regular sectors.

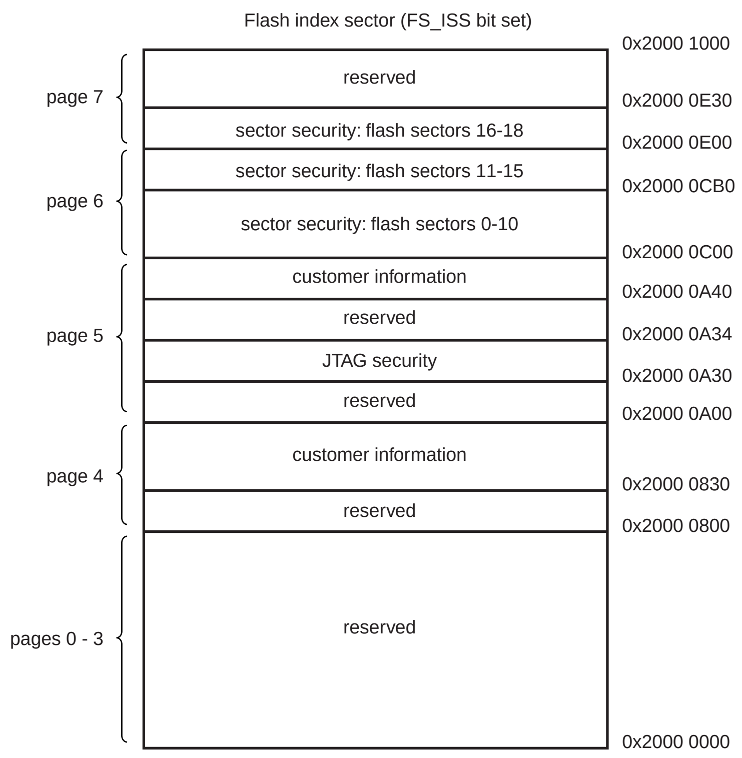


**Fig 121. Index sector layout**

By writing to specific locations in this sector the following features can be enabled:

- JTAG access protection
- Storage of customer information
- Sector security

**Remark:** It is not possible to erase the index sector. As a result the sector is write-only and enabled features cannot be disabled again.

In the following sections these features and the procedures to enable them are described in detail.

**Remark:** As the index sector shares the address space of the regular sectors it is not possible to access it via code in flash. Accessing is only possible via code outside flash memory (e.g. internal RAM).

**Remark:** Take care when writing locations in the index sector. The sector cannot be erased, and using unspecified values or locations might result in a corrupted or malfunctioning device which cannot be recovered.

## 2.6.1 Index sector programming

During index sector programming it is necessary to disable ECC. In the FTCTR register, set bits 29 and 28 to disable ECC. The other bits in FTCTR should not be changed (0). The following code example shows how to disable ECC and unprotect the index sector:

```
#define UNPROTECT 0x0000 0000
```

```
#define PROTECT 0xFFFF FFFF
/* unprotect index sector */
FMC_INX_SECTOR = UNPROTECT;
FCTR = (FS_ISS| FS_LOADREQ | FS_WPB | FS_WEB | FS_WRE | FS_CS);
FTCTR=(FTCTR |(1<<29)|(1<<28)); /* disable error correction! */
... /* program the index sector */
/* re enable ECC */
FTCTR=(FTCTR & 0xCFFFFFFF);
/* protect the index sector */
FMC_INX_SECTOR = PROTECT
```

### 2.6.2 Index sector customer info

The index sector can also be used to program customer-specific information. A part of the index sector on page 4 and 5 (32 flash words) can be programmed at the customer's discretion. The range available for this purpose is shown in Table 28–420:

**Table 420. Customer-specific information**

| Index Sector Page # (FS_ISS bit set) | Customer Info Address Range |
|---|---|
| 4 | 0x2000 0830 to 0x2000 09FF |
| 5 | 0x2000 0A40 to 0x2000 0BFF |

### 2.6.3 JTAG security

Two levels of security can be implemented in the LPC29xx:

1. Soft Security: To enable JTAG security, program 128 bits of the index sector address starting at 0x2000 0A30 with 0x7FFF FFFF. JTAG access is now disabled but can be re-enabled by software. In this case software must enable JTAG during startup before the counter in the security block reaches 0. Otherwise the JTAG remains disabled.
2. Hard Security: JTAG access disabled and cannot be re-enabled.

The JTAG is disabled if the index sector is appropriately programmed. The setting for the JTAG security in the flash index sector is reflected in the CFID register FEAT3 (see Table 7–68). To re-enable JTAG and override this JTAG security setting, it is necessary to have a piece of code in the Flash (in the user program or application) that sets bit 0 in the SEC_DIS register to '1' (see Section 6–3.2) before the Reset Timer times out. **If the counter has expired before the SEC_DIS is set, the JTAG will remain locked**. The counter runs off OSC1M, and the counter register is 2047 bits.

The SEC_STA register (see Table 6–63) can be monitored to determine the current status of JTAG security. The FEAT3 register bit 31 (see Table 7–68) indicates the state of security for the LPC29xx regardless of the override in the SEC_DIS register.

### 2.6.4 Flash memory sector protection

Sector protection is a feature for setting sectors to Read-Only. It is possible to enable this feature for each individual sector. Once it has been enabled it is no longer possible to write (erase/burn) to the sector. This feature can be used, for example, to prevent a boot sector from being replaced.

For every sector in flash memory there is a corresponding flash-word in the index sector that defines whether it is secured or not. Table 28–421 shows the link between index sector flash-words and sectors in flash memory.

**Table 421. Index sector flash-words**[1]

| Flash memory address range | | Index sector page # | Flash memory sector # | Flash-word address (FS_ISS bit set) |
|---|---|---|---|---|
| 0x2000 0000 - | 0x2000 1FFF | 6 | 11 | 0x2000 0CB0 |
| 0x2000 2000 - | 0x2000 3FFF | 6 | 12 | 0x2000 0CC0 |
| 0x2000 4000 - | 0x2000 5FFF | 6 | 13 | 0x2000 0CD0 |
| 0x2000 6000 - | 0x2000 7FFF | 6 | 14 | 0x2000 0CE0 |
| 0x2000 8000 - | 0x2000 9FFF | 6 | 15 | 0x2000 0CF0 |
| 0x2000 A000 - | 0x2000 BFFF | 7 | 16 | 0x2000 0E00 |
| 0x2000 C000 - | 0x2000 DFFF | 7 | 17 | 0x2000 0E10 |
| 0x2000 E000 - | 0x2000 FFFF | 7 | 18 | 0x2000 0E20 |
| 0x2001 0000 - | 0x2001 FFFF | 6 | 0 | 0x2000 0C00 |
| 0x2002 0000 - | 0x2002 FFFF | 6 | 1 | 0x2000 0C10 |
| 0x2003 0000 - | 0x2003 FFFF | 6 | 2 | 0x2000 0C20 |
| 0x2004 0000 - | 0x2004 FFFF | 6 | 3 | 0x2000 0C30 |
| 0x2005 0000 - | 0x2005 FFFF | 6 | 4 | 0x2000 0C40 |
| 0x2006 0000 - | 0x2006 FFFF | 6 | 5 | 0x2000 0C50 |
| 0x2007 0000 - | 0x2007 FFFF | 6 | 6 | 0x2000 0C60 |
| 0x2008 0000 - | 0x2008 FFFF | 6 | 7 | 0x2000 0C70 |
| 0x2009 0000 - | 0x2009 FFFF | 6 | 8 | 0x2000 0C80 |
| 0x200A 0000 - | 0x200A FFFF | 6 | 9 | 0x2000 0C90 |
| 0x200B 0000 - | 0x200B FFFF | 6 | 10 | 0x2000 0CA0 |

[1] See Table 28–419 and Table 28–417 for available flash sectors.

In Table 28–422 decoding of the flash-word is listed:

**Table 422. Sector security values**

| Flash-word value | Description |
|---|---|
| All bits ‘1’ | Corresponding sector is Read/Write (default) |
| All bits ‘0’ | Corresponding sector is Read-Only |

**Remark:** After enabling flash memory security, this feature is not activated until the next reset.

**Remark:** When flash memory security is enabled, it is not possible to disable this feature.

## 2.7 EEPROM functional description

EEPROM is a non-volatile memory mostly used for storing relatively small amounts of data, for example for storing settings.

There are three operations for accessing the memory: reading, writing and erase/program. “Writing” to memory is split up into two separate operations, writing and erase/program. The first operation which will be called “writing” in this document is not really updating the memory, but only updating the temporary data register called the “page register”. The page register needs to be written with minimum 1 byte and maximum 8 bytes before the second operation which is called “erase/program” in this document can be used to actually update the non-volatile memory. Note that the data written to the page register is not “cached”, it can’t be read before it is actually programmed into non-volatile memory.

The 64-byte page register, present in every EEPROM device, is exactly the size of a page in memory.

**Remark:** The EEPROM device contains 256 pages of 64 byte each. The last page must not be accessed. Reading from or writing to the last page will cause undefined behavior of the device.

## 2.8 Addressing

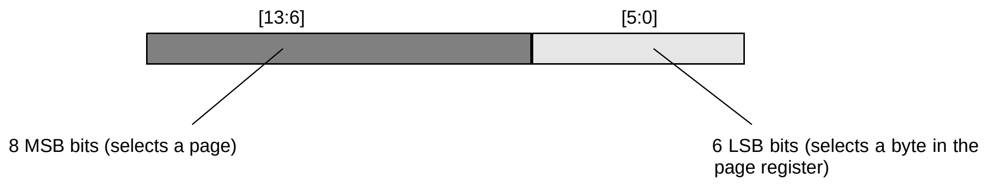

[13:6]
[5:0]
8 MSB bits (selects a page)
6 LSB bits (selects a byte in the page register)

**Fig 122. Address fields**

When doing a write operation the LSB bits need to be used to select a byte in a page register. For a write operation the MSB bits are don’t care. It is possible to write to different page register before starting an erase/program operation on any device.

For an erase/program operation the LSB bits are don’t care. The MSB bits are needed to select the page. During the erase/program operation the other devices can still be accessed.

When doing a read operation all the address fields are needed.

## 2.9 Initialization

**Remark:** The minimum operating voltage for the data EEPROM is $V_{dd}$ = 1.5 V.

To initialize the EEPROM, follow these steps:

1. Apply reset for $\geq$ 100 μs.
2. Program the EECLKDIV register (Table 28–441) to obtain a 375 kHz EEPROM clock.
3. Set waitstates in the EEWSTATE register (Table 28–440).

At power-up the reset should be applied for at least 100 μs. However a normal reset (not at power-up) period is only 40 ns. The longer reset period at power-up relates to the bandgap of the FLASH and EEPROM devices.

After the reset period the EEPROM devices will initialize themselves. Therefore the first 2 bytes (containing trimming information) of the last (protected) page of every EEPROM device will be read. The information that is read will be used as input values of the EEPROM devices, it will be ignored by the controller and is not visible on the AHB/VPB bus.

Before starting any EEPROM operation some registers need to be programmed. First of all the EEPROM devices need a 375 kHz clock for erase/program actions, so the system bus clock needs to be divided to generate an internal clock with this frequency. Register EECLKDIV needs to be programmed to divide the system bus clock. Next to this also the EEWSTATE needs to be programmed with wait state values. After programming these registers operations on the EEPROM devices can be started.

## 2.10 EEPROM operations

An EEPROM device can not be programmed directly. Writing data to it and the actual erase/program of the memory are two separate steps. The page register (64 bytes) will temporarily hold write data. But as soon as this data needs to be read from the EEPROM or data needs to be written to another page, the contents of the page register first needs to programmed into the EEPROM memory.

It is important to know that the EEPROM devices are completely independent from the FLASH module. Therefore EEPROM and FLASH operations can be mixed and performed in parallel without interfering with each other.

The following sections explain the EEPROM operations (read, write and program) in more detail.

### 2.10.1 Writing

The EEPROM controller supports writing of 8-bit, 16-bit or 32-bit elements. Since the EEPROM device doesn’t support 32-bit operations the controller splits the operation into two 16-bit operations.

For doing a write operation first an address needs to be written into the address register and the kind of write operation needs to be selected in the command register. This can be done in any order. After this the data is written to the write data register, which automatically starts the write operation on the EEPROM device.

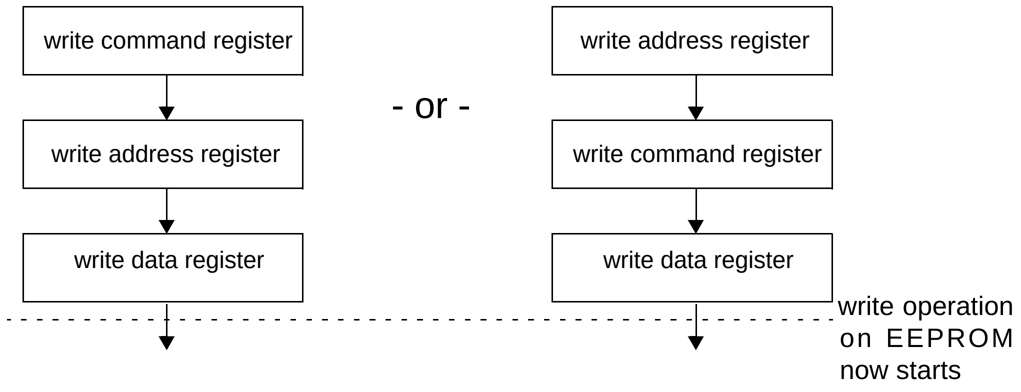


Fig 123. Starting a write operation

A write operation causes an automatic post-increment of the address. This allows consecutive writes to the page register without the need of writing a new address for every write operation. Of course the address register could be written with another address value to write to another location.

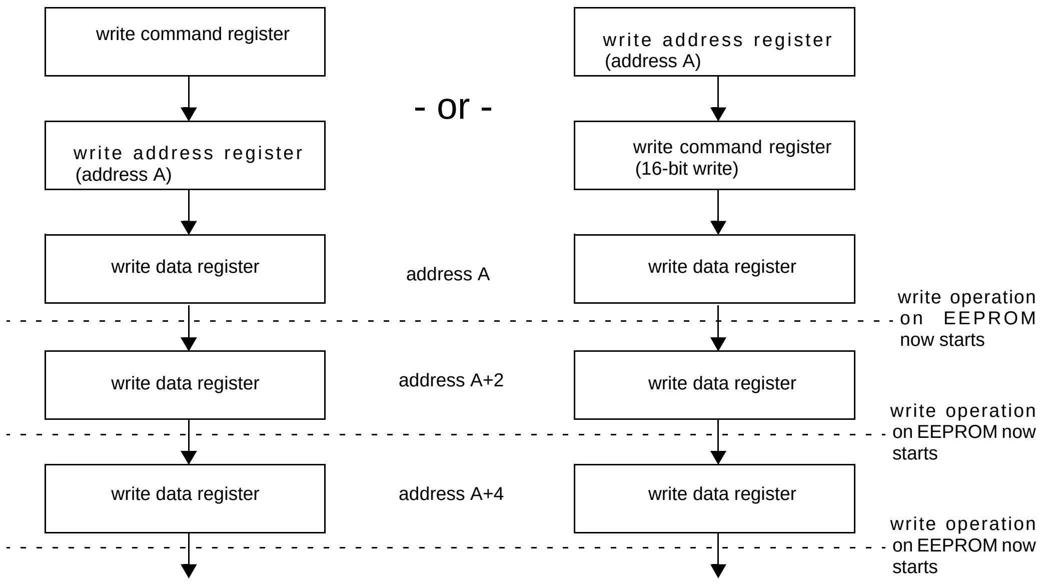

write command register
write address register (address A)
write data register
write data register
write data register
- or -
address A
address A+2
address A+4
write address register (address A)
write command register (16-bit write)
write data register
write data register
write data register
write operation on EEPROM now starts
write operation on EEPROM now starts
write operation on EEPROM now starts

**Fig 124. Write operations with post-incrementing of address**

If the data register is written while a previous EEPROM operation is still pending, the write transfer on the system bus is stalled by de-asserting the ready signal until the previous operation is finished. This can be avoided by polling the interrupt status register to see if an operation is still pending before starting the write operation. Polling is only sensible for systems running at a high frequency (>200 MHz).

Software has to make sure that the following rules are followed:

- Overwriting (writing it two times before an erase/program operation) one of the locations in a 64-byte page register is not allowed, it will cause the loss of the previously written data.
- In case the default address post-incrementing is used the upper boundary of the page register may not be crossed.
- Before reading the just written data the contents of the page register needs to be programmed into non-volatile memory.
- Write operations to a misaligned address will result on an error response on the write transfer to the write data register (for example a 32-bit write operation to an address other than a multiple of 0x4). The operation will not be performed.

## 2.10.2 Programming

When the page register has been written the data has to programmed into non-volatile memory. This is a separate step, writing only the page register will not write the EEPROM memory.

Programming the page into memory takes a long time, therefore the corresponding interrupt can be enabled or the interrupt status bit can be polled to avoid stalling of the system bus.

An erase/program operation starts by providing the MSBs of the address that selects the page in memory of a device. The 6 LSBs are “don’t care”. The operation is started by writing the command register (selecting the erase/program operation).

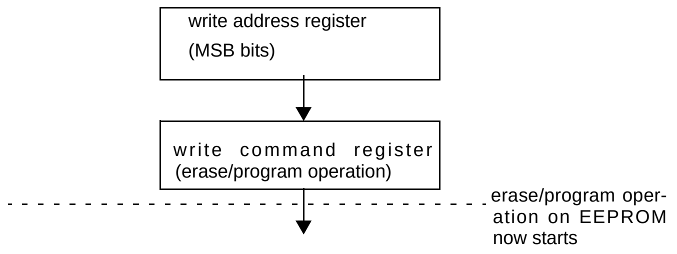

write address register
(MSB bits)
write command register
(erase/program operation)
erase/program operation on EEPROM now starts

Fig 125. Programming a page into memory

### 2.10.3 Reading

The EEPROM controller supports reading of 8-bit, 16-bit or 32-bit elements. Since the EEPROM device doesn’t support 32-bit operations the controller splits the operation into two 16-bit operations.

For doing a read operation first an address needs to be written into the address register. Then the operation needs to be selected in the command register. Writing the command register will automatically start the read operation on the EEPROM device.

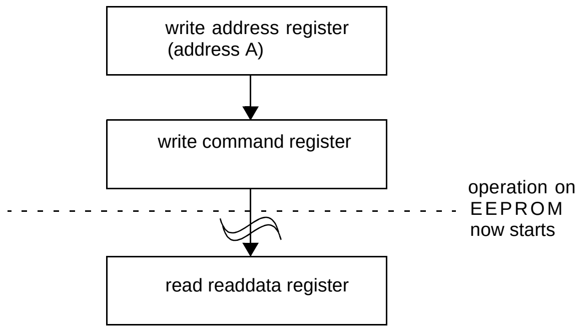


Fig 126. Starting a read operation (read from address A)

If the read data register is read while the read operation is still pending, then the read transfer on the system bus is stalled by de-asserting the ready signal until the previous read operation is finished. This can be avoided by polling the interrupt status register to see if the operation is still pending before reading the read data register.

By default read operations will automatically post-increment the address register. This allows consecutive reads from the EEPROM memory without the need of writing a new address for every read operation. By setting the read data pre-fetch bit in the command register reading from the read data register automatically starts up a read operation from the next (incremented) address location. When doing consecutive reads in this way the first read operation is started as result of writing the command register. The following read operations are started as result of reading the read data register to obtain the result of the previous read operations.

Read operations from a misaligned address will result on an error response on the write transfer to the command register (for example a 32-bit read operation from an address other than a multiple of 0x4). The operation will not be performed.

### 2.10.4 Error responses

The controller can generate the following EEPROM related error responses in the following situations:

- An erase/program operation on the protected page will result in an error response on the write transfer to the command register.
- Writing a read-only register or reading a write-only register will result in an error response.
- A transfer to a non-existing register location will result in an error.

### 2.10.5 EEPROM usage note

The minimum operating voltage for the data EEPROM is $V_{dd}$ = 1.5 V.

## 2.11 EEPROM interrupts

Burn, erase and signature generation (MISR) are asynchronous operations; i.e. after initiating them it takes some time before they complete. During this period access to the flash memory results in wait-states.

Completion of these operations is checked via the interrupt status register (INT_STATUS). This can be done either by polling the corresponding interrupt status or by enabling the generation of an interrupt via the interrupt enable register (INT_SET_ENABLE).

The following interrupt sources are available for EEPROM (see Section 28–3.18):

- END_OF_PROG1; indicates the completion of a program operation.
- END_OF_BIST; indicates the completion of a BIST operation.
- END_OF_RDWR; indicates the completion of a read/write operation.

For each of these interrupt sources generation of an interrupt can be enabled (INT_SET_ENABLE register) or disabled (INT_CLR_ENABLE register). The interrupt status is always available even if the corresponding interrupt is disabled. INT_STATUS indicates the raw, unmasked interrupt status.

**Remark:** The interrupt status of an operation should be cleared via the INT_CLR_STATUS register before starting the operation, otherwise the status might indicate completion of a previous operation.

**Remark:** To detect completion of an operation, it is also possible to poll the interrupt status register. This register indicates the raw interrupt status; i.e. the status is independent of whether an interrupt is enabled or not. In this case the interrupts of the Flash Memory Controller must be disabled (default value after reset).

Polling is the easiest way to detect completion of an operation. This method is also used in the previous examples.

## 3. Register overview

Table 423. Register overview: FMC (base address 0x2020 0000)

| Name | Access | Address offset | Description | Reset Value | Reference |
|---|---|---|---|---|---|
| **Flash registers** | | | | | |
| FCTR | R/W | 0x000 | Flash control register | 0x0005 | Table 28–424 |
| reserved | - | 0x004 | Reserved register; do not modify | - | - |
| FPTR | R/W | 0x008 | Flash program-time register | 0x0000 | Table 28–425 |
| FTCTR | R/W | 0x00C | Flash test control register | - | - |
| FBWST | R/W | 0x010 | Flash bridge wait-state register | 0xC004 | Table 28–427 |
| reserved | - | 0x014 | Reserved register; do not modify | - | - |
| reserved | - | 0x018 | Reserved register; do not modify | - | - |
| FCRA | R/W | 0x01C | Flash clock divider register | 0x000 | Table 28–428 |
| FMSSTART | R/W | 0x020 | Flash Built-In Self Test (BIST) start-address register | 0x0 0000 | Table 28–429 |
| FMSSTOP | R/W | 0x024 | Flash BIST stop-address register | 0x0 0000 | Table 28–430 |
| FMS16 | R | 0x028 | Flash 16-bit signature register | - | Table 28–431 |
| FMSW0 | R | 0x02C | Flash 128-bit signature Word 0 register | - | Table 28–432 |
| FMSW1 | R | 0x030 | Flash 128-bit signature Word 1 register | - | Table 28–433 |
| FMSW2 | R | 0x034 | Flash 128-bit signature Word 2 register | - | Table 28–434 |
| FMSW3 | R | 0x038 | Flash 128-bit signature Word 3 register | - | Table 28–435 |
| **EEPROM registers** | | | | | |
| EECMD | R/W | 0x080 | EEPROM command register | 0x0 | Table 28–436 |
| EEADDR | R/W | 0x084 | EEPROM address register | 0x0 | Table 28–437 |
| EEWDATA | W | 0x088 | EEPROM write data register | n.a. | Table 28–438 |
| EERDATA | R | 0x08C | EEPROM read data register | undef. | Table 28–439 |
| EEWSTATE | R/W | 0x090 | EEPROM wait state register | 0x0 | Table 28–440 |
| EECLKDIV | R/W | 0x094 | EEPROM clock divider register | 0x0 | Table 28–441 |
| EEPWRDWN | R/W | 0x098 | EEPROM power-down/start register | 0x0 | Table 28–442 |
| EEMSSTART | R/W | 0x09C | EEPROM BIST start address register | 0x0 | Table 28–443 |
| EEMSSTOP | R/W | 0x0A0 | EEPROM BIST stop address register | 0x0 | Table 28–444 |
| EEMSSIG | R | 0x0A4 | EEPROM 24-bit BIST signature register | 0x0 | Table 28–445 |
| **Registers shared by flash and EEPROM** | | | | | |
| INT_CLR_ENABLE | W | 0xFD8 | Flash/EEPROM interrupt clear- enable register | - | Table 10–91 |
| INT_SET_ENABLE | W | 0xFDC | Flash/EEPROM interrupt set- enable register | - | Table 10–92 |
| INT_STATUS | R | 0xFE0 | Flash/EEPROM interrupt status register | 0x0 | Table 10–93 |
| INT_ENABLE | R | 0xFE4 | Flash/EEPROM interrupt enable register | 0x0 | Table 10–94 |
| INT_CLR_STATUS | W | 0xFE8 | Flash/EEPROM interrupt clear-status register | - | Table 10–95 |
| INT_SET_STATUS | W | 0xFEC | Flash/EEPROM interrupt set-status register | - | Table 10–96 |

## 3.1 Flash memory control register

The flash memory control register (FCTR) is used to select read modes and to control the programming of flash memory.

Flash memory has data latches to store the data that is to be programmed into it, so that the data-latch contents can be read instead of reading the flash memory contents. Data-latch reading is always done without buffering, with the programmed number of wait-states (WSTs) on every beat of the burst. Data-latch reading can be done both synchronously and asynchronously, and is selected with the FS_RLD bit.

Index-sector reading is always done without buffering, with the programmed number of WSTs on every beat of the burst. Index-sector reading can be done both synchronously and asynchronously and is selected with the FS_ISS bit.

Table 28–424 shows the bit assignment of the FCTR register.

**Table 424. FCTR register bit description (FTCR, address: 0x2020 0000)**
** = reset value*

| Symbol | Bit | Access | Value | Description |
|---|---|---|---|---|
| reserved | 31 to 16 | R | - | Reserved; do not modify. Read as logic 0 |
| FS_LOADREQ | 15 | R/W | | Data load request. |
| | | | 1 | The flash memory is written if FS_WRE has been set; the data load is automatically triggered after the last word was written to the load register. |
| | | | 0* | Automatically cleared; always read as logic 0. |
| FS_CACHECLR | 14 | R/W | | Buffer-line clear. |
| | | | 1 | All bits of the data-transfer register are set. |
| | | | 0* | Reset value. |
| FS_CACHEBYP | 13 | R/W | | Buffering bypass. |
| | | | 1 | Reading from flash memory is without buffering. |
| | | | 0* | Read-buffering is active. |
| FS_PROGREQ | 12 | R/W | | Programming request. |
| | | | 1 | Flash memory programming is requested. |
| | | | 0* | Reset value. |
| FS_RLS | 11 | R/W | | Select sector latches for reading. |
| | | | 1 | The sector latches are read. |
| | | | 0* | The flash memory array is read. |
| FS_PDL | 10 | R/W | | Preset data latches. |
| | | | 1 | All bits in the data latches are set. |
| | | | 0* | Reset value. |
| FS_PD | 9 | R/W | | Power-down. |
| | | | 1 | The flash memory is in power-down. |
| | | | 0* | Reset value. |
| reserved | 8 | R | - | Reserved; do not modify. Read as logic 0 |

**Table 424. FCTR register bit description (FTCR, address: 0x2020 0000)** *…continued*
** = reset value*

| Symbol | Bit | Access | Value | Description |
|---|---|---|---|---|
| FS_WPB | 7 | R/W | | Program and erase protection. |
| | | | 1 | Program and erase enabled. |
| | | | 0* | Program and erase disabled. |
| FS_ISS | 6 | R/W | | Index-sector selection. |
| | | | 1 | The index sector will be read. |
| | | | 0* | The flash memory array will be read. |
| FS_RLD | 5 | R/W | | Read data latches. |
| | | | 1 | The data latches are read for verification of data that is loaded to be programmed. |
| | | | 0* | The flash memory array is read. |
| FS_DCR | 4 | R/W | | DC-read mode. |
| | | | 1 | Asynchronous reading selected. |
| | | | 0* | Synchronous reading selected. |
| reserved | 3 | R | - | Reserved; do not modify. Read as logic 0 |
| FS_WEB | 2 | R/W | | Program and erase enable. |
| | | | 1* | Program and erase disabled. |
| | | | 0 | Program and erase enabled. |
| FS_WRE | 1 | R/W | | Program and erase selection. |
| | | | 1 | Program and data-load selected. |
| | | | 0* | Erase selected. |
| FS_CS | 0 | R/W | | Flash memory chip-select. |
| | | | 1* | The flash memory is active. |
| | | | 0 | The flash memory is in standby. |

In the following, code examples for setting the FTCR register for different flash operations are listed (see Section 28–4 for details):

### Unprotect sector

```
#define UNPROTECT 0x0000 0000
*Sector = UNPROTECT;
FCTR = (FS_LOADREQ | FS_WPB | FS_WEB | FS_WRE | FS_CS);
```

### Erase sector

```
 *Sector = 0;
FCTR = (FS_PROGREQ | FS_WPB | FS_CS);
```

### Burn page

```
 FCTR = (FS_PROGREQ | FS_WPB | FS_WRE | FS_CS);
```

### Preset data latches

```
 FCTR = (FS_ISS | FS_WRE | FS_WEB | FS_CS | FS_PDL);
 FCTR = (FS_ISS | FS_WRE | FS_WEB | FS_CS);
```

## 3.2 Flash memory program-time register

The flash memory program-time register (FPTR) controls the timer for burning and erasing the flash memory. It also allows reading of the remaining burn or erase time.

A built-in timer is used to control the burn time or erase time. The timer is started by writing the burn or erase time to the timer register FPTR.TR, and by enabling it through FPTR.EN_T. During burning or erasing, the timer register counts back to zero, and its current value is returned when reading the FPTR register. This timer register can be used to observe the progress of burning/erasing, and to terminate the burning/erasing.

Erase time ($t_{er}$) to be programmed can be calculated from the following formula:

$$t_{er} = \frac{t_{er(sect)}}{512 \times t_{clk(sys)}}$$

Burn time ($t_{wr}$) to be programmed can be calculated from the following formula:

$$t_{er} = \frac{t_{wr(pg)}}{512 \times t_{clk(sys)}}$$

Table 28–425 shows the bit assignment of the FPTR register.

**Table 425. FPTR register bit description (FPTR, address: 0x2020 0008)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 | EN_T | R/W | | Program-timer enable. |
| | | | 1 | Flash memory program timer enabled. |
| | | | 0* | Flash memory program timer disabled. |
| 14 to 0 | TR[14:0] | R/W | | Program timer; the (remaining) burn and erase time is 512 × TR clock cycles. |
| | | | 0x0000* | Reset value. |

## 3.3 Flash test control register

The FLASH test control register gives access to several pins of the flash memory instance.

**Table 426. FTCTR - flash test control register (FTCTR, address 0x2020 000C)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:30 | - | 0 | - | Reserved; do not modify. Read as logic 0. |
| 29 | R/W | 0 | FS_BYPASS_R | Connected to FLASH memory pin BYPASS_R. |
| 28 | R/W | 0 | FS_BYPASS_W | Connected to FLASH memory pin BYPASS_W. |
| 27:0 | - | 0 | - | Reserved; do not modify. Read as logic 0. |

## 3.4 Flash bridge wait-states register

The flash bridge wait-states register (FBWST) controls the number of wait-states inserted for flash-read transfers. This register also controls the second buffer line for asynchronous reading.

To eliminate the delay associated with synchronizing flash-read data, a predefined number of wait-states must be programmed. These depend on flash-memory response time and system clock period. The minimum wait-states value can be calculated with the following formulas where $t_{acc(clk)}$ = clock access time, $t_{clk(sys)}$ = system clock period and $t_{acc(addr)}$ = address access time (see Ref. 31–1 for further details):

Synchronous reading:

$$WST > \frac{t_{acc(clk)}}{t_{tclk(sys)}} - 1$$

Asynchronous reading:

$$WST > \frac{t_{acc(addr)}}{t_{tclk(sys)}} - 1$$

**Remark:** If the programmed number of wait-states is more than three, flash-data reading cannot be performed at full speed (i.e. with zero wait-states at the AHB bus) if speculative reading is active.

Table 28–427 shows the bit assignment of the FBWST register.

**Table 427. FBWST register bit description (FBWST, address: 0x2020 0010)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 15 | CACHE2EN | R/W | | Dual buffering enable. |
| | | | 1* | Second buffer line is enabled. |
| | | | 0 | Second buffer line is disabled. |
| 14 | SPECALWAYS | R/W | | Speculative reading. |
| | | | 1* | Speculative reading is always performed. |
| | | | 0 | Single speculative reading is performed. |
| 13 to 8 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 7 to 0 | WST[7:0] | R/W | | Number of wait-states. Contains the number of wait-states to be inserted for flash memory reading. The minimum calculated value must be programmed for proper flash memory read-operation. |
| | | | 04h* | Reset value. |

## 3.5 Flash-memory clock divider register

The flash-memory clock divider register (FCRA) controls the clock divider for the flash-memory program-and-erase clock CRA. This clock should be programmed to 66 kHz during burning or erasing.

The CRA clock frequency fed to flash memory is the system clock frequency divided by 3 × (FCRA + 1). The programmed value must result in a CRA clock frequency of 66 kHz ± 20 %.

Table 28–428 shows the bit assignment of the FCRA register.

**Table 428. FCRA register bit description (FCRA, address: 0x2020 001C)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 12 | reserved | R | - | Reserved; do not modify. Read as logic 0 |
| 11 to 0 | FCRA[11:0] | R/W | | Clock divider setting. |
| | | | 000h* | No CRA clock is fed to the flash memory. |

## 3.6 Flash-memory BIST control registers

The flash-memory Built-In Self Test (BIST) control registers control the embedded BIST signature generation. This is implemented via the BIST start-address register FMSSTART and the stop-address register FMSSTOP.

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 BIST start-address register and the stop address to the BIST stop-address register. The BIST start and stop addresses must be flash memory word-aligned and can be derived from the AHB byte addresses through division by 16. Signature generation is started by setting the BIST start-bit in the BIST stop-address register. Setting the BIST start-bit is typically combined with defining the signature stop address.

Flash access is blocked during the BIST signature calculation. The duration of the flash BIST time is $t_{BIST} = (t_{fl(BIST)} + 3 \times t_{clk(sys)}) \times (FMSSTOP - FMSSTART + 1)$

Table 28–429 and Table 28–430 show the bit assignment of the FMSSTART and FMSSTOP registers respectively.

**Table 429. FMSSTART register bit description (FMSSTART, address: 0x2020 0020)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 17 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0. |
| 16 to 0 | FMSSTART[16:0] | R/W | 0 0000h* | BIST start address (corresponds to AHB byte address [20:4]). |

**Table 430. FMSSTOP register bit description (FMSSTOP, address: 0x2020 0024)**
** = reset value*

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 18 | reserved | R | - | Reserved; do not modify. Read as logic 0, write as logic 0. |
| 17 | MISR_START | R/W | | BIST start. |
| | | | 1 | BIST signature generation is initiated. |
| | | | 0* | Reset value. |
| 16 to 0 | FMSSTOP[16:0] | R/W | | BIST stop address divided by 16 (corresponds to AHB byte address [20:4]). |
| | | | 0 0000h* | Reset value. |

## 3.7 Flash-memory BIST signature registers

The flash-memory BIST signature registers return signatures as produced by the embedded signature generator. There is a 128-bit signature reflected by the four registers FMSW0, FMSW1, FMSW2 and FMSW3. The 16-bit signature is reflected by the FMS16 register.

The signature generated by the flash memory is used to verify the flash memory contents. The generated signature can be compared with an expected signature and thus makes unnecessary the more time- and code-consuming procedure of reading back the entire contents.

Table 28–431 to Table 28–435 show bit assignment of the FMS16, FMSW0 and FMSW1, FMSW2, FMSW3 registers respectively.

**Table 431. FMS16 register bit description (FMS16, address: 0x2020 0028)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 16 | - | R | - | Flash BIST 128-bit signature (bits 31 to 0). |
| 15 to 0 | FMS16 | R | - | Flash BIST 16-bit signature. |

**Table 432. FMSW0 register bit description (FMSW0, address: 0x2020 002C)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | FMSW0[31:0] | R | - | Flash BIST 128-bit signature (bits 31 to 0). |

**Table 433. FMSW1 register bit description (FMSW1, address: 0x2020 0030)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | FMSW1[63:32] | R | - | Flash BIST 128-bit signature (bits 63 to 32). |

**Table 434. FMSW2 register bit description (FMSW2, address: 0x2020 0034)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | FMSW2[95:64] | R | - | Flash BIST 128-bit signature (bits 95 to 64). |

**Table 435. FMSW3 register bit description (FMSW3, address: 0x2020 0038)**

| Bit | Symbol | Access | Value | Description |
|---|---|---|---|---|
| 31 to 0 | FMSW3[127:96] | R | - | Flash BIST 128-bit signature (bits 127 to 96). |

## 3.8 EEPROM command register

The EEPROM command register is used to select and start a read, write or erase/program operation. Read and erase/program operations are started on the EEPROM device as a side-effect of writing to this register. (Write operations are started as a side-effect of writing to the write data register).

Table 436. EEPROM command register bit description (EECMD - address 0x2020 0080)

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:5 | - | 0x0 | reserved | |
| 4 | R/W | 0x0 | PAR_ACCESS | Parallel access<br>0: no parallel access<br>1: the selected write or erase/program operation (no read!) is performed in parallel on all devices. For write operations this means that the same write data is written to all devices. |
| 3 | R/W | 0x0 | RDPREFETCH | Read data pre-fetch bit<br>0: do not pre-fetch next read data as result of reading from the read data register<br>1: pre-fetch read data as result of reading from the read data register<br>When this bit is set multiple consecutive data elements can be read without the need of programming new address values in the address register. The default address post-increment and the automatic read data pre-fetch allow only reading from the read data register to be done to read the data. |
| 2:0 | R/W | 0x0 | CMD | Command<br>000: 8-bit read<br>001: 16-bit read<br>010: 32-bit read<br>011: 8-bit write<br>100: 16-bit write<br>101: 32-bit write<br>110: erase/program page<br>111: reserved |

## 3.9 EEPROM address register

The EEPROM address register is used to program the address for read, write or erase/program operations. The width of the address field depends on the number of EEPROM devices which is selectable by a configuration parameter. For a detailed description of the address bits see Figure 28–122. Addressing is discussed in Section 28–2.8.

Table 437. EEPROM address register bit description (EEADDR - address 0x2020 0084)

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:14 | - | 0x0 | reserved | |
| 13:0 | R/W | 0x0 | ADDR | Address<br>Address field with:<br>1 EEPROM device => 8 MSB, 0 CS, and 6 LSB bits |

## 3.10 EEPROM write data register

The EEPROM write data register is used to write data into the page register (write operations).

Writing this register will start the write operation as side-effect. If the post-increment bit in the command register is set, consecutive writes to this register can be done to write a burst of data. The address will be incremented automatically according to the datasize of the write operation.

If data is written to this register while a previous operation (read, write or an erase/program on the same device) is still pending, the write command on the AHB is stalled by de-asserting the ready signal until the previous operation is finished. To avoid stalling of the system bus the interrupt status register can be used for polling the status of pending operations.

**Table 438. EEPROM write data register bit description (EEWDATA - address 0x2020 0088)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:0 | W | - | WDATA | Write data<br>In case of:<br>8-bit write operations: bits [7:0] must contain valid write data.<br>16-bit write operations: bits [15:0] must contain valid write data<br>32-bit write operations: bits [31:0] must contain valid write data |

## 3.11 EEPROM read data register

The EEPROM read data register is used to read data from memory.

If the post-increment bit in the command register is set, reading this register will start the next read operation (from the incremented address location) as side-effect. In this case consecutive reads from this register can be done to read a burst of data. The address will be incremented automatically according to the datasize of the read operation.

If data is read from this register while the read operation is still pending the read command on the AHB is stalled by de-asserting the ready signal until the pending operation is finished. To avoid stalling of the system bus the interrupt status register can be used for polling the status of pending operations.

**Table 439. EEPROM read data register bit description (EERDATA - address 0x2020 008C)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:0 | R | - | RDATA | Read data<br>In case of:<br>8-bit read operations: bits [7:0] contain read data, others are zero<br>16-bit read operations: bits [15:0] contain read data, others are zero<br>32-bit read operations: bits [31:0] contain read data |

## 3.12 EEPROM wait state register

The EEPROM controller has no awareness of absolute time, while for EEPROM operations several minimum absolute timing constraints have to be met. Therefore it can only derive time from its clock by frequency division. The user must program the wait state fields to appropriate values in this wait state register. These fields are -1 encoded so programming zero will result in a one cycle wait state.

**Table 440. EEPROM wait state register bit description (EEESTATE - address 0x2020 0090)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:24 | - | 0x0 | - | Reserved. |
| 23:16 | R/W | 0x0 | PHASE1 | Wait states 1 (minus 1 encoded)<br>The number of system clock periods to meet a duration equal to the maximum setup delay time found in the read, write or erase/program operation on an EEPROM device. See the EEPROM device specification.<br><br>At the moment of writing this documentation the duration was 35 ns. |
| 15:8 | R/W | 0x0 | PHASE2 | Wait states 2 (minus 1 encoded)<br>The number of system clock periods to meet a duration equal to the maximum of the following delay times (see the EEPROM device specification and Figure 28–127):<br>- read operations' max. propagation time PRECH to DO ($t_{p\_prech\_do}$)<br>- write operation's min. high time WE_N ($t_{hw\_we\_n}$)<br>- erase/program operation's min. hold time PE_N to EPP ($t_{h\_pe\_n\_epp}$)<br><br>At the moment of writing this documentation the duration was 55 ns. |
| 7:0 | R/W | 0x0 | PHASE3 | Wait states 3 (minus 1 encoded)<br>The number of system clock periods to meet a duration equal to the maximum of the following delay times (see the EEPROM device specification and Figure 28–127):<br>- write operations' min. hold time input to WE_N ($t_{h\_i\_we\_n}$)<br>- erase/program operations' min. hold time BE_N to WE_N ($t_{h\_be\_n\_we\_n}$)<br>- erase/program operations' min. hold time input to EPP ($t_{h\_i\_epp}$)<br><br>At the moment of writing this documentation the duration was 15 ns. |

The register contains three fields, each representing a minimum duration of a phase of a EEPROM operation. The fields have to be programmed such that:

$$(waitstates + 1) \times Tclk \geq duration$$

Several (almost identical) delays of the different EEPROM operations have been put together in these three wait state fields. This has been done to simplify the software interface. Giving the opportunity to program every single delay separately might improve performance a bit, but this is not enough to justify the more complex software interface. Since programming these fields sets only "common" delays of the operations, re-programming is not necessary when switching between the different operations.

NOTE: the wait states in the register are minus 1 encoded.

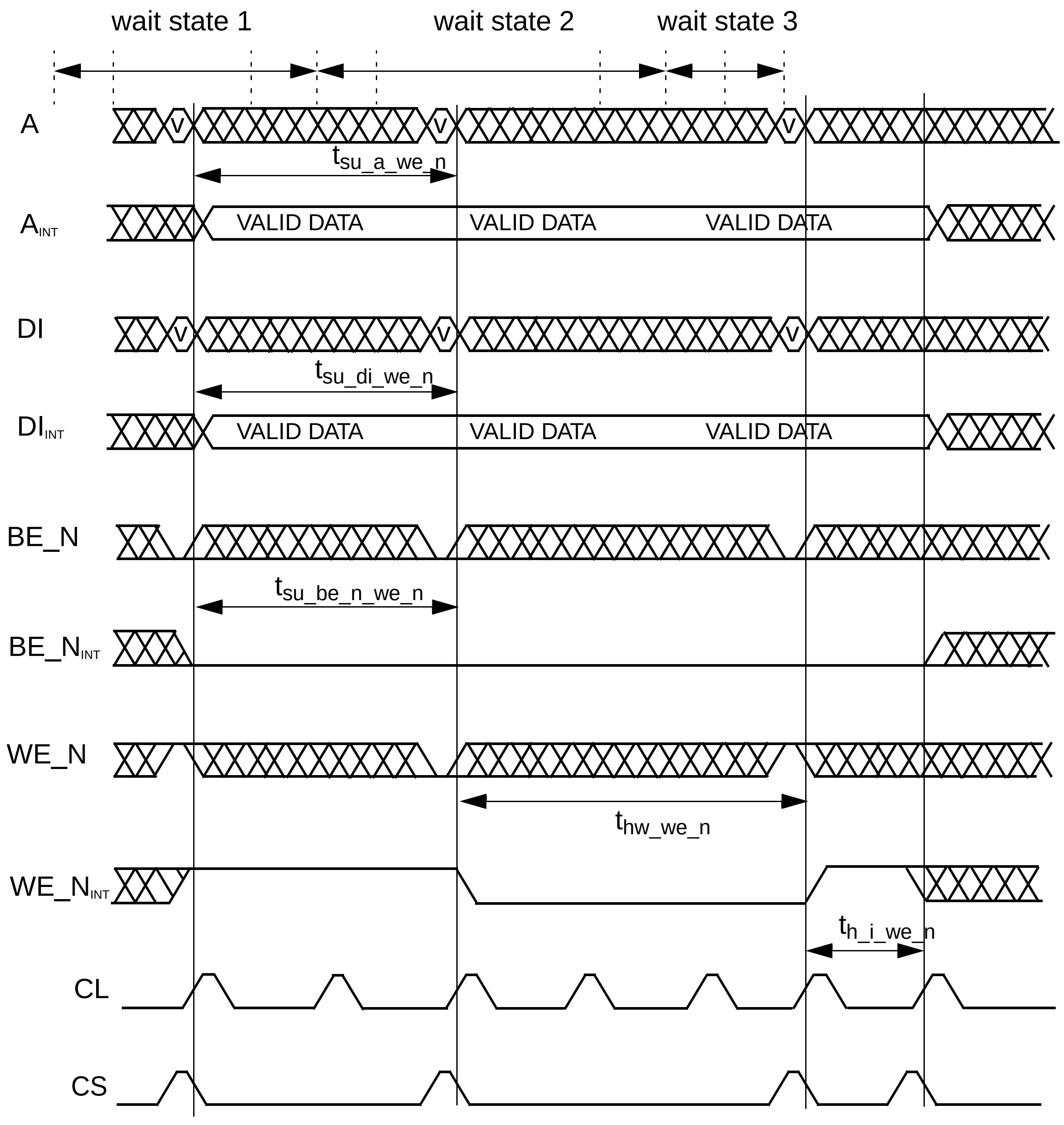


**Fig 127. Wait states in a write operation**

## 3.13 EEPROM clock divider register

The EEPROM device(s) require(s) a 375 kHz clock. This clock is generated by dividing the system bus clock. The clock divider register contains the division factor.

If the division factor is 0 the clock will be IDLE to save power.

$$\frac{Fclk}{divisionfactor + 1} \approx 375kH \pm 6.67\%$$

Table 441. EEPROM clock divider register bit description (EECLKDIV - address 0x2020 0094)

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:16 | - | 0x0 | - | Reserved |
| 15:0 | R/W | 0x0 | CLKDIV | Division factor (minus 1 encoded) |

## 3.14 EEPROM power-down register

The EEPROM power-down register can be used to put the EEPROM device in Power-down mode.

The device may not be put in Power-down mode during a pending EEPROM operation. After clearing this bit any EEPROM operation has to be suspended for 100 μs due to the ramp-up time of the bandgap of the EEPROM device.

Table 442. EEPROM power down/DCM register bit description (EEPWRDWN - address 0x2020 0098)

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:1 | - | 0x0 | - | Reserved |
| 0 | R/W | 0x0 | PWRDWN | Power-down mode bit<br>0: not in Power-down mode<br>1: power down mode (this will put the EEPROM device in Power-down mode) |

## 3.15 EEPROM BIST start address register

The EEPROM BIST start address register is used to program the start address for the BIST. During BIST the EEPROM devices are accessed with 16-bit read operations so the LSB of the address is fixed zero.

Table 443. EEPROM BIST start address register bit description (EEMSSTART - address 0x2020 009C)

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:x+1 | - | 0x0 | reserved | |
| x:0 | R/W | 0x0 | STARTA | BIST start address:<br>Bit 0 is fixed zero since only even addresses are allowed.<br>The width of this field depends on the number of EEPROM devices:<br>1 EEPROM device => x = 13 |

## 3.16 EEPROM BIST stop address register

The EEPROM BIST stop address register is used to program the stop address for the BIST and also to start the BIST. During BIST the EEPROM devices are accessed with 16-bit read operations so the LSB of the address is fixed zero.

**Table 444. EEPROM BIST stop address register bit description (EEMSSTOP - address 0x2020 00A0)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31 | R/W | 0x0 | STRTBIST | BIST start bit<br>Setting this bit will start the BIST. This bit is self-clearing. |
| 30 | R/W | 0x0 | DEVSEL | BIST device select bit<br>0: the BIST signature is generated over the total memory space. Singe pages are interleaved over the EEPROM devices when multiple devices are used, the signature is generated over memory of multiple devices.<br>1: the BIST signature is generated only over a memory range located on a single EEPROM device. Therefore the internal address generation is done such that the address' CS bits are kept stable to select only the same device. The address' MSB and LSB bits are used to step through the memory range specified by the start and stop address fields.<br>Note: if this bit is set the start and stop address fields must be programmed such that they both address the same EEPROM device. Therefore the address' CS bits in both the start and stop address must be the same. |
| 29:x+1 | - | 0x0 | reserved | |
| x:0 | R/W | 0x0 | STOPA | BIST stop address:<br>Bit 0 is fixed zero since only even addresses are allowed.<br>The width of this field depends on the number of EEPROM devices:<br>1 EEPROM device => x = 13 (8 MSB, 0 CS and 6 LSB bits) |

## 3.17 EEPROM signature register

The EEPROM BIST signature register returns the signatures as produced by the embedded signature generators.

**Table 445. EEPROM BIST signature register bit description (EEMSSIG - address 0x2020 00A4)**

| Bits | Access | Reset value | Field name | Description |
|---|---|---|---|---|
| 31:16 | R | 0x0 | PARITY_SIG | BIST 16-bit signature calculated from only the parity bits of the data bytes |
| 15:0 | R | 0x0 | DATA_SIG | BIST 16-bit signature calculated from only the data bytes |

## 3.18 FMC interrupt bit description

Table 28–446 gives the interrupts for the FMC. The first column gives the bit number in the interrupt registers. For a general explanation of the interrupt concept and a description of the interrupt registers see Section 10–6.1.

**Table 446. FMC interrupt sources**

| Register bit | Interrupt source | Description |
|---|---|---|
| 31 to 29 | - | Unused |
| 28 | END_OF_PROG | Program operation has finished for EEPROM device 1 |
| 27 | END_OF_BIST | BIST operation has finished (EEPROM) |
| 26 | END_OF_RDWR | Read/write operation has finished (EEPROM) |
| 25 to 3 | - | Unused |

Table 446. FMC interrupt sources

| Register bit | Interrupt source | Description |
|---|---|---|
| 2 | END_OF_MISR | BIST signature generation has finished (flash) |
| 1 | END_OF_BURN | Page burning has finished (flash) |
| 0 | END_OF_ERASE | Erasing of one or more sectors has finished (flash) |

## 4. Flash and EEPROM programming details

### 4.1 AHB programming

Programming an embedded flash memory is more complex than just writing data to the appropriate address, like e.g. for embedded SRAM.

A flash memory is organized in sectors that must be erased before data can be written into them. A flash memory also has sector protection. Figure 28–128 shows the programming flow chart. The part on the dark background is automatically done by the flash hardware.

Flash programming contains the following elements.

- unprotecting
- erasing
- presetting data latches
- writing
- loading
- burning
- protecting

In the remainder of this chapter these are described in more detail.

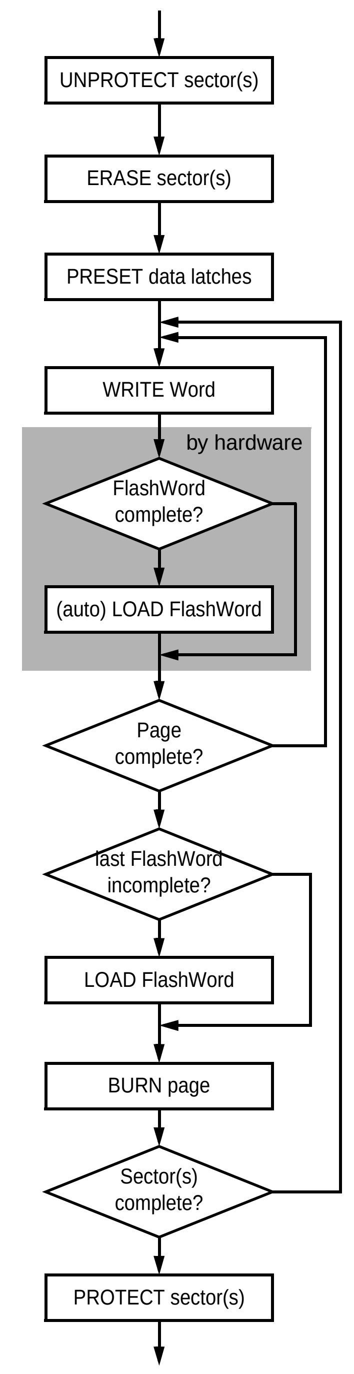


**Fig 128. Flash (AHB/APB) programming**

### 4.1.1 (Un)protecting sectors

A sector gets unprotected by writing an even value to its base address, followed by writing the (un)protect trigger value to the FCTR register.

**Table 447. Un(protect) trigger**

| | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| (un)protect | 1 | 0 | 1 | 1 | 1 | 1 |

A sector gets protected by writing an odd value to its base address, followed by the same trigger as for unprotecting.

Although the flash module does not need the CRA clock to select a sector for (un)protecting, the CRA clock must already be enabled to ensure an active APB clock.

### 4.1.2 Erasing a single sector

Before the erasing, the erase time must be written to the timer register FPTR.TR, and the timer must be enabled through FPTR.EN_T. During erasing, the timer register counts back to zero. Therefore, the timer register must be rewritten before every erase cycle.

The programmed erase time must obey:

$$(512 \cdot \text{FPTR.TR} + 2) \cdot t_{clk} \geq \text{tlw\_web\_erase}$$

which is true for:

$$\text{FPTR.TR} = \frac{\text{tlw\_web\_erase}}{512 \cdot t_{clk}}$$

A single sector gets erased by writing any value to an address within that sector, followed by writing the erase trigger value to the FCTR register.

**Table 448. Single sector erase trigger**

| | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| erase | 0 | 1 | 1 | 0 | 0 | 1 |

Only unprotected sectors can be erased.

For erasing, the flash module needs a 66 kHz CRA clock being active. This clock is derived from the APB clock, dividing it by a factor programmed in FCRA.FCRA. A value of zero inactivates the CRA clock.

### 4.1.3 Erasing multiple sectors

Erasing multiple sectors can be done with only one (time consuming) erase cycle. First all sectors except the last are selected for erasure. Then the last sector is erased using the single sector erase procedure.

A sector gets selected for erasure by writing any value to an address within that sector, followed by writing the select for erase trigger value to the FCTR register.

**Table 449. Select for erase trigger**

| | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| select for erasure | 1 | 0 | 1 | 1 | 0 | 1 |

Although the flash module does not need the CRA clock to select a sector for erasure, the CRA clock must already be enabled to ensure an active APB clock.

### 4.1.4 Presetting data latches

When only a part of a page has to be programmed, the data latches for the rest of the page must be preset to logical 1’s. This can be done with a single control by setting and clearing the FCTL.FS_PDL bit. Only the data latches in the flash (1 page wide) are preset with this action, the data latches in the controller (1 FlashWord wide) are not affected.

### 4.1.5 Writing and loading

Writing to flash is accomplished by writing a Word through the AHB data port to the data inputs of the flash module. Every beat takes 2 clock cycles (1 wait state) and results in a partial update of the data input of the flash module.

Writing is done per Word. Byte or halfword writing is not possible. However, because writing logical 1’s leaves the flash contents unchanged, it is possible to do byte writing by encapsulating this byte in a Word of logical 1’s. This encapsulation must be done by the AHB master that initiates the transfer. The flash itself does not offer this feature.

Note that multiple partial writing is best done with ECC bypassed, to avoid corruption of the ECC bits. ECC must be enabled again on the last write of a sequence of partial writes.

Every 4th write, a FlashWord (=4 Words) is loaded automatically into the data latches of the flash module. Loading is done per FlashWord.

Loading is done automatically after writing to address 0x0C. This requires that values are already written to addresses 0x00.. 0x08. An AHB incremental burst starting on a FlashWord aligned address meets this requirement. Automatic loading is enabled by writing a pattern to the FCTR register.

**Table 450. Automatic load trigger**

| | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| enable automatic loading | 0 | 0 | 0 | 1 | 1 | 1 |

Loading can also be done manually by writing a ‘1’ to FCTR.FS_LOADREQ. In that case, whatever the content of the controller’s data registers is will be transferred to the flash. Manual loading is started by writing a trigger value to the FCTR register.

**Table 451. manual load trigger**

| trigger for | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| manual load | 1 | 0 | 0 | 1 | 1 | 1 |

### 4.1.6 Burning

Burning is the data transfer from the data latches of the flash module to the flash array. Burning is done per page.

Before the burning, the burning time must be written to the timer register FPTR.TR, and the timer must be enabled through FPTR.EN_T. During burning, the timer register counts back to zero. Therefore, the timer register must be rewritten before every burning cycle.

The programmed burning time must obey:

$$(512 \cdot \mathrm{FPTR.TR} + 2) \cdot t_{clk} \geq \mathrm{tlw\_web\_write}$$

which is true for:

$$\mathrm{FPTR.TR} = \frac{\mathrm{tlw\_web\_write}}{512 \cdot t_{clk}}$$

The burning is started by writing a trigger value to the FCTR register.

**Table 452. Burn trigger value**

| | LOADREQ | PROGREQ | WPB | WEB | WRE | CS |
|---|---|---|---|---|---|---|
| burn | 0 | 1 | 1 | 0 | 1 | 1 |

The page address that is offered to the flash module during burning is the page address of the most recent data port write transfer.

Only pages within unprotected sectors can be programmed.

For burning, the flash module needs a 66 kHz CRA clock being active. This clock is derived from the VPB clock, dividing it by a factor programmed in FCRA.FCRA. A value of zero inactivates the CRA clock.

### 4.1.7 Index sector programming

The index sector is (un)protected using the same procedure as normal sector (un)protecting (see Section 28–4.1.1), except that the ISS bit is also set as FCTR trigger. The erasable part of the index sector is erased using the same procedure as normal sector erasing (see Section 28–4.1.3), except that the ISS bit is also set as FCTR trigger.

Writable pages in the index sector are burned using the same procedure as normal burning (see Section 28–4.1.6), except that the ISS bit is also set as FCTR trigger. Data is loaded using the normal writing and loading procedure (see Section 28–4.1.5).

## 4.2 Algorithm for signature generation

To verify the flash contents, a signature can be generated by the flash module, which in turn must be compared with an expected signature. The alternative would be reading back all contents, which would be much more time and code consuming.

#### 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. The asynchronous ready output RY of the memory is synchronized and then used to trigger the reading of the next address. The wait state register FBWST is ignored. A safe estimation for the duration of the signature generation is:

$$\text{MISR duration} = \text{int}\left(\frac{\text{tp\_cl\_ry\_rr}}{\text{clock period}} + 3\right) \times (\text{FMSSTOP - FMSSTART} + 1)$$

When signature generation is triggered via AHB or VPB, the duration is expressed in ahb_clk cycles. Polling the INT_STATUS.END_OF_MISR bit can also be used to check the completion of the signature generation.

When signature generation is triggered via JTAG, the duration is expressed in tck cycles. Polling the INT_STATUS.END_OF_MISR bit is not possible.

After signature generation, a 16 bits signature can be read from the FMS16 register, and a 128 bits signature can be read from the FMSW0 .. FMSW3 registers.

### Content verification

The signatures as they are read from the FMS16 and FMSW0 .. FMSW3 registers must be equal to reference signatures. The algorithms to derive the reference signatures are given in Figure 28–129 and Figure 28–130.

The 128 bit signature is derived from the corrected 128 bits of the FlashWord.

```
sign = 0
FOR address = FMSTART.FMSTART TO FMSTOP.FMSTOP
{
    FOR i = 0 TO 7
        nextSign[i] = f_Q[address][i+128] XOR sign[i+1]
    nextSign[8] = ms_err XOR sign[9]
    nextSIgn[9] = ms_check1 XOR sign[10]
    nextSign[10] = ms_check0 XOR sign[11]
    FOR i = 11 TO 14
        nextSign[i] = sign[i+1]
    nextSign[15] = sign[0] XOR sign[4] XOR sign[13] XOR sign[15]
    sign = nextSign
}
signature16 = sign
```

**Fig 129. Algorithm for generating a 16 bits signature**

```
sign = 0

FOR address = FMSTART.FMSTART TO FMSTOP.FMSTOP
{
    FOR i = 0 TO 126
        nextSign[i] = f_Q[address][i] XOR sign[i+1]

    nextSign[127] = f_Q[address][127] XOR sign[0] XOR sign[2] XOR sign[27] XOR
                        sign[29]

    sign = nextSign
}

signature128 = sign
```

**Fig 130. Algorithm for generating a 128 bits signature**

# UM10316

# Chapter 29: LPC29xx General Purpose DMA (GPDMA) controller

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Introduction

The DMA controller allows peripheral-to memory, memory-to-peripheral, peripheral-to-peripheral, and memory-to-memory transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bi-directional port requires one stream for transmit and one for receives. The source and destination areas can each be either a memory region or a peripheral.

## 3. Features

- Eight DMA channels. Each channel can support an unidirectional transfer.
- 16 DMA request lines.
- Single DMA and burst DMA request signals. Each peripheral connected to the DMA Controller can assert either a burst DMA request or a single DMA request. The DMA burst size is set by programming the DMA Controller.
- Memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral transfers are supported.
- Scatter or gather DMA is supported through the use of linked lists. This means that the source and destination areas do not have to occupy contiguous areas of memory.
- Hardware DMA channel priority.
- AHB slave DMA programming interface. The DMA Controller is programmed by writing to the DMA control registers over the AHB slave interface.
- Two AHB bus masters for transferring data. These interfaces transfer data when a DMA request goes active. Either master can be selected for source or destination on each DMA channel.
- 32-bit AHB master bus width.
- Incrementing or non-incrementing addressing for source and destination.
- Programmable DMA burst size. The DMA burst size can be programmed to more efficiently transfer data.
- Internal four-word FIFO per channel.
- Supports 8, 16, and 32-bit wide transactions.
- Big-endian and little-endian support. The DMA Controller defaults to little-endian mode on reset.
- An interrupt to the processor can be generated on a DMA completion or when a DMA error has occurred.

- Raw interrupt status. The DMA error and DMA count raw interrupt status can be read prior to masking.

## 4. Functional description

This section describes the major functional blocks of the DMA Controller.

### 4.1 DMA controller functional description

The DMA Controller enables peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral, and memory-to-memory transactions. Each DMA stream provides unidirectional serial DMA transfers for a single source and destination. For example, a bidirectional port requires one stream for transmit and one for receive. The source and destination areas can each be either a memory region or a peripheral, and can be accessed through the AHB master. Figure 29–131 shows a block diagram of the DMA Controller.

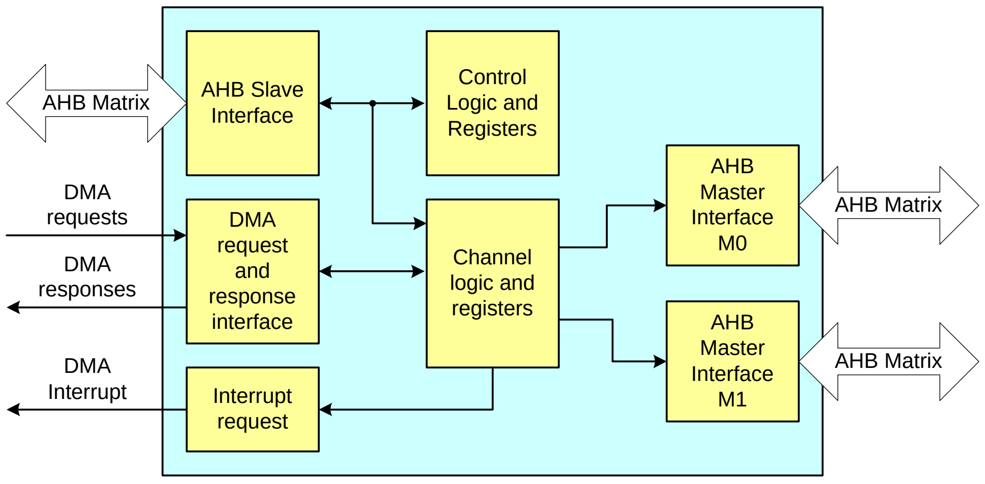


**Fig 131. DMA controller block diagram**

The functions of the DMA Controller are described in the following sections.

#### 4.1.1 AHB slave interface

All transactions to DMA Controller registers on the AHB slave interface are 32 bits wide. Eight bit and 16-bit accesses are not supported and will result in an exception.

#### 4.1.2 Control logic and register bank

The register block stores data written or to be read across the AHB interface.

#### 4.1.3 DMA request and response interface

See DMA Interface description for information on the DMA request and response interface.

### 4.1.4 Channel logic and channel register bank

The channel logic and channel register bank contains registers and logic required for each DMA channel.

### 4.1.5 Interrupt request

The interrupt request generates the interrupt to the ARM processor.

### 4.1.6 AHB master interface

The DMA Controller contains two AHB master interfaces. Each AHB master is capable of dealing with all types of AHB transactions, including:

- Split, retry, and error responses from slaves. If a peripheral performs a split or retry, the DMA Controller stalls and waits until the transaction can complete.
- Locked transfers for source and destination of each stream.
- Setting of protection bits for transfers on each stream.

#### 4.1.6.1 Bus and transfer widths

The physical width of the AHB bus is 32 bits. Source and destination transfers can be of differing widths and can be the same width or narrower than the physical bus width. The DMA Controller packs or unpacks data as appropriate.

#### 4.1.6.2 Endian behavior

The DMA Controller can cope with both little-endian and big-endian addressing. Software can set the endianness of each AHB master individually.

Internally the DMA Controller treats all data as a stream of bytes instead of 16-bit or 32-bit quantities. This means that when performing mixed-endian activity, where the endianness of the source and destination are different, byte swapping of the data within the 32-bit data bus is observed.

Note: If byte swapping is not required, then use of different endianness between the source and destination addresses must be avoided. Table 29–453 shows endian behavior for different source and destination combinations.

**Table 453. Endian behavior**

| Source endian | Destination endian | Source width | Destination width | Source transfer no/byte lane | Source data | Destination transfer no/byte lane | Destination data |
|---|---|---|---|---|---|---|---|
| Little | Little | 8 | 8 | 1/[7:0] | 21 | 1/[7:0] | 21212121 |
| | | | | 2/[15:8] | 43 | 2/[15:8] | 43434343 |
| | | | | 3/[23:16] | 65 | 3/[23:16] | 65656565 |
| | | | | 4/[31:24] | 87 | 4/[31:24] | 87878787 |
| Little | Little | 8 | 16 | 1/[7:0] | 21 | 1/[15:0] | 43214321 |
| | | | | 2/[15:8] | 43 | 2/[31:16] | 87658765 |
| | | | | 3/[23:16] | 65 | | |
| | | | | 4/[31:24] | 87 | | |
| Little | Little | 8 | 32 | 1/[7:0] | 21 | 1/[31:0] | 87654321 |
| | | | | 2/[15:8] | 43 | | |
| | | | | 3/[23:16] | 65 | | |
| | | | | 4/[31:24] | 87 | | |
| Little | Little | 16 | 8 | 1/[7:0] | 21 | 1/[7:0] | 21212121 |
| | | | | 1/[15:8] | 43 | 2/[15:8] | 43434343 |
| | | | | 2/[23:16] | 65 | 3/[23:16] | 65656565 |
| | | | | 2/[31:24] | 87 | 4/[31:24] | 87878787 |
| Little | Little | 16 | 16 | 1/[7:0] | 21 | 1/[15:0] | 43214321 |
| | | | | 1/[15:8] | 43 | 2/[31:16] | 87658765 |
| | | | | 2/[23:16] | 65 | | |
| | | | | 2/[31:24] | 87 | | |
| Little | Little | 16 | 32 | 1/[7:0] | 21 | 1/[31:0] | 87654321 |
| | | | | 1/[15:8] | 43 | | |
| | | | | 2/[23:16] | 65 | | |
| | | | | 2/[31:24] | 87 | | |
| Little | Little | 32 | 8 | 1/[7:0] | 21 | 1/[7:0] | 21212121 |
| | | | | 1/[15:8] | 43 | 2/[15:8] | 43434343 |
| | | | | 1/[23:16] | 65 | 3/[23:16] | 65656565 |
| | | | | 1/[31:24] | 87 | 4/[31:24] | 87878787 |
| Little | Little | 32 | 16 | 1/[7:0] | 21 | 1/[15:0] | 43214321 |
| | | | | 1/[15:8] | 43 | 2/[31:16] | 87658765 |
| | | | | 1/[23:16] | 65 | | |
| | | | | 1/[31:24] | 87 | | |
| Little | Little | 32 | 32 | 1/[7:0] | 21 | 1/[31:0] | 87654321 |
| | | | | 1/[15:8] | 43 | | |
| | | | | 1/[23:16] | 65 | | |
| | | | | 1/[31:24] | 87 | | |
| Big | Big | 8 | 8 | 1/[31:24] | 12 | 1/[31:24] | 12121212 |
| | | | | 2/[23:16] | 34 | 2/[23:16] | 34343434 |
| | | | | 3/[15:8] | 56 | 3/[15:8] | 56565656 |
| | | | | 4/[7:0] | 78 | 4/[7:0] | 78787878 |

**Table 453. Endian behavior** *…continued*

| Source endian | Destination endian | Source width | Destination width | Source transfer no/byte lane | Source data | Destination transfer no/byte lane | Destination data |
|---|---|---|---|---|---|---|---|
| Big | Big | 8 | 16 | 1/[31:24] | 12 | 1/[15:0] | 12341234 |
| | | | | 2/[23:16] | 34 | 2/[31:16] | 56785678 |
| | | | | 3/[15:8] | 56 | | |
| | | | | 4/[7:0] | 78 | | |
| Big | Big | 8 | 32 | 1/[31:24] | 12 | 1/[31:0] | 12345678 |
| | | | | 2/[23:16] | 34 | | |
| | | | | 3/[15:8] | 56 | | |
| | | | | 4/[7:0] | 78 | | |
| Big | Big | 16 | 8 | 1/[31:24] | 12 | 1/[31:24] | 12121212 |
| | | | | 1/[23:16] | 34 | 2/[23:16] | 34343434 |
| | | | | 2/[15:8] | 56 | 3/[15:8] | 56565656 |
| | | | | 2/[7:0] | 78 | 4/[7:0] | 78787878 |
| Big | Big | 16 | 16 | 1/[31:24] | 12 | 1/[15:0] | 12341234 |
| | | | | 1/[23:16] | 34 | 2/[31:16] | 56785678 |
| | | | | 2/[15:8] | 56 | | |
| | | | | 2/[7:0] | 78 | | |
| Big | Big | 16 | 32 | 1/[31:24] | 12 | 1/[31:0] | 12345678 |
| | | | | 1/[23:16] | 34 | | |
| | | | | 2/[15:8] | 56 | | |
| | | | | 2/[7:0] | 78 | | |
| Big | Big | 32 | 8 | 1/[31:24] | 12 | 1/[31:24] | 12121212 |
| | | | | 1/[23:16] | 34 | 2/[23:16] | 34343434 |
| | | | | 1/[15:8] | 56 | 3/[15:8] | 56565656 |
| | | | | 1/[7:0] | 78 | 4/[7:0] | 78787878 |
| Big | Big | 32 | 16 | 1/[31:24] | 12 | 1/[15:0] | 12341234 |
| | | | | 1/[23:16] | 34 | 2/[31:16] | 56785678 |
| | | | | 1/[15:8] | 56 | | |
| | | | | 1/[7:0] | 78 | | |
| Big | Big | 32 | 32 | 1/[31:24] | 12 | 1/[31:0] | 12345678 |
| | | | | 1/[23:16] | 34 | | |
| | | | | 1/[15:8] | 56 | | |
| | | | | 1/[7:0] | 78 | | |

#### 4.1.6.3 Error conditions

An error during a DMA transfer is flagged directly by the peripheral by asserting an Error response on the AHB bus during the transfer. The DMA Controller automatically disables the DMA stream after the current transfer has completed, and can optionally generate an error interrupt to the CPU. This error interrupt can be masked.

### 4.1.7 Channel hardware

Each stream is supported by a dedicated hardware channel, including source and destination controllers, as well as a FIFO. This enables better latency than a DMA controller with only a single hardware channel shared between several DMA streams and simplifies the control logic.

### 4.1.8 DMA request priority

DMA channel priority is fixed. DMA channel 0 has the highest priority and DMA channel 7 has the lowest priority.

If the DMA Controller is transferring data for the lower priority channel and then the higher priority channel goes active, it completes the number of transfers delegated to the master interface by the lower priority channel before switching over to transfer data for the higher priority channel. In the worst case this is as large as a one quadword.

It is recommended that memory-to-memory transactions use the lowest priority channel. Otherwise other AHB bus masters are prevented from accessing the bus during DMA Controller memory-to-memory transfer.

### 4.1.9 Interrupt generation

A combined interrupt output is generated as an OR function of the individual interrupt requests of the DMA Controller and is connected to the interrupt controller.

## 4.2 DMA system connections

The connection of the DMA Controller to supported peripheral devices is shown in Table 29–454.

**Table 454. Peripheral connections to the DMA controller and matching flow control signals**

| Peripheral Number | DMA Slave | DMACBREQ | DMACSREQ |
|---|---|---|---|
| 0 | SPI0 transmit | x | x |
| 1 | SPI0 receive | x | x |
| 2 | SPI1 transmit | x | x |
| 3 | SPI1 receive | x | x |
| 4 | SPI2 transmit | x | x |
| 5 | SPI2 receive | x | x |
| 6 | UART0 transmit | x | - |
| 7 | UART0 receive | x | - |
| 8 | UART1 transmit | x | - |
| 9 | UART1 receive | x | - |
| 15:10 | reserved | - | - |

In addition to the UART and SPI peripherals, the GPIOs, the WDT, and the timers can be accessed by the GPDMA as a memory-to-memory transaction with no flow control.

### 4.2.1 DMA request signals

The DMA request signals are used by peripherals to request a data transfer. The DMA request signals indicate whether a single or burst transfer of data is required and whether the transfer is the last in the data packet. The DMA available request signals are:

**DMACBREQ[15:0] —** Burst request signals. These cause a programmed burst number of data to be transferred.

**DMACSREQ[15:0] —** Single transfer request signals. These cause a single data to be transferred. The DMA controller transfers a single transfer to or from the peripheral.

**DMACLBREQ[15:0] —** Last burst request signals.

**DMACLSREQ[15:0] —** Last single transfer request signals.

Note that most peripherals do not support all request types.

### 4.2.2 DMA response signals

The DMA response signals indicate whether the transfer initiated by the DMA request signal has completed. The response signals can also be used to indicate whether a complete packet has been transferred. The DMA response signals from the DMA controller are:

**DMACCLR[15:0] —** DMA clear or acknowledge signals. The DMACCLR signal is used by the DMA controller to acknowledge a DMA request from the peripheral.

**DMACTC[15:0] —** DMA terminal count signals. The DMACTC signal can be used by the DMA controller to indicate to the peripheral that the DMA transfer is complete.

## 5. Register overview

The DMA Controller supports 8 channels. Each channel has registers specific to the operation of that channel. Other registers controls aspects of how source peripherals relate to the DMA Controller. There are also global DMA control and status registers.

**Table 455. Register overview: GPDMA (base address 0xE014 0000)**

| Name | Access | Address offset | Description | Reset state |
|---|---|---|---|---|
| **General registers** | | | | |
| DMACIntStat | RO | 0x000 | DMA Interrupt Status Register | 0 |
| DMACIntTCStat | RO | 0x004 | DMA Interrupt Terminal Count Request Status Register | 0 |
| DMACIntTCClear | WO | 0x008 | DMA Interrupt Terminal Count Request Clear Register | - |
| DMACIntErrStat | RO | 0x00C | DMA Interrupt Error Status Register | 0 |
| DMACIntErrClr | WO | 0x010 | DMA Interrupt Error Clear Register | - |
| DMACRawIntTCStat | RO | 0x014 | DMA Raw Interrupt Terminal Count Status Register | 0 |
| DMACRawIntErrStat | RO | 0x018 | DMA Raw Error Interrupt Status Register | 0 |
| DMACEnbldChns | RO | 0x01C | DMA Enabled Channel Register | 0 |
| DMACSoftBReq | R/W | 0x020 | DMA Software Burst Request Register | 0 |
| DMACSoftSReq | R/W | 0x024 | DMA Software Single Request Register | 0 |
| DMACSoftLBReq | R/W | 0x028 | DMA Software Last Burst Request Register | 0 |

**Table 455. Register overview: GPDMA (base address 0xE014 0000)** *…continued*

| Name | Access | Address offset | Description | Reset state |
|---|---|---|---|---|
| DMACSoftLSReq | R/W | 0x02C | DMA Software Last Single Request Register | 0 |
| DMACConfig | R/W | 0x030 | DMA Configuration Register | 0 |
| DMACSync | R/W | 0x034 | DMA Synchronization Register | 0 |
| **Channel 0 registers** | | | | |
| DMACC0SrcAddr | R/W | 0x100 | DMA Channel 0 Source Address Register | 0 |
| DMACC0DestAddr | R/W | 0x104 | DMA Channel 0 Destination Address Register | 0 |
| DMACC0LLI | R/W | 0x108 | DMA Channel 0 Linked List Item Register | 0 |
| DMACC0Control | R/W | 0x10C | DMA Channel 0 Control Register | 0 |
| DMACC0Config | R/W | 0x110 | DMA Channel 0 Configuration Register | 0[1] |
| **Channel 1 registers** | | | | |
| DMACC1SrcAddr | R/W | 0x120 | DMA Channel 1 Source Address Register | 0 |
| DMACC1DestAddr | R/W | 0x124 | DMA Channel 1 Destination Address Register | 0 |
| DMACC1LLI | R/W | 0x128 | DMA Channel 1 Linked List Item Register | 0 |
| DMACC1Control | R/W | 0x12C | DMA Channel 1 Control Register | 0 |
| DMACC1Config | R/W | 0x130 | DMA Channel 1 Configuration Register | 0[1] |
| **Channel 2 registers** | | | | |
| DMACC2SrcAddr | R/W | 0x140 | DMA Channel 2 Source Address Register | 0 |
| DMACC2DestAddr | R/W | 0x144 | DMA Channel 2 Destination Address Register | 0 |
| DMACC2LLI | R/W | 0x148 | DMA Channel 2 Linked List Item Register | 0 |
| DMACC2Control | R/W | 0x14C | DMA Channel 2 Control Register | 0 |
| DMACC2Config | R/W | 0x150 | DMA Channel 2 Configuration Register | 0[1] |
| **Channel 3 registers** | | | | |
| DMACC3SrcAddr | R/W | 0x160 | DMA Channel 3 Source Address Register | 0 |
| DMACC3DestAddr | R/W | 0x164 | DMA Channel 3 Destination Address Register | 0 |
| DMACC3LLI | R/W | 0x168 | DMA Channel 3 Linked List Item Register | 0 |
| DMACC3Control | R/W | 0x16C | DMA Channel 3 Control Register | 0 |
| DMACC3Config | R/W | 0x170 | DMA Channel 3 Configuration Register | 0[1] |
| **Channel 4 registers** | | | | |
| DMACC4SrcAddr | R/W | 0x180 | DMA Channel 4 Source Address Register | 0 |
| DMACC4DestAddr | R/W | 0x184 | DMA Channel 4 Destination Address Register | 0 |
| DMACC4LLI | R/W | 0x188 | DMA Channel 4 Linked List Item Register | 0 |
| DMACC4Control | R/W | 0x18C | DMA Channel 4 Control Register | 0 |
| DMACC4Config | R/W | 0x190 | DMA Channel 4 Configuration Register | 0[1] |
| **Channel 5 registers** | | | | |
| DMACC5SrcAddr | R/W | 0x1A0 | DMA Channel 5 Source Address Register | 0 |
| DMACC5DestAddr | R/W | 0x1A4 | DMA Channel 5 Destination Address Register | 0 |
| DMACC5LLI | R/W | 0x1A8 | DMA Channel 5 Linked List Item Register | 0 |
| DMACC5Control | R/W | 0x1AC | DMA Channel 5 Control Register | 0 |
| DMACC5Config | R/W | 0x1B0 | DMA Channel 5 Configuration Register | 0[1] |
| **Channel 6 registers** | | | | |

**Table 455. Register overview: GPDMA (base address 0xE014 0000)** *…continued*

| Name | Access | Address offset | Description | Reset state |
|---|---|---|---|---|
| DMACC6SrcAddr | R/W | 0x1C0 | DMA Channel 6 Source Address Register | 0 |
| DMACC6DestAddr | R/W | 0x1C4 | DMA Channel 6 Destination Address Register | 0 |
| DMACC6LLI | R/W | 0x1C8 | DMA Channel 6 Linked List Item Register | 0 |
| DMACC6Control | R/W | 01CC | DMA Channel 6 Control Register | 0 |
| DMACC6Config | R/W | 0x1D0 | DMA Channel 6 Configuration Register | 0[1] |
| **Channel 7 registers** | | | | |
| DMACC7SrcAddr | R/W | 0x1E0 | DMA Channel 7 Source Address Register | 0 |
| DMACC7DestAddr | R/W | 0x1E4 | DMA Channel 7 Destination Address Register | 0 |
| DMACC7LLI | R/W | 0x1E8 | DMA Channel 7 Linked List Item Register | 0 |
| DMACC7Control | R/W | 0x1EC | DMA Channel 7 Control Register | 0 |
| DMACC7Config | R/W | 0x1F0 | DMA Channel 7 Configuration Register | 0[1] |

[1] Bit 17 of this register is a read-only status flag.

## 5.1 DMA Interrupt Status Register (DMACIntStat - 0xE014 0000)

The DMACIntStat Register is read-only and shows the status of the interrupts after masking. A HIGH bit indicates that a specific DMA channel interrupt request is active. The request can be generated from either the error or terminal count interrupt requests. Table 29–456 shows the bit assignments of the DMACIntStat Register.

**Table 456. DMA Interrupt Status Register (DMACIntStat - 0xE014 0000)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | IntStat | Status of DMA channel interrupts after masking. Each bit represents one channel:<br>0 - the corresponding channel has no active interrupt request.<br>1 - the corresponding channel does have an active interrupt request. |

## 5.2 DMA Interrupt Terminal Count Request Status Register (DMACIntTCStat - 0xE014 0004)

The DMACIntTCStat Register is read-only and indicates the status of the terminal count after masking. Table 29–457 shows the bit assignments of the DMACIntTCStat Register.

**Table 457. DMA Interrupt Terminal Count Request Status Register (DMACIntTCStat - 0xE014 0004)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | IntTCStat | Terminal count interrupt request status for DMA channels. Each bit represents one channel:<br>0 - the corresponding channel has no active terminal count interrupt request.<br>1 - the corresponding channel does have an active terminal count interrupt request. |

### 5.3 DMA Interrupt Terminal Count Request Clear Register (DMACIntTCClear - 0xE014 0008)

The DMACIntTCClear Register is write-only and clears one or more terminal count interrupt requests. When writing to this register, each data bit that is set HIGH causes the corresponding bit in the status register (DMACIntTCStat) to be cleared. Data bits that are LOW have no effect. Table 29–458 shows the bit assignments of the DMACIntTCClear Register.

**Table 458. DMA Interrupt Terminal Count Request Clear Register (DMACIntTCClear - 0xE014 0008)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 7:0 | IntTCClear | Allows clearing the Terminal count interrupt request (IntTCStat) for DMA channels. Each bit represents one channel:<br>0 - writing 0 has no effect.<br>1 - clears the corresponding channel terminal count interrupt. |

### 5.4 DMA Interrupt Error Status Register (DMACIntErrStat - 0xE014 000C)

The DMACIntErrStat Register is read-only and indicates the status of the error request after masking. Table 29–459 shows the bit assignments of the DMACIntErrStat Register.

**Table 459. DMA Interrupt Error Status Register (DMACIntErrStat - 0xE014 000C)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | IntErrStat | Interrupt error status for DMA channels. Each bit represents one channel:<br>0 - the corresponding channel has no active error interrupt request.<br>1 - the corresponding channel does have an active error interrupt request. |

### 5.5 DMA Interrupt Error Clear Register (DMACIntErrClr - 0xE014 0010)

The DMACIntErrClr Register is write-only and clears the error interrupt requests. When writing to this register, each data bit that is HIGH causes the corresponding bit in the status register to be cleared. Data bits that are LOW have no effect on the corresponding bit in the register. Table 29–460 shows the bit assignments of the DMACIntErrClr Register.

**Table 460. DMA Interrupt Error Clear Register (DMACIntErrClr - 0xE014 0010)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 7:0 | IntErrClr | Writing a 1 clears the error interrupt request (IntErrStat) for DMA channels. Each bit represents one channel:<br>0 - writing 0 has no effect.<br>1 - clears the corresponding channel error interrupt. |

## 5.6 DMA Raw Interrupt Terminal Count Status Register (DMACRawIntTCStat - 0xE014 0014)

The DMACRawIntTCStat Register is read-only and indicates which DMA channel is requesting a transfer complete (terminal count interrupt) prior to masking. (Note: the DMACIntTCStat Register contains the same information after masking.) A HIGH bit indicates that the terminal count interrupt request is active prior to masking. Table 29–461 shows the bit assignments of the DMACRawIntTCStat Register.

**Table 461. DMA Raw Interrupt Terminal Count Status Register (DMACRawIntTCStat - 0xE014 0014)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | RawIntTCStat | Status of the terminal count interrupt for DMA channels prior to masking. Each bit represents one channel:<br>0 - the corresponding channel has no active terminal count interrupt request.<br>1 - the corresponding channel does have an active terminal count interrupt request. |

## 5.7 DMA Raw Error Interrupt Status Register (DMACRawIntErrStat - 0xE014 0018)

The DMACRawIntErrStat Register is read-only and indicates which DMA channel is requesting an error interrupt prior to masking. (Note: the DMACIntErrStat Register contains the same information after masking.) A HIGH bit indicates that the error interrupt request is active prior to masking. Table 29–462 shows the bit assignments of register of the DMACRawIntErrStat Register.

**Table 462. DMA Raw Error Interrupt Status Register (DMACRawIntErrStat - 0xE014 0018)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | RawIntErrStat | Status of the error interrupt for DMA channels prior to masking. Each bit represents one channel:<br>0 - the corresponding channel has no active error interrupt request.<br>1 - the corresponding channel does have an active error interrupt request. |

## 5.8 DMA Enabled Channel Register (DMACEnbldChns - 0xE014 001C)

The DMACEnbldChns Register is read-only and indicates which DMA channels are enabled, as indicated by the Enable bit in the DMACCxConfig Register. A HIGH bit indicates that a DMA channel is enabled. A bit is cleared on completion of the DMA transfer. Table 29–463 shows the bit assignments of the DMACEnbldChns Register.

**Table 463. DMA Enabled Channel Register (DMACEnbldChns - 0xE014 001C)**

| Bit | Name | Function |
|---|---|---|
| 31:8 | - | Reserved. Read undefined. |
| 7:0 | EnabledChannels | Enable status for DMA channels. Each bit represents one channel:<br>0 - DMA channel is disabled.<br>1 - DMA channel is enabled. |

## 5.9 DMA Software Burst Request Register (DMACSoftBReq - 0xE014 0020)

The DMACSoftBReq Register is read/write and enables DMA burst requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting DMA burst transfers. A request can be generated from either a peripheral or the software request register. Each bit is cleared when the related transaction has completed. Table 29–464 shows the bit assignments of the DMACSoftBReq Register.

**Table 464. DMA Software Burst Request Register (DMACSoftBReq - 0xE014 0020)**

| Bit | Name | Function |
|---|---|---|
| 31:16 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 15:0 | SoftBReq | Software burst request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function (refer to Table 29–454 for peripheral hardware connections to the DMA controller):<br>0 - writing 0 has no effect.<br>1 - writing 1 generates a DMA burst request for the corresponding request line. |

**Note:** It is recommended that software and hardware peripheral requests are not used at the same time.

## 5.10 DMA Software Single Request Register (DMACSoftSReq - 0xE014 0024)

The DMACSoftSReq Register is read/write and enables DMA single transfer requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting single DMA transfers. A request can be generated from either a peripheral or the software request register. Table 29–465 shows the bit assignments of the DMACSoftSReq Register.

**Table 465. DMA Software Single Request Register (DMACSoftSReq - 0xE014 0024)**

| Bit | Name | Function |
|---|---|---|
| 31:16 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 15:0 | SoftSReq | Software single transfer request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function:<br>0 - writing 0 has no effect.<br>1 - writing 1 generates a DMA single transfer request for the corresponding request line. |

## 5.11 DMA Software Last Burst Request Register (DMACSoftLBReq - 0xE014 0028)

The DMACSoftLBReq Register is read/write and enables DMA last burst requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting last burst DMA transfers. A request can be generated from either a peripheral or the software request register. Table 29–466 shows the bit assignments of the DMACSoftLBReq Register.

**Table 466. DMA Software Last Burst Request Register (DMACSoftLBReq - 0xE014 0028)**

| Bit | Name | Function |
|---|---|---|
| 31:16 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 15:0 | SoftLBReq | Software last burst request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function:<br>0 - writing 0 has no effect.<br>1 - writing 1 generates a DMA last burst request for the corresponding request line. |

## 5.12 DMA Software Last Single Request Register (DMACSoftLSReq - 0xE014 002C)

The DMACSoftLSReq Register is read/write and enables DMA last single requests to be generated by software. A DMA request can be generated for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Reading the register indicates which sources are requesting last single DMA transfers. A request can be generated from either a peripheral or the software request register. Table 29–467 shows the bit assignments of the DMACSoftLSReq Register.

**Table 467. DMA Software Last Single Request Register (DMACSoftLSReq - 0xE014 002C)**

| Bit | Name | Function |
|---|---|---|
| 31:16 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 15:0 | SoftLSReq | Software last single transfer request flags for each of 16 possible sources. Each bit represents one DMA request line or peripheral function:<br>0 - writing 0 has no effect.<br>1 - writing 1 generates a DMA last single transfer request for the corresponding request line. |

## 5.13 DMA Configuration Register (DMACConfig - 0xE014 0030)

The DMACConfig Register is read/write and configures the operation of the DMA Controller. The endianness of the AHB master interface can be altered by writing to the M bit of this register. The AHB master interface is set to little-endian mode on reset. Table 29–468 shows the bit assignments of the DMACConfig Register.

**Table 468. DMA Configuration Register (DMACConfig - 0xE014 0030)**

| Bit | Name | Function |
|---|---|---|
| 31:3 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 2 | M1 | AHB Master 1 endianness configuration:<br>0 = little-endian mode (default).<br>1 = big-endian mode. |
| 1 | M0 | AHB Master 0 endianness configuration:<br>0 = little-endian mode (default).<br>1 = big-endian mode. |
| 0 | E | DMA Controller enable:<br>0 = disabled (default). Disabling the DMA Controller reduces power consumption.<br>1 = enabled. |

### 5.14 DMA Synchronization Register (DMACSync - 0xE014 0034)

The DMACSync Register is read/write and enables or disables synchronization logic for the DMA request signals. The DMA request signals consist of the DMACBREQ[15:0], DMACSREQ[15:0], DMACLBREQ[15:0], and DMACLSREQ[15:0]. A bit set to 0 enables the synchronization logic for a particular group of DMA requests. A bit set to 1 disables the synchronization logic for a particular group of DMA requests. This register is reset to 0, synchronization logic enabled. Table 29–469 shows the bit assignments of the DMACSync Register.

**Table 469. DMA Synchronization Register (DMACSync - 0xE014 0034)**

| Bit | Name | Function |
|---|---|---|
| 31:16 | - | Reserved. Read undefined. Write reserved bits as zero. |
| 15:0 | DMACSync | Controls the synchronization logic for DMA request signals. Each bit represents one set of DMA request lines as described in the preceding text:<br>0 - synchronization logic for the corresponding DMA request signals are disabled.<br>1 - synchronization logic for the corresponding request line signals are enabled. |

### 5.15 DMA Channel registers

The channel registers are used to program the eight DMA channels. These registers consist of:

- Eight DMACCxSrcAddr Registers.
- Eight DMACCxDestAddr Registers.
- Eight DMACCxLLI Registers.
- Eight DMACCxControl Registers.
- Eight DMACCxConfig Registers.

When performing scatter/gather DMA, the first four of these are automatically updated.

### 5.16 DMA Channel Source Address Registers (DMACCxSrcAddr - 0xE014 01x0)

The eight read/write DMACCxSrcAddr Registers (DMACC0SrcAddr to DMACC7SrcAddr) contain the current source address (byte-aligned) of the data to be transferred. Each register is programmed directly by software before the appropriate channel is enabled. When the DMA channel is enabled this register is updated:

- As the source address is incremented.
- By following the linked list when a complete packet of data has been transferred.

Reading the register when the channel is active does not provide useful information. This is because by the time software has processed the value read, the address may have progressed. It is intended to be read only when the channel has stopped, in which case it shows the source address of the last item read.

Note: The source and destination addresses must be aligned to the source and destination widths.

Table 29–470 shows the bit assignments of the DMACCxSrcAddr Registers.

**Table 470. DMA Channel Source Address Registers (DMACCxSrcAddr - 0xE014 01x0)**

| Bit | Name | Function |
|---|---|---|
| 31:0 | SrcAddr | DMA source address. Reading this register will return the current source address. |

## 5.17 DMA Channel Destination Address registers (DMACCxDestAddr - 0xE014 01x4)

The eight read/write DMACCxDestAddr Registers (DMACC0DestAddr to DMACC7DestAddr) contain the current destination address (byte-aligned) of the data to be transferred. Each register is programmed directly by software before the channel is enabled. When the DMA channel is enabled the register is updated as the destination address is incremented and by following the linked list when a complete packet of data has been transferred. Reading the register when the channel is active does not provide useful information. This is because by the time that software has processed the value read, the address may have progressed. It is intended to be read only when a channel has stopped, in which case it shows the destination address of the last item read. Table 29–471 shows the bit assignments of the DMACCxDestAddr Register.

**Table 471. DMA Channel Destination Address registers (DMACCxDestAddr - 0xE014 01x4)**

| Bit | Name | Function |
|---|---|---|
| 31:0 | DestAddr | DMA Destination address. Reading this register will return the current destination address. |

## 5.18 DMA Channel Linked List Item registers (DMACCxLLI - 0xE014 01x8)

The eight read/write DMACCxLLI Registers (DMACC0LLI to DMACC7LLI) contain a word-aligned address of the next Linked List Item (LLI). If the LLI is 0, then the current LLI is the last in the chain, and the DMA channel is disabled when all DMA transfers associated with it are completed. Programming this register when the DMA channel is enabled may have unpredictable side effects. Table 29–472 shows the bit assignments of the DMACCxLLI Register.

**Table 472. DMA Channel Linked List Item registers (DMACCxLLI - 0xE014 01x8)**

| Bit | Name | Function |
|---|---|---|
| 31:2 | LLI | Linked list item. Bits [31:2] of the address for the next LLI. Address bits [1:0] are 0. |
| 1 | R | Reserved, and must be written as 0, masked on read. |
| 0 | LM | AHB master select for loading the next LLI:<br>0 - AHB Master 0.<br>1 - AHB Master 1. |

## 5.19 DMA channel control registers (DMACCxControl - 0xE014 01xC)

The eight read/write DMACCxControl Registers (DMACC0Control to DMACC7Control) contain DMA channel control information such as the transfer size, burst size, and transfer width. Each register is programmed directly by software before the DMA channel is enabled. When the channel is enabled the register is updated by following the linked list when a complete packet of data has been transferred. Reading the register while the channel is active does not give useful information. This is because by the time software has processed the value read, the channel may have advanced. It is intended to be read only when a channel has stopped. Table 29–473 shows the bit assignments of the DMACCxControl Register.

### 5.19.1 Protection and access information

AHB access information is provided to the source and destination peripherals when a transfer occurs. The transfer information is provided by programming the DMA channel (the Prot bits of the DMACCxControl Register, and the Lock bit of the DMACCxConfig Register). These bits are programmed by software.Peripherals can use this information if necessary. Three bits of information are provided, and are used as shown in Table 29–473.

**Table 473. DMA channel control registers (DMACCxControl - 0xE014 01xC)**

| Bit | Name | Function |
|---|---|---|
| 31 | I | Terminal count interrupt enable bit.<br>0 - the terminal count interrupt is disabled.<br>1 - the terminal count interrupt is enabled. |
| 30 | Prot3 | Indicates that the access is cacheable or not cacheable:<br>0 - access is not cacheable.<br>1 - access is cacheable. |
| 29 | Prot2 | Indicates that the access is bufferable or not bufferable:<br>0 - access is not bufferable.<br>1 - access is bufferable. |
| 28 | Prot1 | Indicates that the access is in user mode or privileged mode:<br>0 - access is in user mode.<br>1 - access is in privileged mode. |
| 27 | DI | Destination increment:<br>0 - the destination address is not incremented after each transfer<br>1 - the destination address is incremented after each transfer. |
| 26 | SI | Source increment:<br>0 - the source address is not incremented after each transfer.<br>1 - the source address is incremented after each transfer. |
| 25 | D | Destination AHB master select:<br>0 - AHB Master 0 selected for destination transfer.<br>1 - AHB Master 1 selected for destination transfer. |
| 24 | S | Source AHB master select:<br>0 - AHB Master 0 selected for source transfer.<br>1 - AHB Master 1 selected for source transfer. |
| 23:21 | DWidth | Destination transfer width. Transfers wider than the AHB master bus width are not supported. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data as required.<br>000 - Byte (8-bit)<br>001 - Halfword (16-bit)<br>010 - Word (32-bit)<br>011 to 111 - Reserved |

**Table 473. DMA channel control registers (DMACCxControl - 0xE014 01xC)** *…continued*

| Bit | Name | Function |
|---|---|---|
| 20:18 | SWidth | Source transfer width. Transfers wider than the AHB master bus width are illegal. The source and destination widths can be different from each other. The hardware automatically packs and unpacks the data as required.<br>000 - Byte (8-bit)<br>001 - Halfword (16-bit)<br>010 - Word (32-bit)<br>011 to 111 - Reserved |
| 17:15 | DBSize | Destination burst size. Indicates the number of transfers that make up a destination burst transfer request. This value must be set to the burst size of the destination peripheral or, if the destination is memory, to the memory boundary size. The burst size is the amount of data that is transferred when the DMACBREQ signal goes active in the destination peripheral.<br>000 - 1<br>001 - 4<br>010 - 8<br>011 - 16<br>100 - 32<br>101 - 64<br>110 - 128<br>111 - 256 |
| 14:12 | SBSize | Source burst size. Indicates the number of transfers that make up a source burst. This value must be set to the burst size of the source peripheral, or if the source is memory, to the memory boundary size. The burst size is the amount of data that is transferred when the DMACBREQ signal goes active in the source peripheral.<br>000 - 1<br>001 - 4<br>010 - 8<br>011 - 16<br>100 - 32<br>101 - 64<br>110 - 128<br>111 - 256 |
| 11:0 | TransferSize | Transfer size. A write to this field sets the size of the transfer when the DMA Controller is the flow controller. The transfer size value must be set before the channel is enabled. Transfer size is updated as data transfers are completed.<br>A read from this field indicates the number of transfers completed on the destination bus. Reading the register when the channel is active does not give useful information because by the time that the software has processed the value read, the channel might have progressed. It is intended to be used only when a channel is enabled and then disabled.<br>The transfer size value is not used if the DMA Controller is not the flow controller. |

## 5.20 Channel Configuration registers (DMACCxConfig - 0xE014 01x0)

The eight DMACCxConfig Registers (DMACC0Config to DMACC7Config) are read/write with the exception of bit[17] which is read-only. Used these to configure the DMA channel. The registers are not updated when a new LLI is requested. Table 29–474 shows the bit assignments of the DMACCxConfig Register.

Table 474. Channel Configuration registers (DMACCxConfig - 0xE014 01x0)

| Bit | Name | Function |
|---|---|---|
| 31:19 | Reserved | Reserved, do not modify, masked on read. |
| 18 | H | Halt:<br>0 = enable DMA requests.<br>1 = ignore further source DMA requests.<br>The contents of the channel FIFO are drained.<br>This value can be used with the Active and Channel Enable bits to cleanly disable a DMA channel. |
| 17 | A | Active:<br>0 = there is no data in the FIFO of the channel.<br>1 = the channel FIFO has data.<br>This value can be used with the Halt and Channel Enable bits to cleanly disable a DMA channel. This is a read-only bit. |
| 16 | L | Lock. When set, this bit enables locked transfers. |
| 15 | ITC | Terminal count interrupt mask. When cleared, this bit masks out the terminal count interrupt of the relevant channel. |
| 14 | IE | Interrupt error mask. When cleared, this bit masks out the error interrupt of the relevant channel. |
| 13:11 | FlowCntrl | Flow control and transfer type. This value indicates the flow controller and transfer type. The flow controller can be the DMA Controller, the source peripheral, or the destination peripheral.<br>The transfer type can be memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-to-peripheral.<br>Refer to Table 29–475 for the encoding of this field. |
| 10:6 | DestPeripheral | Destination peripheral. This value selects the DMA destination request peripheral. This field is ignored if the destination of the transfer is to memory. See Table 29–454 for peripheral identification. |
| 5:1 | SrcPeripheral | Source peripheral. This value selects the DMA source request peripheral. This field is ignored if the source of the transfer is from memory. See Table 29–454 for peripheral identification. |
| 0 | E | Channel enable. Reading this bit indicates whether a channel is currently enabled or disabled:<br>0 = channel disabled.<br>1 = channel enabled.<br>The Channel Enable bit status can also be found by reading the DMACEnbldChns Register.<br>A channel is enabled by setting this bit.<br>A channel can be disabled by clearing the Enable bit. This causes the current AHB transfer (if one is in progress) to complete and the channel is then disabled. Any data in the FIFO of the relevant channel is lost. Restarting the channel by setting the Channel Enable bit has unpredictable effects, the channel must be fully re-initialized.<br>The channel is also disabled, and Channel Enable bit cleared, when the last LLI is reached, the DMA transfer is completed, or if a channel error is encountered.<br>If a channel must be disabled without losing data in the FIFO, the Halt bit must be set so that further DMA requests are ignored. The Active bit must then be polled until it reaches 0, indicating that there is no data left in the FIFO. Finally, the Channel Enable bit can be cleared. |

### 5.20.1 Lock control

The lock control may set the lock bit by writing a 1 to bit 16 of the DMACCxConfig Register. When a burst occurs, the AHB arbiter will not de-grant the master during the burst until the lock is deasserted. The DMA Controller can be locked for a a single burst such as a long source fetch burst or a long destination drain burst. The DMA Controller does not usually assert the lock continuously for a source fetch burst followed by a destination drain burst.

There are situations when the DMA Controller asserts the lock for source transfers followed by destination transfers. This is possible when internal conditions in the DMA Controller permit it to perform a source fetch followed by a destination drain back-to-back.

### 5.20.2 Flow control and transfer type

Table 29–475 lists the bit values of the three flow control and transfer type bits identified in Table 29–474.

**Table 475. Flow control and transfer type bits**

| Bit value | Transfer type | Controller |
|---|---|---|
| 000 | Memory to memory | DMA |
| 001 | Memory to peripheral | DMA |
| 010 | Peripheral to memory | DMA |
| 011 | Source peripheral to destination peripheral | DMA |
| 100 | Source peripheral to destination peripheral | Destination peripheral |
| 101 | Memory to peripheral | Peripheral |
| 110 | Peripheral to memory | Peripheral |
| 111 | Source peripheral to destination peripheral | Source peripheral |

## 6. Using the DMA controller

### 6.1 Programming the DMA controller

All accesses to the DMA Controller internal register must be word (32-bit) reads and writes.

### 6.1.1 Enabling the DMA controller

To enable the DMA controller set the Enable bit in the DMACConfig register.

### 6.1.2 Disabling the DMA controller

To disable the DMA controller:

- Read the DMACEnbldChns register and ensure that all the DMA channels have been disabled. If any channels are active, see Disabling a DMA channel.
- Disable the DMA controller by writing 0 to the DMA Enable bit in the DMACConfig register.

### 6.1.3 Enabling a DMA channel

To enable the DMA channel set the channel enable bit in the relevant DMA channel configuration register. Note that the channel must be fully initialized before it is enabled.

### 6.1.4 Disabling a DMA channel

A DMA channel can be disabled in three ways:

- By writing directly to the channel enable bit. Any outstanding data in the FIFO’s is lost if this method is used.
- By using the active and halt bits in conjunction with the channel enable bit.
- By waiting until the transfer completes. This automatically clears the channel.

#### Disabling a DMA channel and losing data in the FIFO

Clear the relevant channel enable bit in the relevant channel configuration register. The current AHB transfer (if one is in progress) completes and the channel is disabled. Any data in the FIFO is lost.

#### Disabling the DMA channel without losing data in the FIFO

- Set the halt bit in the relevant channel configuration register. This causes any future DMA request to be ignored.
- Poll the active bit in the relevant channel configuration register until it reaches 0. This bit indicates whether there is any data in the channel that has to be transferred.
- Clear the channel enable bit in the relevant channel configuration register

### 6.1.5 Setting up a new DMA transfer

To set up a new DMA transfer:

If the channel is not set aside for the DMA transaction:

1. Read the DMACEnbldChns controller register and find out which channels are inactive.
2. Choose an inactive channel that has the required priority.
3. Program the DMA controller

### 6.1.6 Halting a DMA channel

Set the halt bit in the relevant DMA channel configuration register. The current source request is serviced. Any further source DMA request is ignored until the halt bit is cleared.

### 6.1.7 Programming a DMA channel

1. Choose a free DMA channel with the priority needed. DMA channel 0 has the highest priority and DMA channel 7 the lowest priority.
2. Clear any pending interrupts on the channel to be used by writing to the DMACIntTCClear and DMACIntErrClear register. The previous channel operation might have left interrupt active.
3. Write the source address into the DMACCxSrcAddr register.
4. Write the destination address into the DMACCxDestAddr register.
5. Write the address of the next LLI into the DMACCxLLI register. If the transfer comprises of a single packet of data then 0 must be written into this register.
6. Write the control information into the DMACCxControl register.

7. Write the channel configuration information into the DMACCxConfig register. If the enable bit is set then the DMA channel is automatically enabled.

## 6.2 Flow control

The peripheral that controls the length of the packet is known as the flow controller. The flow controller is usually the DMA Controller where the packet length is programmed by software before the DMA channel is enabled. If the packet length is unknown when the DMA channel is enabled, either the source or destination peripherals can be used as the flow controller.

For simple or low-performance peripherals that know the packet length (that is, when the peripheral is the flow controller), a simple way to indicate that a transaction has completed is for the peripheral to generate an interrupt and enable the processor to reprogram the DMA channel.

The transfer size value (in the DMACCxControl register) is ignored if a peripheral is configured as the flow controller.

When the DMA transfer is completed:

1. The DMA Controller issues an acknowledge to the peripheral in order to indicate that the transfer has finished.
2. A TC interrupt is generated, if enabled.
3. The DMA Controller moves on to the next LLI.

The following sections describe the DMA Controller data flow sequences for the four allowed transfer types:

- Memory-to-peripheral.
- Peripheral-to-memory.
- Memory-to-memory.
- Peripheral-to-peripheral.

Each transfer type can have either the peripheral or the DMA Controller as the flow controller so there are eight possible control scenarios.

Table 29–476 indicates the request signals used for each type of transfer.

**Table 476. DMA request signal usage**

| Transfer direction | Request generator | Flow controller |
|---|---|---|
| Memory-to-peripheral | Peripheral | DMA Controller |
| Memory-to-peripheral | Peripheral | Peripheral |
| Peripheral-to-memory | Peripheral | DMA Controller |
| Peripheral-to-memory | Peripheral | Peripheral |
| Memory-to-memory | DMA Controller | DMA Controller |
| Source peripheral to destination peripheral | Source peripheral and destination peripheral | Source peripheral |
| Source peripheral to destination peripheral | Source peripheral and destination peripheral | Destination peripheral |
| Source peripheral to destination peripheral | Source peripheral and destination peripheral | DMA Controller |

### 6.2.1 Peripheral-to-memory or memory-to-peripheral DMA flow

For a peripheral-to-memory or memory-to-peripheral DMA flow, the following sequence occurs:

1. Program and enable the DMA channel.
2. Wait for a DMA request.
3. The DMA Controller starts transferring data when:
   – The DMA request goes active.
   – The DMA stream has the highest pending priority.
   – The DMA Controller is the bus master of the AHB bus.
4. If an error occurs while transferring the data, an error interrupt is generated and disables the DMA stream, and the flow sequence ends.
5. Decrement the transfer count if the DMA Controller is performing the flow control.
6. If the transfer has completed (indicated by the transfer count reaching 0, if the DMA Controller is performing flow control, or by the peripheral sending a DMA request, if the peripheral is performing flow control):
   – The DMA Controller responds with a DMA acknowledge.
   – The terminal count interrupt is generated (this interrupt can be masked).
   – If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

### 6.2.2 Peripheral-to-peripheral DMA flow

For a peripheral-to-peripheral DMA flow, the following sequence occurs:

1. Program and enable the DMA channel.
2. Wait for a source DMA request.
3. The DMA Controller starts transferring data when:
   – The DMA request goes active.
   – The DMA stream has the highest pending priority.
   – The DMA Controller is the bus master of the AHB bus.
4. If an error occurs while transferring the data an error interrupt is generated, the DMA stream is disabled, and the flow sequence ends.
5. Decrement the transfer count if the DMA Controller is performing the flow control.
6. If the transfer has completed (indicated by the transfer count reaching 0 if the DMA Controller is performing flow control, or by the peripheral sending a DMA request if the peripheral is performing flow control):
   – The DMA Controller responds with a DMA acknowledge to the source peripheral.
   – Further source DMA requests are ignored.
7. When the destination DMA request goes active and there is data in the DMA Controller FIFO, transfer data into the destination peripheral.
8. If an error occurs while transferring the data, an error interrupt is generated, the DMA stream is disabled, and the flow sequence ends.

9. If the transfer has completed it is indicated by the transfer count reaching 0 if the DMA Controller is performing flow control, or by the sending a DMA request if the peripheral is performing flow control. The following happens:
   – The DMA Controller responds with a DMA acknowledge to the destination peripheral.
   – The terminal count interrupt is generated (this interrupt can be masked).
   – If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl Registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

### 6.2.3 Memory-to-memory DMA flow

For a memory-to-memory DMA flow the following sequence occurs:

1. Program and enable the DMA channel.
2. Transfer data whenever the DMA channel has the highest pending priority and the DMA Controller gains mastership of the AHB bus.
3. If an error occurs while transferring the data, generate an error interrupt and disable the DMA stream.
4. Decrement the transfer count.
5. If the count has reached zero:
   – Generate a terminal count interrupt (the interrupt can be masked).
   – If the DMACCxLLI Register is not 0, then reload the DMACCxSrcAddr, DMACCxDestAddr, DMACCxLLI, and DMACCxControl Registers and go to back to step 2. However, if DMACCxLLI is 0, the DMA stream is disabled and the flow sequence ends.

**Note:** Memory-to-memory transfers should be programmed with a low channel priority, otherwise other DMA channels cannot access the bus until the memory-to-memory transfer has finished, or other AHB masters cannot perform any transaction.

## 6.3 Interrupt requests

Interrupt requests can be generated when an AHB error is encountered or at the end of a transfer (terminal count), after all the data corresponding to the current LLI has been transferred to the destination. The interrupts can be masked by programming bits in the relevant DMACCxControl and DMACCxConfig Channel Registers. Interrupt status registers are provided which group the interrupt requests from all the DMA channels prior to interrupt masking (DMACRawIntTCStat and DMACRawIntErrStat), and after interrupt masking (DMACIntTCStat and DMACIntErrStat). The DMACIntStat Register combines both the DMACIntTCStat and DMACIntErrStat requests into a single register to enable the source of an interrupt to be quickly found. Writing to the DMACIntTCClear or the DMACIntErrClr Registers with a bit set HIGH enables selective clearing of interrupts.

### 6.3.1 Hardware interrupt sequence flow

When a DMA interrupt request occurs, the Interrupt Service Routine needs to:

1. Read the DMACIntTCStat Register to determine whether the interrupt was generated due to the end of the transfer (terminal count). A HIGH bit indicates that the transfer completed. If more than one request is active, it is recommended that the highest priority channels be checked first.
2. Read the DMACIntErrStat Register to determine whether the interrupt was generated due to an error occurring. A HIGH bit indicates that an error occurred.
3. Service the interrupt request.
4. For a terminal count interrupt, write a 1 to the relevant bit of the DMACIntTCClr Register. For an error interrupt write a 1 to the relevant bit of the DMACIntErrClr Register to clear the interrupt request.

## 6.4 Address generation

Address generation can be either incrementing or non-incrementing (address wrapping is not supported).

Some devices, especially memories, disallow burst accesses across certain address boundaries. The DMA controller assumes that this is the case with any source or destination area, which is configured for incrementing addressing. This boundary is assumed to be aligned with the specified burst size. For example, if the channel is set for 16-transfer burst to a 32-bit wide device then the boundary is 64-bytes aligned (that is address bits [5:0] equal 0). If a DMA burst is to cross one of these boundaries, then, instead of a burst, that transfer is split into separate AHB transactions.

**Note:** When transferring data to or from the SDRAM, the SDRAM access must always be programmed to 32 bit accesses. The SDRAM memory controller does not support AHB-INCR4 or INCR8 bursts using halfword or byte transfer-size. Start address in SDRAM should always be aligned to a burst boundary address.

### 6.4.1 Word-aligned transfers across a boundary

The channel is configured for 16-transfer bursts, each transfer 32-bits wide, to a destination for which address incrementing is enabled. The start address for the current burst is 0x0C000024, the next boundary (calculated from the burst size and transfer width) is 0x0C000040.

The transfer will be split into two AHB transactions:

- a 7-transfer burst starting at address 0x0C000024
- a 9-transfer burst starting at address 0x0C000040.

## 6.5 Scatter/gather

Scatter/gather is supported through the use of linked lists. This means that the source and destination areas do not have to occupy contiguous areas in memory. Where scatter/gather is not required, the DMACCxLLI Register must be set to 0.

The source and destination data areas are defined by a series of linked lists. Each Linked List Item (LLI) controls the transfer of one block of data, and then optionally loads another LLI to continue the DMA operation, or stops the DMA stream. The first LLI is programmed into the DMA Controller.

The data to be transferred described by a LLI (referred to as the packet of data) usually requires one or more DMA bursts (to each of the source and destination).

### 6.5.1 Linked list items

A Linked List Item (LLI) consists of four words. These words are organized in the following order:

1. DMACCxSrcAddr.
2. DMACCxDestAddr.
3. DMACCxLLI.
4. DMACCxControl.

**Note:** The DMACCxConfig DMA channel Configuration Register is not part of the linked list item.

#### 6.5.1.1 Programming the DMA controller for scatter/gather DMA

To program the DMA Controller for scatter/gather DMA:

1. Write the LLIs for the complete DMA transfer to memory. Each linked list item contains four words:
   – Source address.
   – Destination address.
   – Pointer to next LLI.
   – Control word.

   The last LLI has its linked list word pointer set to 0.
2. Choose a free DMA channel with the priority required. DMA channel 0 has the highest priority and DMA channel 7 the lowest priority.
3. Write the first linked list item, previously written to memory, to the relevant channel in the DMA Controller.
4. Write the channel configuration information to the channel Configuration Register and set the Channel Enable bit. The DMA Controller then transfers the first and then subsequent packets of data as each linked list item is loaded.
5. An interrupt can be generated at the end of each LLI depending on the Terminal Count bit in the DMACCxControl Register. If this bit is set an interrupt is generated at the end of the relevant LLI. The interrupt request must then be serviced and the relevant bit in the DMACIntTCClear Register must be set to clear the interrupt.

#### 6.5.1.2 Example of scatter/gather DMA

See Figure 29–132 for an example of an LLI. A rectangle of memory has to be transferred to a peripheral. The addresses of each line of data are given, in hexadecimal, at the left-hand side of the figure. The LLIs describing the transfer are to be stored contiguously from address 0x20000.

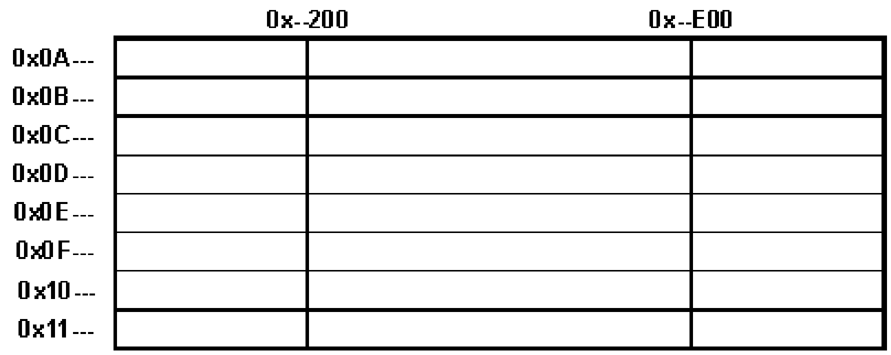


**Fig 132. LLI example**

The first LLI, stored at 0x20000, defines the first block of data to be transferred, which is the data stored between addresses 0x0A200 and 0x0AE00:

- Source start address 0x0A200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0XC00).
- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x20010.

The second LLI, stored at 0x20010, describes the next block of data to be transferred:

- Source start address 0x0B200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0xC00).
- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x20020.

A chain of descriptors is built up, each one pointing to the next in the series. To initialize the DMA stream, the first LLI, 0x20000, is programmed into the DMA Controller. When the first packet of data has been transferred the next LLI is automatically loaded.

The final LLI is stored at 0x20070 and contains:

- Source start address 0x11200.
- Destination address set to the destination peripheral address.
- Transfer width, word (32-bit).
- Transfer size, 3072 bytes (0xC00).
- Source and destination burst sizes, 16 transfers.
- Next LLI address, 0x0.

Because the next LLI address is set to zero, this is the last descriptor, and the DMA channel is disabled after transferring the last item of data. The channel is probably set to generate an interrupt at this point to indicate to the ARM processor that the channel can be reprogrammed.

# UM10316

## Chapter 30: LPC29xx ETM/ETB interface

**Rev. 3 — 19 October 2010** **User manual**

## 1. How to read this chapter

The contents of this chapter apply to all LPC29xx parts.

## 2. Features

- Closely tracks the instructions that the ARM core is executing.
- On-chip trace data storage (ETB).
- All registers are programmed through JTAG interface.
- Does not consume power when trace is not being used.
- THUMB/Java instruction set support.

## 3. Introduction

The ETM provides real-time trace capability for deeply embedded processor cores. It outputs information about processor execution to a trace buffer. A software debugger allows configuration of the ETM using a JTAG interface and displays the trace information that has been captured in a format that a user can easily understand. The ETB stores trace data produced by the ETM.

## 4. Description

The ETM is connected directly to the ARM core and not to the main AMBA system bus. It compresses the trace information and exports it to a trace buffer. An internal Embedded Trace Buffer captures the trace information under software debugger control. ETM can broadcast the Instruction/data trace information. Bytecodes executed while in Java state can also be traced. The trace contains information about when the ARM core switches between states. Instruction trace (or PC trace) shows the flow of execution of the processor and provides a list of all the instructions that were executed. Instruction trace is significantly compressed by only broadcasting branch addresses as well as a set of status signals that indicate the pipeline status on a cycle by cycle basis. For data accesses either data or address or both can be traced. Trace information generation can be controlled by selecting the trigger resource. Trigger resources include address/data comparators, counters and sequencers. Since trace information is compressed the software debugger requires a static image of the code being executed. Self-modifying code can not be traced because of this restriction.

### 4.1 ETM9 configuration

The following standard configuration is selected for the ETM9 macrocell.

Table 477. ETM configuration

| Resource description | Qty.[1] |
|---|---|
| Pairs of address comparators | 8 |
| Data Comparators | 8 |
| Memory Map Decoders | 16 |
| Counters | 4 |
| Sequencer Present | Yes |
| External Inputs | 4 (Not brought out) |
| External Outputs | 4 (Not brought out) |
| FIFOFULL Present | Yes |
| FIFO depth | 45 bytes |
| Trace Packet Width | 4/8/16 (Trace pins are not brought out) |

[1] For details refer to ARM documentation “Embedded Trace Macrocell Specification (ARM IHI 0014E)”.

### 4.2 ETB configuration

The ETB has a 2048 $\times$ 24 bit RAM for instruction/data history storage.

## 5. Block diagram

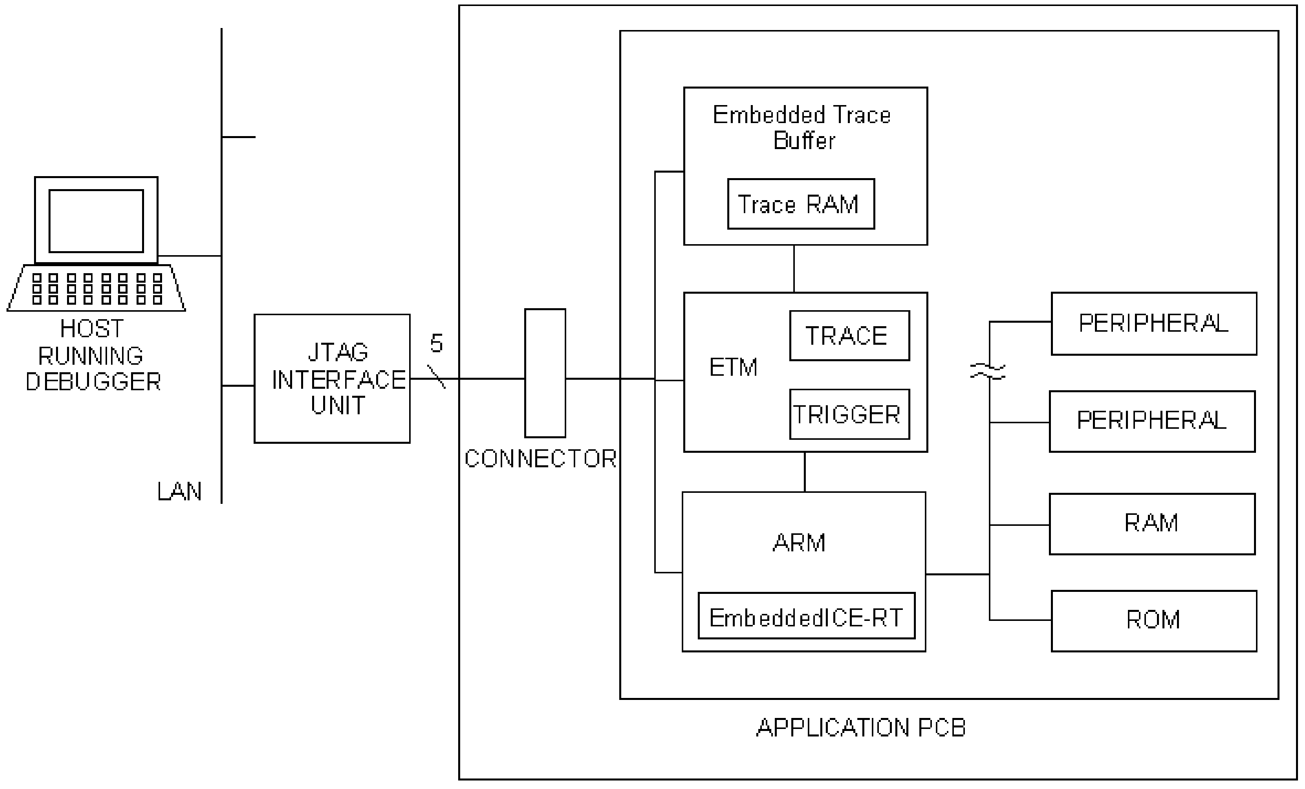


Fig 133. ETM/ETB debug environment block diagram

## 6. Register description

### 6.1 ETM registers

Please refer to ARM9 Embedded Trace Macrocell (ETM9) Technical Reference manual published by ARM

### 6.2 ETB registers

Please refer to Embedded Trace Buffer Technical Reference manual published by ARM.

## 31.1 Abbreviations

Table 478. Abbreviations

| Acronym | Description |
|---|---|
| ADC | Analog to Digital Converter |
| BEMF | Back Electromotive Force |
| BIST | Built-In Self Test |
| BLDCM | Brushless Direct Current Motor |
| BSDL | Boundary-Scan Description Language |
| CAN | Controller Area Network, see Ref. 5 |
| CPU | Central Processing Unit |
| EC-Motor | Electronically Commutated Motor |
| FIFO | First In First Out |
| FIQ | Fast Interrupt |
| FMC | Flash Memory Controller |
| GPIO | General Purpose Input Output |
| IGBT | Insulated Gate Bipolar Transistor |
| IRQ | Interrupt Request |
| ISR | Interrupt Service Routine |
| JTAG | Joint Test Action Group |
| LIN | Local Interconnect Network, see Ref. 6 |
| LUT | Lookup Table |
| MISR | Multiple Input Signature Register |
| MOSFET | Metal Oxide Semiconductor Field-Effect Transistor |
| MSCSS | Modulation and Sampling Control Subsystem |
| OS | Operating System |
| P-Controller | Proportional Controller |
| PI-Controller | Proportional Integral Controller |
| PMSM | Permanent Magnet Synchronous Motor |
| PWM | Pulse Width Modulation |
| RPM | Revolutions Per Minute |
| SRAM | Static RAM |
| TAP | Test Access Point |

## 31.2 Glossary

**ADC —** Analog to Digital Converter; converts an analog input value to a discrete integer value.

**Atomic action —** A number of software instructions executed without the possibility to interrupt them. A common solution is to disable the interrupts before this instruction sequence, and to restore the interrupt allowance afterwards.

**Baud rate —** The number of symbols transmitted/received per second. The size of the symbol depends on the protocol. Often used to indicate the number of bits per second.

**Byte lane —** An 8-bit wide bus (or 8 bits of a bus) used to pass individual data bytes.

**CAN —** Controller Area Network Ref. 5; a multicast shared serial bus standard for connecting electronic control units.

**CC —** Communication Controller.

**CGU —** Clock Generation Unit; controls clock sources and clock domains.

**EmbeddedICE —** Embedded In-Circuit Emulator; integrated unit to support debugging the ARM core via the JTAG interface (in ARM debug mode), see Ref. 2.

**ER —** Event Router; routes wake-up events and external interrupts (to CGU/VIC).

**FIFO —** First In First Out queuing/buffering mechanism.

**Flash-word —** An element containing 128 bits (16 bytes). The flash memory is arranged to store and access these elements.

**FMC —** Flash Memory Controller; controller for the internal flash memory.

**GPIO —** General Purpose Input/Output; controller for I/O pins.

**Half word —** In the context of ARM processors, an element containing 16 bits (2 bytes).

**JTAG —** Standardized interface (Ref. 4) to support boundary-scan. Also used to access device peripherals, e.g. for debugging.

**LIN —** Local Interconnect Network; low cost serial communication system with single master and rate up to 20 kb/s. Targeted for distributed electronic systems in vehicles.

**SCU —** System Control Unit; configures memory-map and I/O functions.

**SMC —** Static Memory Controller; controller for the external memory banks.

**SPI —** Serial Peripheral Interface; serial interface supporting various industry standard protocols.

**TMR —** Timer; generic timer providing match output and capture inputs.

**WD —** Watchdog; timer to guard (software) execution.

**UART —** Universal Asynchronous Receiver/Transmitter; standard 550 serial port.

**VIC —** Vectored Interrupt Controller; controller for vectored interrupt handling.

**Word —** In the context of ARM processors, an element containing 32 bits (4 bytes).

## 31.3 References

[1] Data Sheet: LPC2917/19/01 ARM9 microcontroller with CAN and LIN controllers.

[2] ARM web site;

[3] ARM PrimeCell Synchronous Serial Port (PL022) Technical Reference Manual;

[4] IEEE Std. 1149.1; IEEE Standard Test Access Port and Boundary-Scan Architecture

[5] ISO 11898-1:2002 Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling

[6] LIN Specification Package, Revision 2.0;

[7] Control of three phase Brushless DC Motors, RWR-501-FT-93094-ft Unclassified Report 026193, F.P.H. Tjin A Ton 1993

[8] 3 Phase Bridge Driver IR2136 Datasheet, International Rectifier 2005

[9] Technology - short and to the point, Maxon Motor AG 2005

[10] Maxon EC-max Datasheet, Maxon Motor AG 2005

[11] TrenchMos transistor BUK7528-55A Datasheet, NXP 2000

[12] MSCSS Requirement Specification v0.1, Rolf van de Burgt, NXP Semiconductors - ABL 2005

[13] OsCan performance measurements on SJA2510 and SJA2020, Janett Honko, NXP Semiconductors - AIC 2006

## 31.4 Legal information

### 31.4.1 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.

### 31.4.2 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.

### 31.4.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

**I$^2$C-bus** — logo is a trademark of NXP B.V.

## 31.5 Tables

## 31.6 Figures

## 31.7 Contents

## Chapter 18: LPC29xx SPI0/1/2

## Chapter 19: LPC29xx Universal Asynchronous Receiver/Transmitter (UART)

## Chapter 20: LPC29xx WatchDog Timer (WDT)

## Chapter 21: LPC29xx CAN 0/1

## Chapter 23: LPC29xx I2C-interface

# Chapter 24: LPC29xx Modulation and Sampling Control Subsystem (MSCSS)

# Chapter 25: LPC29xx Pulse Width Modulator (PWM)

# Chapter 26: LPC29xx Analog-to-Digital Converter (ADC)

## Chapter 30: LPC29xx ETM/ETB interface

## Chapter 31: Supplementary information

**Date of release: 19 October 2010**
**Document identifier: UM10316**