UM10275

LPC2104/2105/2106 User manual

Rev. 02 — 8 April 2009 **User manual**

Document information

Info	Content
Keywords	LPC2104, LPC2104/00, LPC2104/01, LPC2105, LPC2105/00, LPC2105/01, LPC2106, LPC2106/00, LPC2106/01, ARM, ARM7, 32-bit, Microcontroller
Abstract	LPC2104, LPC2105, LPC2106 User manual including /00 and /01 parts.

founded by Philips

Revision history

Rev	Date	Description
02	20090408	LPC2104/05/06 User manual revision.
		Modifications: • Part ID numbers added. • EXTPOLAR and EXTMODE registers are available in /01 parts only. • UART0/1: DLL register value must be 3 or greater.
01	20080604	LPC2104/05/06 User manual revision.

Contact information

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

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

1. Introduction

The LPC2104/05/06 are based on a 16/32-bit ARM7TDMI-S CPU with real-time emulation and embedded trace support, together with 128 kB of embedded high speed flash memory. A 128-bit wide memory interface and a unique accelerator architecture enable 32-bit code execution at maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty.

Due to their tiny size and low power consumption, these microcontrollers are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. With a wide range of serial communications interfaces and on-chip SRAM options up to 64 kB, they are very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, PWM channels, and 32 GPIO lines make these microcontrollers particularly suitable for industrial control and medical systems.

2. How to read this manual

The LPC2104/05/06 user manual covers the following parts and versions: LPC2104, LPC2105, LPC2106 with /00 and /01 versions.

All parts exist in **legacy** versions (no suffix and /00) and **enhanced** versions (/01). Enhanced parts (/01 versions) are equipped with enhanced GPIO, SSP, UART, and timer peripherals. They are also backward compatible to the “legacy” parts (/00 and no-suffix versions) containing legacy versions of the same peripherals. Therefore, enhanced parts contain all features of legacy parts as well.

This user manual describes enhanced features together with legacy features for all LPC2104/05/06 parts. Legacy/enhanced specific registers and configurations are listed in at the beginning of each chapter.

3. New features implemented in LPC2104/05/06/01 devices

- Fast GPIO ports enable port pin toggling up to 3.5 times faster than the original device. They also allow for a port pin to be read at any time regardless of its function.
- UART0/1 include fractional baud rate generator, auto-bauding capabilities and handshake flow-control fully implemented in hardware.
- Buffered SSP serial controller supporting SPI, 4-wire SSI, and Microwire formats.
- SPI programmable data length and master mode enhancement.
- Diversified Code Read Protection (CRP) enables different security levels to be implemented.
- General purpose timers can operate as external event counters.

4. Key common features

- 16/32-bit ARM7TDMI-S processor.
- 16/32/64 kB on-chip static RAM.
- 128 kB on-chip flash program memory. 128-bit-wide interface/accelerator enables high speed 60 MHz operation.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software. Flash programming takes 1 ms per 512 B line. Single sector or full chip erase takes 400 ms.
- Vectored Interrupt Controller with configurable priorities and vector addresses.
- EmbeddedICE-RT interface enables breakpoints and watch points. Interrupt service routines can continue to execute whilst the foreground task is debugged with the on-chip RealMonitor software.
- Embedded Trace Macrocell enables non-intrusive high speed real-time tracing of instruction execution.
- Multiple serial interfaces including two UARTs (16C550), Fast I^2C-bus (400 kbit/s), and SPI.
- Two 32-bit timers (7 capture/compare channels), PWM unit (6 outputs), Real Time Clock and Watchdog.
- Up to thirty-two 5 V tolerant general purpose I/O pins in a tiny LQFP48 (7×7 mm^2) package.
- 60 MHz maximum CPU clock available from programmable on-chip Phase-Locked Loop with settling time of 100 μs.
- The on-chip crystal oscillator should have an operating range of 1 MHz to 25 MHz.
- Two low power modes, Idle and Power-down.
- Processor wake-up from Power-down mode via external interrupt.
- Individual enable/disable of peripheral functions for power optimization.
- Dual power supply:
  - CPU operating voltage range of 1.65 V to 1.95 V (1.8 V $\pm$ 8.3 %).
  - I/O power supply range of 3.0 V to 3.6 V (3.3 V $\pm$ 10 %) with 5 V tolerant I/O pads.

5. Ordering information

Table 1. Ordering information

Type number	Package		
	Name	Description	Version
LPC2104BBD48	LQFP48	plastic low profile quad flat package; 48 leads; body $7 \times 7 \times 1.4$ mm	SOT313-2
LPC2104FBD48/00	LQFP48	plastic low profile quad flat package; 48 leads; body $7 \times 7 \times 1.4$ mm	SOT313-2
LPC2104FBD48/01	LQFP48	plastic low profile quad flat package; 48 leads; body $7 \times 7 \times 1.4$ mm	SOT313-2
LPC2105BBD48	LQFP48	plastic low profile quad flat package; 48 leads; body $7 \times 7 \times 1.4$ mm	SOT313-2

Table 1. Ordering information *...continued*

Type number	Package		
	Name	Description	Version
LPC2105FBD48/00	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2105FBD48/01	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2106FBD48	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2106BBD48	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2106FBD48/00	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2106FBD48/01	LQFP48	plastic low profile quad flat package; 48 leads; body 7 × 7 × 1.4 mm	SOT313-2
LPC2106FHN48	HVQFN48	plastic thermal enhanced very thin quad flat package; no leads; 48 terminals; body 7 × 7 × 0.85 mm	SOT619-1
LPC2106FHN48/00	HVQFN48	plastic thermal enhanced very thin quad flat package; no leads; 48 terminals; body 7 × 7 × 0.85 mm	SOT619-1
LPC2106FHN48/01	HVQFN48	plastic thermal enhanced very thin quad flat package; no leads; 48 terminals; body 7 × 7 × 0.85 mm	SOT619-1

Table 2. Ordering options

Type number	Flash memory	RAM	Temperature range
LPC2104BBD48	128 kB	16 kB	0 °C to +70 °C
LPC2104FBD48/00	128 kB	16 kB	−40 °C to +85 °C
LPC2104FBD48/01	128 kB	16 kB	−40 °C to +85 °C
LPC2105BBD48	128 kB	32 kB	0 °C to +70 °C
LPC2105FBD48/00	128 kB	32 kB	−40 °C to +85 °C
LPC2105FBD48/01	128 kB	32 kB	−40 °C to +85 °C
LPC2106FBD48	128 kB	64 kB	−40 °C to +85 °C
LPC2106BBD48	128 kB	64 kB	0 °C to +70 °C
LPC2106FBD48/00	128 kB	64 kB	−40 °C to +85 °C
LPC2106FBD48/01	128 kB	64 kB	−40 °C to +85 °C
LPC2106FHN48	128 kB	64 kB	−40 °C to +85 °C
LPC2106FHN48/00	128 kB	64 kB	−40 °C to +85 °C
LPC2106FHN48/01	128 kB	64 kB	−40 °C to +85 °C

6. Architectural overview

The LPC2104/05/06 consist of an ARM7TDMI-S CPU with emulation support, the ARM7 Local Bus for interface to on-chip memory controllers, the AMBA Advanced High-performance Bus (AHB) for interface to the interrupt controller, and the ARM

Peripheral Bus (APB, a compatible superset of ARM's AMBA Advanced Peripheral Bus) for connection to on-chip peripheral functions. The LPC2104/05/06 configures the ARM7TDMI-S processor in little-endian byte order.

AHB peripherals are allocated a 2 megabyte range of addresses at the very top of the 4 gigabyte ARM memory space. Each AHB peripheral is allocated a 16 kB address space within the AHB address space. LPC2104/05/06 peripheral functions (other than the interrupt controller) are connected to the APB bus. The AHB to APB bridge interfaces the APB to the bus. APB peripherals are also allocated a 2 megabyte range of addresses, beginning at the 3.5 gigabyte address point. Each APB peripheral is allocated a 16 kB address space within the APB address space.

The connection of on-chip peripherals to device pins is controlled by a Pin Connect Block. This must be configured by software to fit specific application requirements for the use of peripheral functions and pins.

7. ARM7TDMI-S processor

The ARM7TDMI-S is a general purpose 32-bit microprocessor, 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. This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective processor core.

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

The ARM7TDMI-S processor also employs a unique architectural strategy known as THUMB, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

The key idea behind THUMB is that of a super-reduced instruction set. Essentially, the ARM7TDMI-S processor has two instruction sets:

- The standard 32-bit ARM instruction set.
- A 16-bit THUMB instruction 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 processor using 16-bit registers. This is possible because THUMB code operates on the same 32-bit register set as ARM code.

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

The ARM7TDMI-S processor is described in detail in the ARM7TDMI-S data sheet that can be found on official ARM website.

8. On-chip flash memory system

The LPC2104/05/06 incorporate a 128 kB flash memory system. This memory may be used for both code and data storage. Programming of the flash memory may be accomplished in several ways:

- using the serial built-in JTAG interface
- using In System Programming (ISP) and UART
- using In Application Programming (IAP) capabilities

The application program, using the IAP functions, may also erase and/or program the flash while the application is running, allowing a great degree of flexibility for data storage field firmware upgrades, etc. The entire flash memory is available for user code because the boot loader resides in a separate memory location.

The LPC2104/05/06 flash memory provides minimum of 100,000 erase/write cycles and 20 years of data-retention.

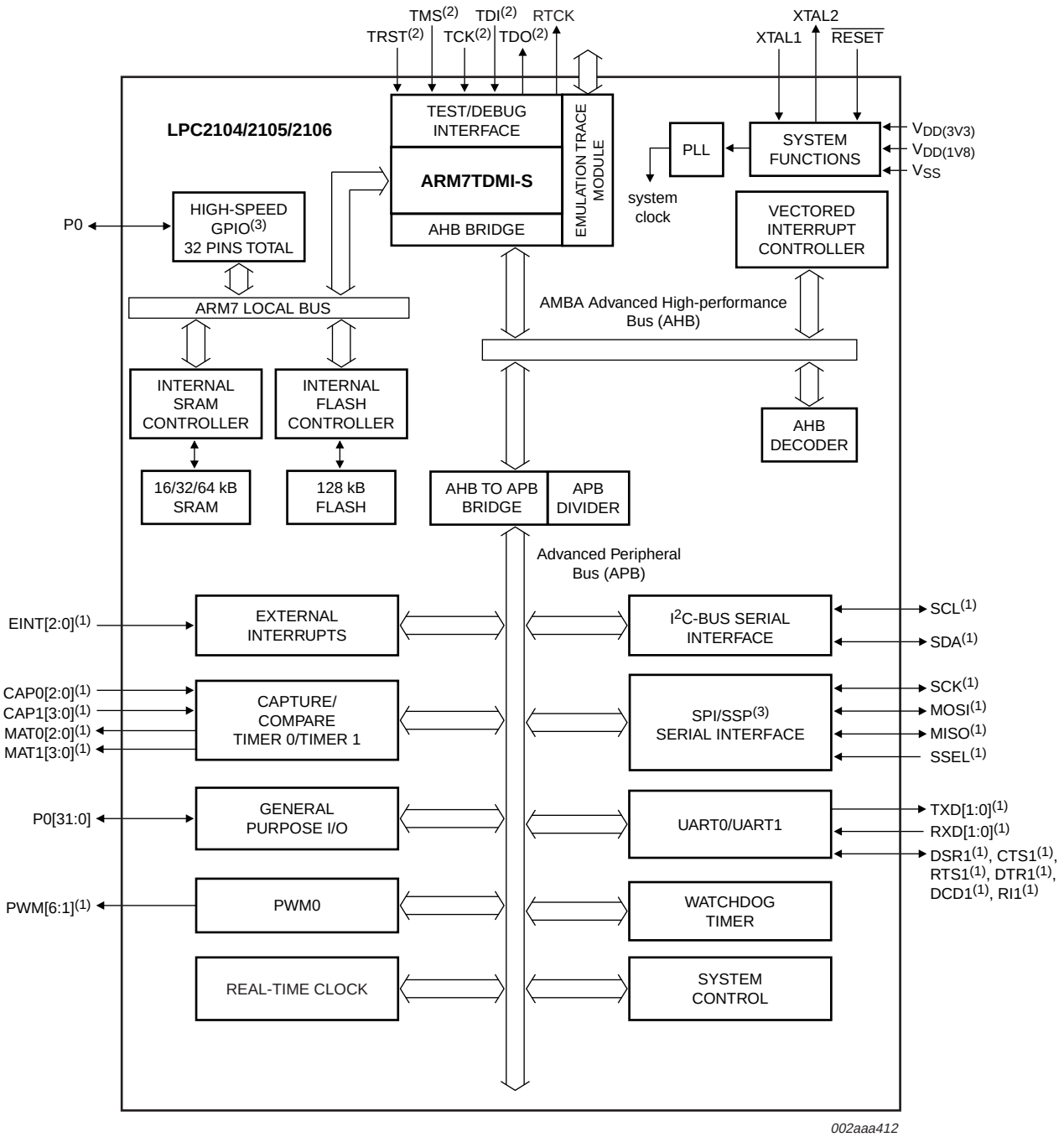
9. On-chip Static RAM (SRAM)

On-chip Static RAM (SRAM) may be used for code and/or data storage. The on-chip SRAM may be accessed as 8-bits, 16-bits, and 32-bits. The LPC2104/05/06 provide 16/32/64 kB of static RAM respectively.

The LPC2104/05/06 SRAM is designed to be accessed as a byte-addressed memory. Word and halfword accesses to the memory ignore the alignment of the address and access the naturally-aligned value that is addressed (so a memory access ignores address bits 0 and 1 for word accesses, and ignores bit 0 for halfword accesses). Therefore valid reads and writes require data accessed as halfwords to originate from addresses with address line 0 being 0 (addresses ending with 0, 2, 4, 6, 8, A, C, and E in hexadecimal notation) and data accessed as words to originate from addresses with address lines 0 and 1 being 0 (addresses ending with 0, 4, 8, and C in hexadecimal notation).

The SRAM controller incorporates a write-back buffer in order to prevent CPU stalls during back-to-back writes. The write-back buffer always holds the last data sent by software to the SRAM. This data is only written to the SRAM when another write is requested by software (the data is only written to the SRAM when software does another write). If a chip reset occurs, actual SRAM contents will not reflect the most recent write request (i.e. after a "warm" chip reset, the SRAM does not reflect the last write operation). Any software that checks SRAM contents after reset must take this into account. Two identical writes to a location guarantee that the data will be present after a Reset. Alternatively, a dummy write operation before entering idle or power-down mode will similarly guarantee that the last data written will be present in SRAM after a subsequent Reset.

10. Block diagram



(1) Shared with GPIO.

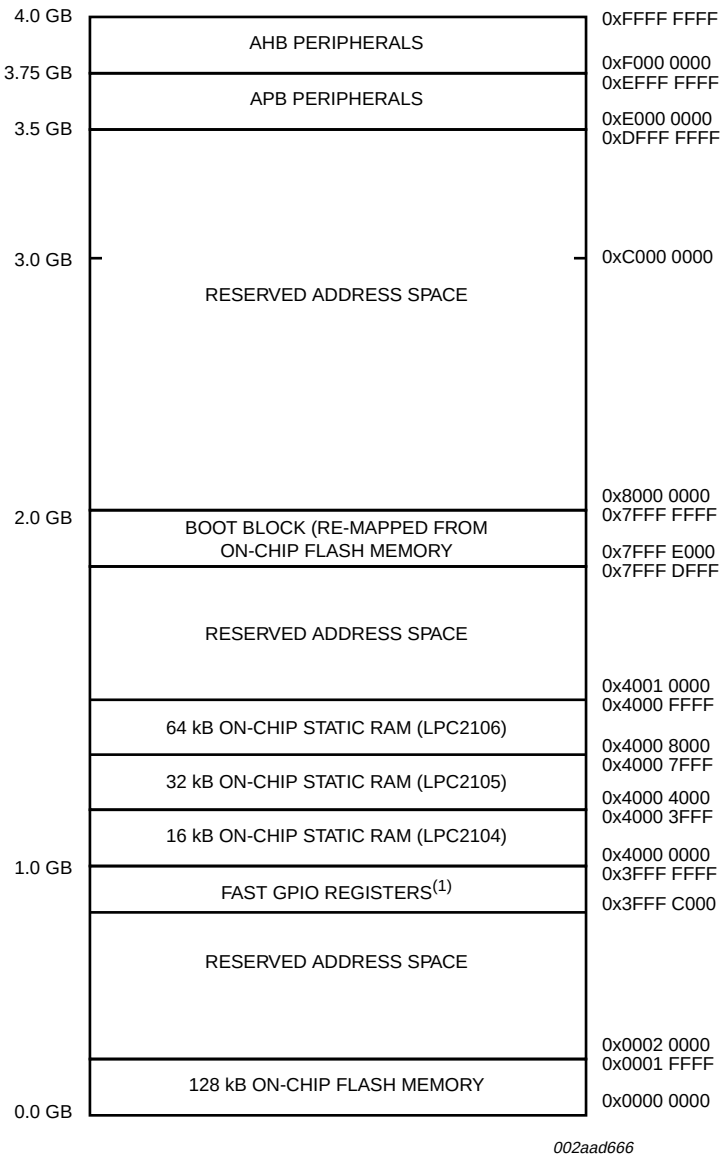
(2) When test/debug interface is used, GPIO/other functions sharing these pins are not available.

(3) Available on LPC2104/2105/2106/01 only.

Fig 1. LPC2104/05/06 block diagram

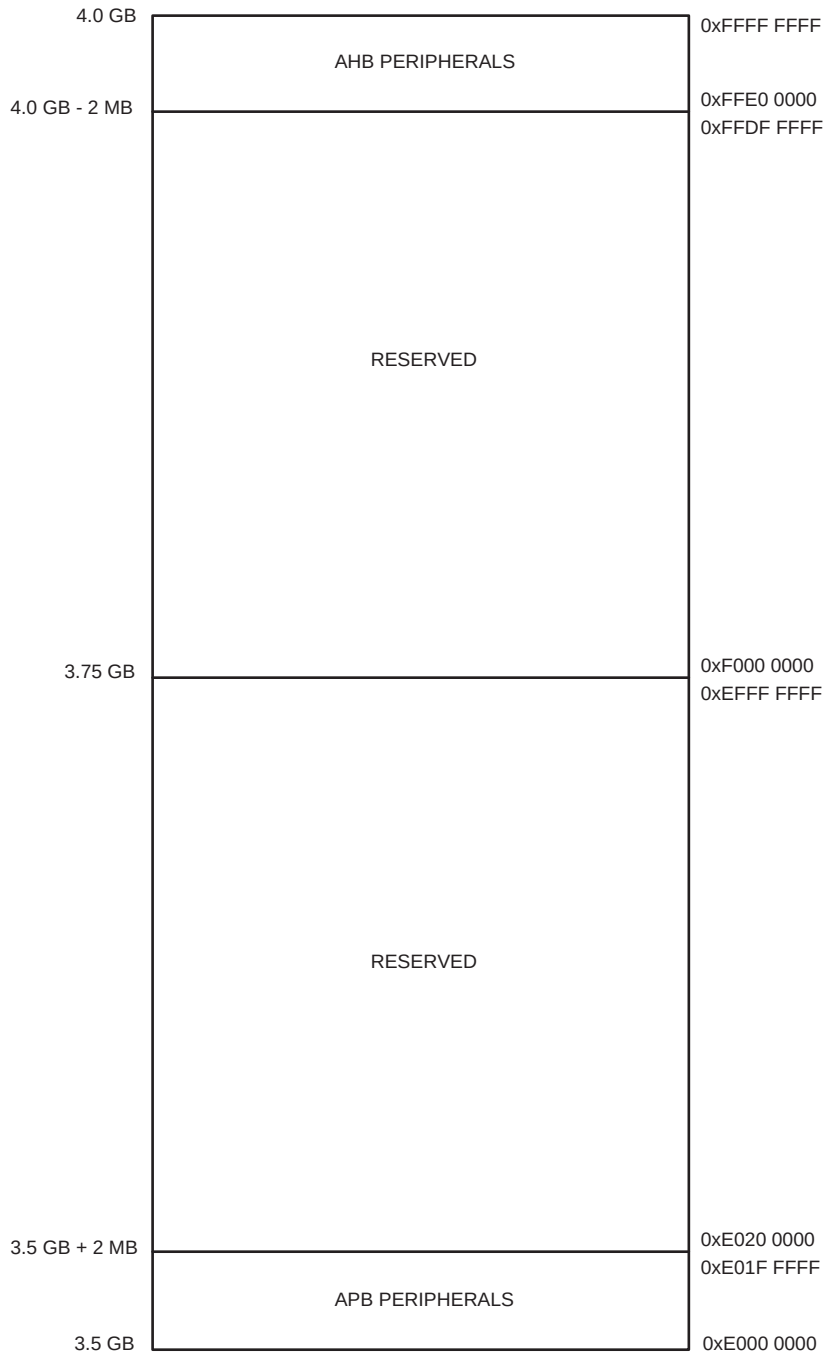
1. Memory map

The LPC2104/05/06 incorporates several distinct memory regions, shown in the following figures. Figure 2–2 shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address remapping, which is described later in this section.



(1) LPC2104/05/06/01 only.

Fig 2. LPC2104/05/06 System memory map



AHB section is 128 x 16 kB blocks (totaling 2 MB).

APB section is 128 x 16 kB blocks (totaling 2MB).

Fig 3. Peripheral memory map

Figure 2–3, Figure 2–4, and Table 2–3 show different views of the peripheral address space. Both the AHB and APB peripheral areas are 2 megabyte spaces which are divided up into 128 peripherals. Each peripheral space is 16 kilobytes in size. This allows simplifying the address decoding for each peripheral. All peripheral register addresses are

word aligned (to 32-bit boundaries) regardless of their size. This eliminates the need for byte lane mapping hardware that would be required to allow byte (8-bit) or half-word (16-bit) accesses to occur at smaller boundaries. An implication of this is that word and half-word registers must be accessed all at once. For example, it is not possible to read or write the upper byte of a word register separately.

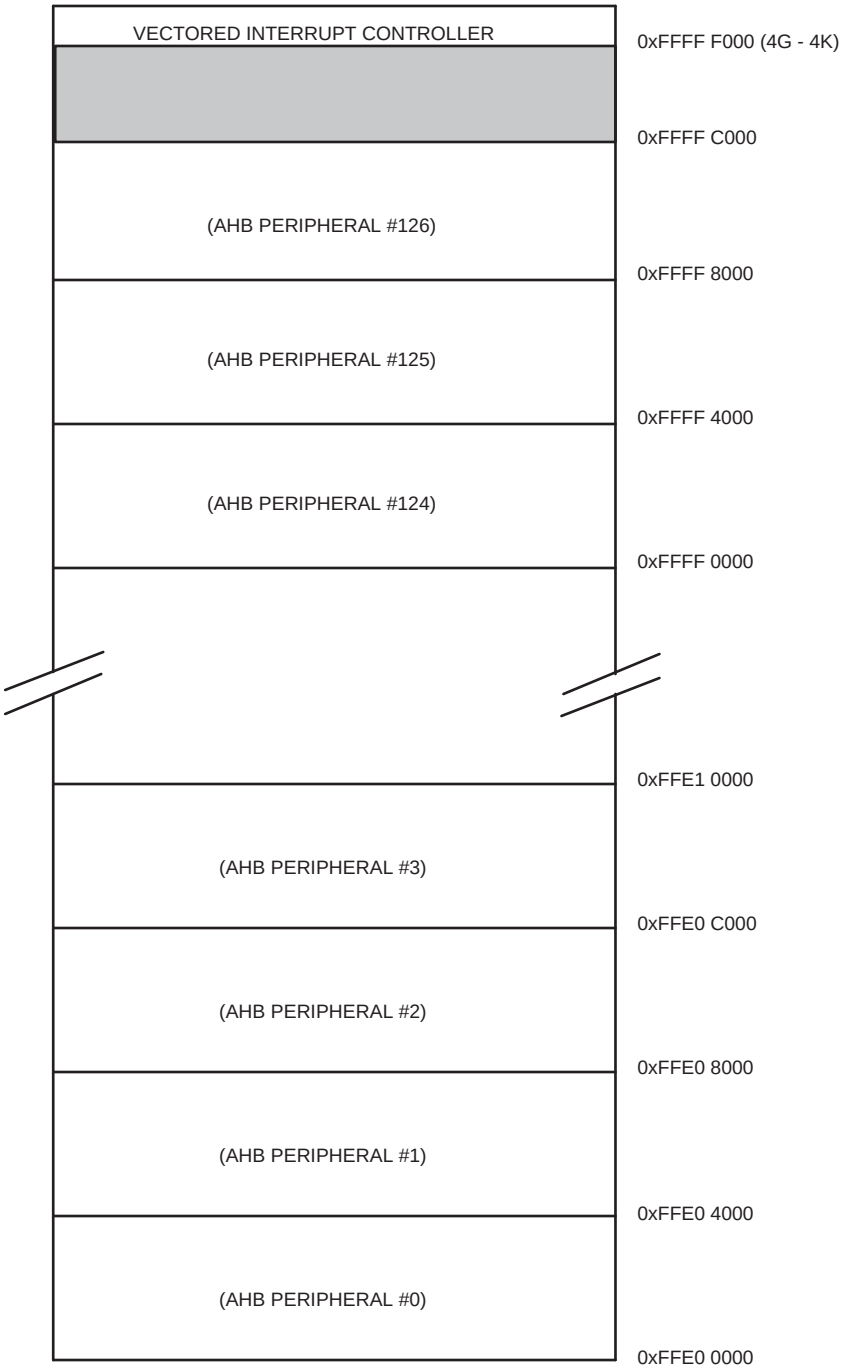


Fig 4. AHB peripheral map

Table 3. APB peripheries and base addresses

APB peripheral	Base address	Peripheral name
0	0xE000 0000	Watchdog timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	PWM
6	0xE001 8000	Not used
7	0xE001 C000	I^2C
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin connect block
12	0xE003 0000	Not used
13	0xE003 4000	Not used
14 - 22	0xE003 8000 - 0xE005 8000	Not used
23	0xE005 C000	SSP
24	0xE006 0000	Not used
25	0xE006 4000	Not used
26	0xE006 8000	Not used
27	0xE006 C000	Not used
28	0xE007 0000	Not used
29	0xE007 4000	Not used
30 - 126	0xE007 8000 - 0xE01F 8000	Not used
127	0xE01F C000	System Control Block

2. LPC2104/05/06 memory re-mapping and boot block

2.1 Memory map concepts and operating modes

The basic concept on the LPC2104/05/06 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. The bulk of 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 interrupt vectors on the ARM7 processor (at addresses 0x0000 0000 through 0x0000 001C, as shown in Table 2–4 below), a small portion of the Boot Block and SRAM spaces need to be re-mapped in order to allow alternative uses of interrupts in the different operating modes described in Table 2–5. Re-mapping of the interrupts is accomplished via the Memory Mapping Control feature (Section 3–8 “Memory mapping control” on page 24).

Table 4. ARM exception vector locations

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved **Note:** Identified as reserved in ARM documentation, this location is used by the Boot Loader as the Valid User Program key. This is described in detail in Section 18–5.2.
0x0000 0018	IRQ
0x0000 001C	FIQ

Table 5. LPC2104/05/06 memory mapping modes

Mode	Activation	Usage
Boot Loader mode	Hardware activation by any Reset	The Boot Loader **always** executes after any reset. The Boot Block interrupt vectors are mapped to the bottom of memory to allow handling exceptions and using interrupts during the Boot Loading process.
User Flash mode	Software activation by Boot code	Activated by Boot Loader when a valid User Program Signature is recognized in memory and Boot Loader operation is not forced. Interrupt vectors are not re-mapped and are found in the bottom of the Flash memory.
User RAM mode	Software activation by User program	Activated by a User Program as desired. Interrupt vectors are re-mapped to the bottom of the Static RAM.

2.2 Memory re-mapping

In order to allow for compatibility with future derivatives, the entire Boot Block is mapped to the top of the on-chip memory space. In this manner, the use of larger or smaller flash modules will not require changing the location of the Boot Block (which would require changing the Boot Loader code itself) or changing the mapping of the Boot Block interrupt vectors. Memory spaces other than the interrupt vectors remain in fixed locations. Figure 2–5 shows the on-chip memory mapping in the modes defined above.

The portion of memory that is re-mapped to allow interrupt processing in different modes includes the interrupt vector area (32 bytes) and an additional 32 bytes, for a total of 64 bytes. The re-mapped code locations overlay addresses 0x0000 0000 through 0x0000 003F. A typical user program in the Flash memory can place the entire FIQ handler at address 0x0000 001C without any need to consider memory boundaries. The vector contained in the SRAM, external memory, and Boot Block must contain branches to the actual interrupt handlers, or to other instructions that accomplish the branch to the interrupt handlers.

There are three reasons this configuration was chosen:

1. To give the FIQ handler in the Flash memory the advantage of not having to take a memory boundary caused by the remapping into account.

2. Minimize the need to for the SRAM and Boot Block vectors to deal with arbitrary boundaries in the middle of code space.
3. To provide space to store constants for jumping beyond the range of single word branch instructions.

Re-mapped memory areas, including the interrupt vectors, continue to appear in their original location in addition to the re-mapped address.

Details on re-mapping and examples can be found in Section 3–8 “Memory mapping control” on page 24.

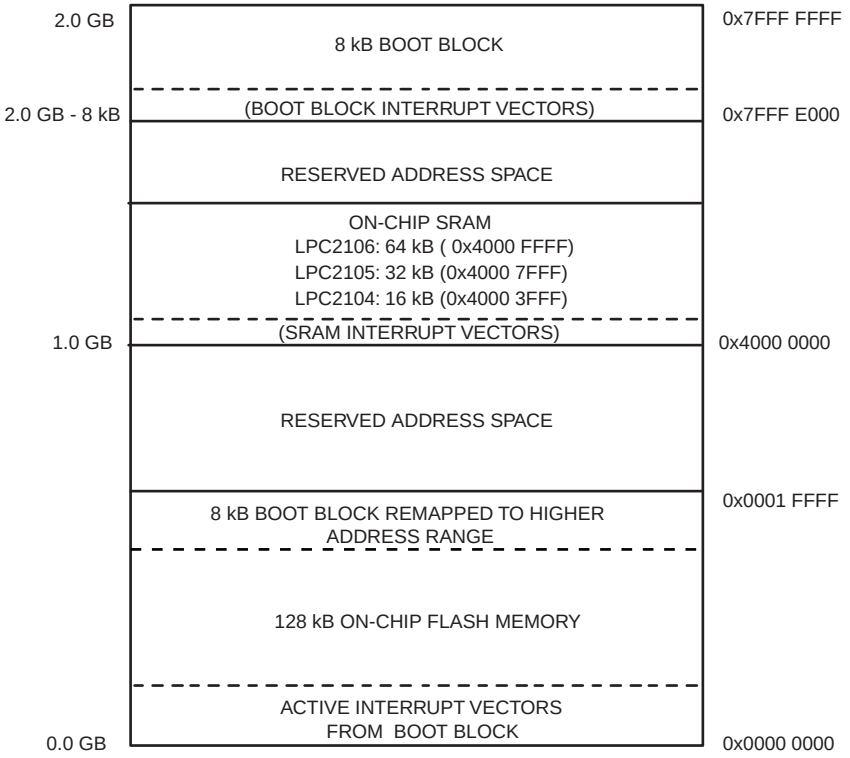


Fig 5. Map of lower memory is showing re-mapped and re-mappable areas

3. Prefetch abort and data abort exceptions

The LPC2104/05/06 generates the appropriate bus cycle abort exception if an access is attempted for an address that is in a reserved or unassigned address region. The regions are:

- Areas of the memory map that are not implemented for a specific ARM derivative. For the LPC2104/05/06, this is:
  - Address space between on-chip Non-Volatile Memory and on-chip SRAM, labelled "Reserved Address Space" in Figure 2–2. For 128 kB Flash device this is memory address range from 0x0001 FFFF to 0x3FFF FFFF.

  - Address space between on-chip Static RAM and the Boot Block. Labelled "Reserved Address Space" in Figure 2–2. For 16 kB SRAM device this is memory address range from 0x4000 4000 to 0x7FFF DFFF, for 32 kB SRAM device this is memory address range from 0x4000 8000 to 0x7FFF DFFF, and for 64 kB SRAM device this range is from 0x4001 000 to 0x7FFF DFFF.
  - Address space between 0x8000 0000 and 0xDFFF FFFF, labelled "Reserved Address Space".
  - Reserved regions of the AHB and APB spaces. See Figure 2–3.
- Unassigned AHB peripheral spaces. See Figure 2–4.
- Unassigned APB peripheral spaces. See Table 2–3.

For these areas, both attempted data access and instruction fetch generate an exception. In addition, a Prefetch Abort exception is generated for any instruction fetch that maps to an AHB or APB peripheral address.

Within the address space of an existing APB 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. For example, an access to address 0xE000 D000 (an undefined address within the UART0 space) may result in an access to the register defined at address 0xE000 C000. Details of such address aliasing within a peripheral space are not defined in the LPC2104/05/06 documentation and are not a supported feature.

Note that the ARM core stores the Prefetch 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 accidental aborts that could be caused by prefetches that occur when code is executed very close to a memory boundary.

1. How to read this chapter

The following registers and register bits are used on **LPC2104/01, LPC2105/01, and LPC2106/01 only**:

- SCS register (see Table 3–14) for fast GPIO access.
- Bit PCSSP in the PCONP register (see Table 3–27) for enabling the SSP interface.
- EXTMODE and EXTPOLAR registers.

2. Introduction

The System Control Block includes several system features and control registers for a number of functions that are not related to specific peripheral devices. These include:

- Crystal Oscillator
- External Interrupt Inputs
- Miscellaneous System Controls and Status
- Memory Mapping Control
- PLL
- Power Control
- Reset
- APB Divider
- Wakeup Timer

Each type of function has its own register(s) if any are required and unneeded bits are defined as reserved in order to allow future expansion. Unrelated functions never share the same register addresses

3. Pin description

Table 3–6 shows pins that are associated with System Control block functions.

Table 6. Pin summary

Pin name	Pin direction	Pin description
XTAL1	Input	**Crystal Oscillator Input** - Input to the oscillator and internal clock generator circuits
XTAL2	Output	**Crystal Oscillator Output** - Output from the oscillator amplifier
EINT0	Input	**External Interrupt Input 0** - An active low/high level or falling/rising edge general purpose interrupt input. This pin may be used to wake up the processor from Idle or Power-down modes. Pin P0.16 can be selected to perform EINT0 function.

Table 6. Pin summary

Pin name	Pin direction	Pin description
EINT1	Input	**External Interrupt Input 1** - See the EINT0 description above. Pin P0.14 can be selected to perform EINT1 function. **Important:** LOW level on pin P0.14 immediately after reset is considered as an external hardware request to start the ISP command handler. More details on ISP and Serial Boot Loader can be found in Section 18–5 on page 225.
EINT2	Input	**External Interrupt Input 2** - See the EINT0 description above. Pin P0.15 can be selected to perform EINT2 function.
$\overline{\text{RESET}}$	Input	**External Reset input** - A LOW on this pin resets the chip, causing I/O ports and peripherals to take on their default states and the processor to begin execution at address 0x0000 0000.

4. Register description

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

Table 7. Summary of system control registers

Name	Description	Access	Reset value[1]	Address
External Interrupts				
EXTINT	External Interrupt Flag Register	R/W	0	0xE01F C140
EXTWAKE	External Interrupt Wakeup Register	R/W	0	0xE01F C144
EXTMODE[2]	External Interrupt Mode Register	R/W	0	0xE01F C148
EXTPOLAR[2]	External Interrupt Polarity Register	R/W	0	0xE01F C14C
Memory Mapping Control				
MEMMAP	Memory Mapping Control	R/W	0	0xE01F C040
Phase Locked Loop				
PLLCON	PLL Control Register	R/W	0	0xE01F C080
PLLCFG	PLL Configuration Register	R/W	0	0xE01F C084
PLLSTAT	PLL Status Register	RO	0	0xE01F C088
PLLFEED	PLL Feed Register	WO	NA	0xE01F C08C
Power Control				
PCON	Power Control Register	R/W	0	0xE01F C0C0
PCONP	Power Control for Peripherals	R/W	see Table 3–27	0xE01F C0C4
APB Divider				
APBDIV	APB Divider Control	R/W	0	0xE01F C100
Syscon Miscellaneous Registers				
SCS[2]	System Controls and Status	R/W	0	0xE01F C1A0

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

[2] LPC2104/05/06/01 only.

5. Crystal oscillator

An input signal of 50-50 duty cycle within a frequency range from 1 MHz to 25 MHz can be used by the LPC2104/05/06 if supplied to its input XTAL1 pin. This microcontroller's onboard oscillator circuit supports external crystals in the range of 1 MHz to 25 MHz only. If the on-chip PLL system or the boot-loader is used, the input clock frequency is limited to an exclusive range of 10 MHz to 25 MHz.

The oscillator output frequency is called F_{OSC}, and the ARM processor clock frequency is referred to as CCLK for purposes of rate equations, etc. elsewhere in this document. F_{OSC} and CCLK are the same value unless the PLL is running and connected. Refer to the Section 3–9 "Phase Locked Loop (PLL)" on page 24 for details and frequency limitations.

The on-board oscillator in the LPC2104/05/06 can operate in one of two modes: slave mode and oscillation mode.

In slave mode the input clock signal should be coupled by means of a capacitor of 100 pF (C_C in Figure 3–6, drawing a), with an amplitude of at least 200 mVrms. The XTAL2 pin in this configuration can be left not connected. If slave mode is selected, the F_{OSC} signal of 50-50 duty cycle can range from 1 MHz to 25 MHz.

External components and models used in oscillation mode are shown in Figure 3–6, drawings b and c, and in Table 3–8. Since the feedback resistance is integrated on chip, only a crystal and the capacitances C_{X1} and C_{X2} need to be connected externally in case of fundamental mode oscillation (the fundamental frequency is represented by L, C_L and R_S). Capacitance C_P in Figure 3–6, drawing c, represents the parallel package capacitance and should not be larger than 7 pF. Parameters F_C, C_L, R_S and C_P are supplied by the crystal manufacturer.

Choosing the oscillation mode as an on-board oscillator mode of operation, limits F_{OSC} clock selection to 1 MHz to 25 MHz.

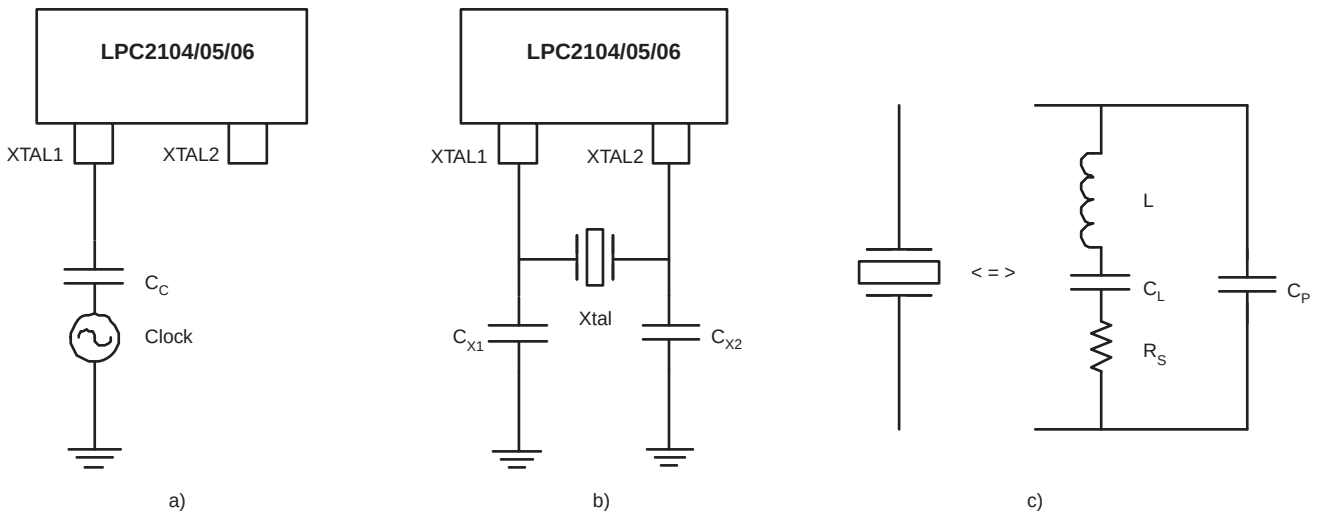


Fig 6. Oscillator modes and models: a) slave mode of operation, b) oscillation mode of operation, c) external crystal model used for $C_{X1}/_{X2}$ evaluation

Table 8. Recommended values for $C_{X1/X2}$ in oscillation mode (crystal and external components parameters)

Fundamental oscillation frequency F_{OSC}	Crystal load capacitance C_L	Maximum crystal series resistance R_S	External load capacitors C_{X1}, C_{X2}
1 MHz - 5 MHz	10 pF	NA	NA
	20 pF	NA	NA
	30 pF	< 300 Ω	58 pF, 58 pF
5 MHz - 10 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 300 Ω	38 pF, 38 pF
	30 pF	< 300 Ω	58 pF, 58 pF
10 MHz - 15 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 220 Ω	38 pF, 38 pF
	30 pF	< 140 Ω	58 pF, 58 pF
15 MHz - 20 MHz	10 pF	< 220 Ω	18 pF, 18 pF
	20 pF	< 140 Ω	38 pF, 38 pF
	30 pF	< 80 Ω	58 pF, 58 pF
20 MHz - 25 MHz	10 pF	< 160 Ω	18 pF, 18 pF
	20 pF	< 90 Ω	38 pF, 38 pF
	30 pF	< 50 Ω	58 pF, 58 pF

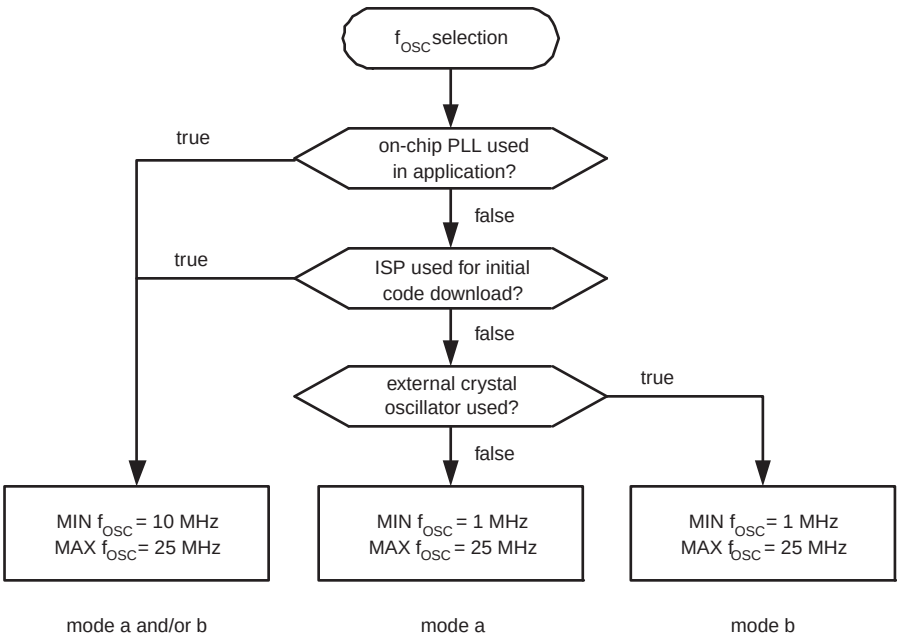


Fig 7. F_{OSC} selection algorithm

6. External interrupt inputs

The LLPC2104/05/06 includes four external interrupt inputs as selectable pin functions. The external interrupt inputs can optionally be used to wake up the processor from Power-down mode.

6.1 Register description

The external interrupt function has four registers associated with it. The EXTINT register contains the interrupt flags, and the EXTWAKE register contains bits that enable individual external interrupts to wake up the microcontroller from Power-down mode. The EXTMODE and EXTPOLAR registers specify the level and edge sensitivity parameters.

Table 9. External interrupt registers

Name	Description	Access	Reset value[1]	Address
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2. See Table 3–10.	R/W	0	0xE01F C140
EXTWAKE	The External Interrupt Wakeup Register contains four enable bits that control whether each external interrupt will cause the processor to wake up from Power-down mode. See Table 3–11.	R/W	0	0xE01F C144
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level sensitive.	R/W	0	0xE01F C148
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt.	R/W	0	0xE01F C14C

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

6.2 External Interrupt Flag register (EXTINT - 0xE01F C140)

When a pin is selected for its external interrupt function, the level or edge on that pin (selected by its bits in the EXTPOLAR and EXTMODE registers) will set its interrupt flag in this register. This asserts the corresponding interrupt request to the VIC, which will cause an interrupt if interrupts from the pin are enabled.

Writing ones to bits EINT0 through EINT2 in EXTINT register clears the corresponding bits. In level-sensitive mode this action has an effect only when the pin is in its inactive state.

Once a bit from EINT0 to EINT2 is set and an appropriate code starts to execute (handling wakeup and/or external interrupt), this bit in EXTINT register must be cleared. Otherwise the event that was just triggered by activity on the EINT pin will not be recognized in the future.

Important: whenever a change of external interrupt operating mode (i.e. active level/edge) is performed (including the initialization of an external interrupt), the corresponding bit in the EXTINT register must be cleared! For details see Section 3–6.4 “External Interrupt Mode register (EXTMODE - 0xE01F C148)” **and** Section 3–6.5 “External Interrupt Polarity register (EXTPOLAR - 0xE01F C14C)”.

For example, if a system wakes up from power-down using a low level on external interrupt 0 pin, its post-wakeup code must reset the EINT0 bit in order to allow future entry into the power-down mode. If the EINT0 bit is left set to 1, subsequent attempt(s) to invoke Power-down mode will fail. The same goes for external interrupt handling.

More details on the Power-down mode will be discussed in the following chapters.

Table 10. External Interrupt Flag register (EXTINT - address 0xE01F C140) bit description

Bit	Symbol	Description	Reset value
0	EINT0	In level-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT0 function is selected for its pin, and the selected edge occurs on the pin. One pin can be selected to perform the EINT0 function (see P0.16 description in Section 6–3). This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state (e.g. if EINT0 is selected to be low level sensitive and a low level is present on the corresponding pin, this bit can not be cleared; this bit can be cleared only when the signal on the pin becomes high).	0
1	EINT1	In level-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT1 function is selected for its pin, and the selected edge occurs on the pin. One pin can be selected to perform the EINT1 function (see P0.14 description in in Section 6–3). This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state (e.g. if EINT1 is selected to be low level sensitive and a low level is present on the corresponding pin, this bit can not be cleared; this bit can be cleared only when the signal on the pin becomes high).	0
2	EINT2	In level-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if the EINT2 function is selected for its pin, and the selected edge occurs on the pin. One pin can be selected to perform the EINT2 function (P0.15 description in in Section 6–3). This bit is cleared by writing a one to it, except in level sensitive mode when the pin is in its active state (e.g. if EINT2 is selected to be low level sensitive and a low level is present on the corresponding pin, this bit can not be cleared; this bit can be cleared only when the signal on the pin becomes high).	0
7:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

6.3 External interrupt Wakeup register (EXTWAKE - 0xE01F C144)

Enable bits in the EXTWAKE register allow the external interrupts and other sources to wake up the processor if it is in Power-down mode. The related EINTn function must be mapped to the pin in order for the wakeup process to take place. It is not necessary for the interrupt to be enabled in the Vectored Interrupt Controller for a wakeup to take place. This arrangement allows additional capabilities, such as having an external interrupt input wake up the processor from Power-down mode without causing an interrupt (simply resuming operation), or allowing an interrupt to be enabled during Power-down without waking the processor up if it is asserted (eliminating the need to disable the interrupt if the wakeup feature is not desirable in the application).

For an external interrupt pin to be a source that would wake up the microcontroller from Power-down mode, it is also necessary to clear the corresponding bit in the External Interrupt Flag register (Section 3–6.2 on page 20).

Table 11. Interrupt Wakeup register (INTWAKE - address 0xE01F C144) bit description

Bit	Symbol	Description	Reset value
0	EXTWAKE0	When one, assertion of EINT0 will wake up the processor from Power-down mode.	0
1	EXTWAKE1	When one, assertion of EINT1 will wake up the processor from Power-down mode.	0
2	EXTWAKE2	When one, assertion of EINT2 will wake up the processor from Power-down mode.	0
7:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

6.4 External Interrupt Mode register (EXTMODE - 0xE01F C148)

The bits in this register select whether each EINT pin is level- or edge-sensitive. Only pins that are selected for the EINT function (see Section 7–2) and enabled via the VICIntEnable register (Section 5–5.4 “Interrupt Enable Register (VICIntEnable - 0xFFFF F010)” on page 48) can cause interrupts from the External Interrupt function (though of course pins selected for other functions may cause interrupts from those functions).

Note: Software should only change a bit in this register when its interrupt is disabled in the VICIntEnable register, and should write the corresponding 1 to the EXTINT register before enabling (initializing) or re-enabling the interrupt, to clear the EXTINT bit that could be set by changing the mode.

Table 12. External Interrupt Mode register (EXTMODE - address 0xE01F C148) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTMODE0	0	Level-sensitivity is selected for EINT0.	0
		1	EINT0 is edge sensitive.	
1	EXTMODE1	0	Level-sensitivity is selected for EINT1.	0
		1	EINT1 is edge sensitive.	
2	EXTMODE2	0	Level-sensitivity is selected for EINT2.	0
		1	EINT2 is edge sensitive.	
7:3	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

6.5 External Interrupt Polarity register (EXTPOLAR - 0xE01F C14C)

In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. Only pins that are selected for the EINT function (see Section 7–2) and enabled in the VICIntEnable register (Section 5–5.4 “Interrupt Enable Register (VICIntEnable - 0xFFFF F010)” on page 48) can cause interrupts from the External Interrupt function (though of course pins selected for other functions may cause interrupts from those functions).

Note: Software should only change a bit in this register when its interrupt is disabled in the VICIntEnable register, and should write the corresponding 1 to the EXTINT register before enabling (initializing) or re-enabling the interrupt, to clear the EXTINT bit that could be set by changing the polarity.

Table 13. External Interrupt Polarity register (EXTPOLAR - address 0xE01F C14C) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTPOLAR0	0	EINT0 is low-active or falling-edge sensitive (depending on EXTMODE0).	0
		1	EINT0 is high-active or rising-edge sensitive (depending on EXTMODE0).	
1	EXTPOLAR1	0	EINT1 is low-active or falling-edge sensitive (depending on EXTMODE1).	0
		1	EINT1 is high-active or rising-edge sensitive (depending on EXTMODE1).	
2	EXTPOLAR2	0	EINT2 is low-active or falling-edge sensitive (depending on EXTMODE2).	0
		1	EINT2 is high-active or rising-edge sensitive (depending on EXTMODE2).	
7:3	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7. Other system controls

Some aspects of controlling LPC2104/05/06 operation that do not fit into peripheral or other registers are grouped here.

7.1 System Control and Status flags register (SCS - 0xE01F C1A0)

Table 14. System Control and Status flags register (SCS - address 0xE01F C1A0) bit description

Bit	Symbol	Value	Description	Reset value
0	GPIO0M		GPIO port 0 mode selection.	0
		0	GPIO port 0 is accessed via APB addresses in a fashion compatible with previous LCP2000 devices.	
		1	High speed GPIO is enabled on GPIO port 0, accessed via addresses in the on-chip memory range. This mode includes the port masking feature described in Section 8–6.2 “Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)”	
31:1	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8. Memory mapping control

The Memory Mapping Control alters the mapping of the interrupt vectors that appear beginning at address 0x0000 0000. This allows code running in different memory spaces to have control of the interrupts.

8.1 Memory Mapping control register (MEMMAP - 0xE01F C040)

Whenever an exception handling is necessary, the microcontroller will fetch an instruction residing on the exception corresponding address as described in Table 2–4 “ARM exception vector locations” on page 13. The MEMMAP register determines the source of data that will fill this table.

Table 15. Memory Mapping control register (MEMMAP - address 0xE01F C040) bit description

Bit	Symbol	Value	Description	Reset value
1:0	MAP	00	Boot Loader Mode. Interrupt vectors are re-mapped to Boot Block.	00[1]
		01	User flash mode. Interrupt vectors are not re-mapped and reside in Flash memory	
		10	User RAM Mode. Interrupt vectors are re-mapped to Static RAM.	
		11	Reserved.	
			Warning: Improper setting of this value may result in incorrect operation of the device.	
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] The hardware reset value of the MAP1:0 bits is 00 for LPC2104/05/06 parts. The apparent reset value visible to the user is different because it is altered by the Boot Loader code, which always runs initially at reset.

8.2 Memory mapping control usage notes

The Memory Mapping Control simply selects one out of three available sources of data (sets of 64 bytes each) necessary for handling ARM exceptions (interrupts).

For example, whenever a Software Interrupt request is generated, the ARM core will always fetch 32-bit data "residing" on 0x0000 0008 see Table 2–4 “ARM exception vector locations” on page 13. This means that when MEMMAP[1:0]=10 (User RAM Mode), a read/fetch from 0x0000 0008 will provide data stored in 0x4000 0008. In case of MEMMAP[1:0]=00 (Boot Loader Mode), a read/fetch from 0x0000 0008 will provide data available also at 0x7FFF E008 (Boot Block remapped from on-chip Bootloader).

9. Phase Locked Loop (PLL)

The PLL accepts an input clock frequency in the range of 10 MHz to 25 MHz only. The input frequency is multiplied up the range of 10 MHz to 75 MHz for the CCLK clock using a Current Controlled Oscillators (CCO). The multiplier can be an integer value from 1 to 32 (in practice, the multiplier value cannot be higher than 7 on the LPC2104/05/06 due to the upper frequency limit of the CPU). The CCO operates in the range of 156 MHz to

320 MHz, so there is an additional divider in the loop to keep the CCO within its frequency range while the PLL is providing the desired output frequency. The output divider may be set to divide by 2, 4, 8, or 16 to produce the output clock. Since the minimum output divider value is 2, it is insured that the PLL output has a 50% duty cycle. A block diagram of the PLL is shown in Figure 3–8.

PLL activation is controlled via the PLLCON register. The PLL multiplier and divider values are controlled by the PLLCFG register. These two registers are protected in order to prevent accidental alteration of PLL parameters or deactivation of the PLL. Since all chip operations, including the Watchdog Timer, are dependent on the PLL when it is providing the chip clock, accidental changes to the PLL setup could result in unexpected behavior of the microcontroller. The protection is accomplished by a feed sequence similar to that of the Watchdog Timer. Details are provided in the description of the PLLFEED register.

The PLL is turned off and bypassed following a chip reset and when by entering Power-down mode. The PLL is enabled by software only. The program must configure and activate the PLL, wait for the PLL to Lock, then connect to the PLL as a clock source.

9.1 Register description

The PLL is controlled by the registers shown in Table 3–16. More detailed descriptions follow.

Warning: Improper setting of the PLL values may result in incorrect operation of the device!

Table 16. PLL registers

Name	Description	Access	Reset value[1]	Address
PLLCON	PLL Control Register. Holding register for updating PLL control bits. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W	0	0xE01F C080
PLLCFG	PLL Configuration Register. Holding register for updating PLL configuration values. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W	0	0xE01F C084
PLLSTAT	PLL Status Register. Read-back register for PLL control and configuration information. If PLLCON or PLLCFG have been written to, but a PLL feed sequence has not yet occurred, they will not reflect the current PLL state. Reading this register provides the actual values controlling the PLL, as well as the status of the PLL.	RO	0	0xE01F C088
PLLFEED	PLL Feed Register. This register enables loading of the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation.	WO	NA	0xE01F C08C

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

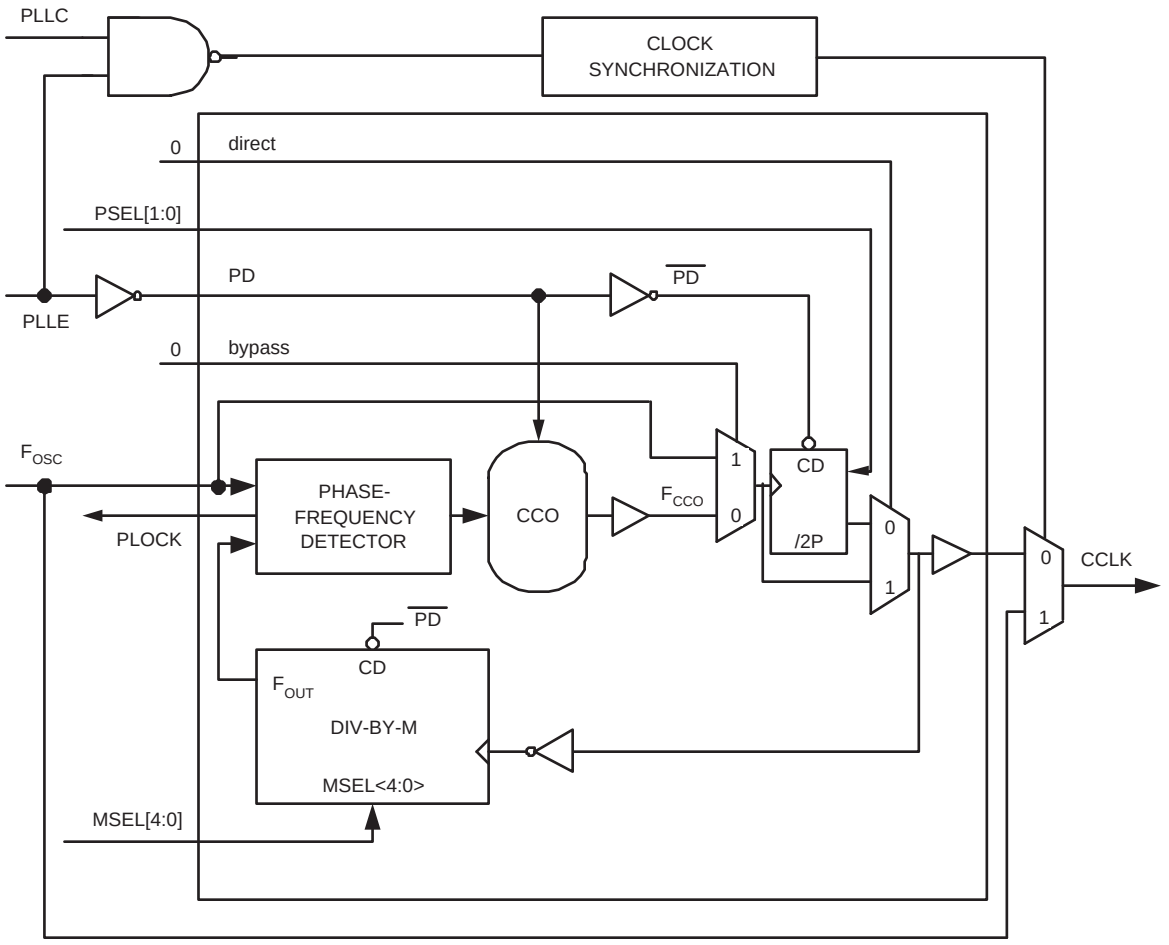


Fig 8. PLL block diagram

9.2 PLL Control register (PLLCON - 0xE01F C080)

The PLLCON register contains the bits that enable and connect the PLL. Enabling the PLL allows it to attempt to lock to the current settings of the multiplier and divider values. Connecting the PLL causes the processor and all chip functions to run from the PLL output clock. Changes to the PLLCON register do not take effect until a correct PLL feed sequence has been given (see Section 3–9.7 “PLL Feed register (PLLFEED - 0xE01F C08C)” and Section 3–9.3 “PLL Configuration register (PLLCFG - 0xE01F C084)” on page 27).

Table 17. PLL Control register (PLLCON - address 0xE01F C080) bit description

Bit	Symbol	Description	Reset value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the PLL and allow it to lock to the requested frequency. See PLLSTAT register, Table 3–19.	0
1	PLLC	PLL Connect. When PLLC and PLLE are both set to one, and after a valid PLL feed, connects the PLL as the clock source for the microcontroller. Otherwise, the oscillator clock is used directly by the microcontroller. See PLLSTAT register, Table 3–19.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

The PLL must be set up, enabled, and Lock established before it may be used as a clock source. When switching from the oscillator clock to the PLL output or vice versa, internal circuitry synchronizes the operation in order to ensure that glitches are not generated. Hardware does not insure that the PLL is locked before it is connected or automatically disconnect the PLL if lock is lost during operation. In the event of loss of PLL lock, it is likely that the oscillator clock has become unstable and disconnecting the PLL will not remedy the situation.

9.3 PLL Configuration register (PLLCFG - 0xE01F C084)

The PLLCFG register contains the PLL multiplier and divider values. Changes to the PLLCFG register do not take effect until a correct PLL feed sequence has been given (see Section 3–9.7 “PLL Feed register (PLLFEED - 0xE01F C08C)” on page 28). Calculations for the PLL frequency, and multiplier and divider values are found in the PLL Frequency Calculation section on page 29.

Table 18. PLL Configuration register (PLLCFG - address 0xE01F C084) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	PLL Multiplier value. Supplies the value "M" in the PLL frequency calculations. **Note:** For details on selecting the right value for MSEL see Section 3–9.9 “PLL frequency calculation” on page 29.	0
6:5	PSEL	PLL Divider value. Supplies the value "P" in the PLL frequency calculations. **Note:** For details on selecting the right value for PSEL see Section 3–9.9 “PLL frequency calculation” on page 29.	0
7	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

9.4 PLL Status register (PLLSTAT - 0xE01F C088)

The read-only PLLSTAT register provides the actual PLL parameters that are in effect at the time it is read, as well as the PLL status. PLLSTAT may disagree with values found in PLLCON and PLLCFG because changes to those registers do not take effect until a proper PLL feed has occurred (see Section 3–9.7 “PLL Feed register (PLLFEED - 0xE01F C08C)”).

Table 19. PLL Status register (PLLSTAT - address 0xE01F C088) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	Read-back for the PLL Multiplier value. This is the value currently used by the PLL.	0
6:5	PSEL	Read-back for the PLL Divider value. This is the value currently used by the PLL.	0
7	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	PLLE	Read-back for the PLL Enable bit. When one, the PLL is currently activated. When zero, the PLL is turned off. This bit is automatically cleared when Power-down mode is activated.	0
9	PLLC	Read-back for the PLL Connect bit. When PLLC and PLLE are both one, the PLL is connected as the clock source for the microcontroller. When either PLLC or PLLE is zero, the PLL is bypassed and the oscillator clock is used directly by the microcontroller. This bit is automatically cleared when Power-down mode is activated.	0
10	PLOCK	Reflects the PLL Lock status. When zero, the PLL is not locked. When one, the PLL is locked onto the requested frequency.	0
15:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

9.5 PLL Interrupt

The PLOCK bit in the PLLSTAT register is connected to the interrupt controller. This allows for software to turn on the PLL and continue with other functions without having to wait for the PLL to achieve lock. When the interrupt occurs (PLOCK = 1), the PLL may be connected, and the interrupt disabled. For details on how to enable and disable the PLL interrupt, see Section 5–5.4 “Interrupt Enable Register (VICIntEnable - 0xFFFF F010)” on page 48 and Section 5–5.5 “Interrupt Enable Clear Register (VICIntEnClear - 0xFFFF F014)” on page 49.

9.6 PLL Modes

The combinations of PLLE and PLLC are shown in Table 3–20.

Table 20. PLL Control bit combinations

PLLC	PLLE	PLL Function
0	0	PLL is turned off and disconnected. The CCLK equals (system runs from) the unmodified clock input.
0	1	The PLL is active, but not yet connected. The PLL can be connected after PLOCK is asserted.
1	0	Same as 00 combination. This prevents the possibility of the PLL being connected without also being enabled.
1	1	The PLL is active and has been connected as the system clock source. CCLK/system clock equals the PLL output.

9.7 PLL Feed register (PLLFEED - 0xE01F C08C)

A correct feed sequence must be written to the PLLFEED register in order for changes to the PLLCON and PLLCFG registers to take effect. The feed sequence is:

1. Write the value 0xAA to PLLFEED.
2. Write the value 0x55 to PLLFEED.

The two writes must be in the correct sequence, and must be consecutive APB bus cycles. The latter requirement implies that interrupts must be disabled for the duration of the PLL feed operation. If either of the feed values is incorrect, or one of the previously mentioned conditions is not met, any changes to the PLLCON or PLLCFG register will not become effective.

Table 21. PLL Feed register (PLLFEED - address 0xE01F C08C) bit description

Bit	Symbol	Description	Reset value
7:0	PLLFEED	The PLL feed sequence must be written to this register in order for PLL configuration and control register changes to take effect.	0x00

9.8 PLL and Power-down mode

Power-down mode automatically turns off and disconnects activated PLL. Wakeup from Power-down mode does not automatically restore the PLL settings, this must be done in software. Typically, a routine to activate the PLL, wait for lock, and then connect the PLL can be called at the beginning of any interrupt service routine that might be called due to the wakeup. It is important not to attempt to restart the PLL by simply feeding it when execution resumes after a wakeup from Power-down mode. This would enable and connect the PLL at the same time, before PLL lock is established.

9.9 PLL frequency calculation

The PLL equations use the following parameters:

Table 22. Elements determining PLL's frequency

Element	Description
F_{OSC}	the frequency from the crystal oscillator/external oscillator
F_{CCO}	the frequency of the PLL current controlled oscillator
CCLK	the PLL output frequency (also the processor clock frequency)
M	PLL Multiplier value from the MSEL bits in the PLLCFG register
P	PLL Divider value from the PSEL bits in the PLLCFG register

The PLL output frequency (when the PLL is both active and connected) is given by:

$CCLK = M \times F_{OSC}$ or $CCLK = F_{CCO} / (2 \times P)$

The CCO frequency can be computed as:

$F_{CCO} = CCLK \times 2 \times P$ or $F_{CCO} = F_{OSC} \times M \times 2 \times P$

The PLL inputs and settings must meet the following:

- F_{OSC} is in the range of 10 MHz to 25 MHz.

- CCLK is in the range of 10 MHz to F_{max} (the maximum allowed frequency for the microcontroller - determined by the system microcontroller is embedded in).
- F_{CCO} is in the range of 156 MHz to 320 MHz.

9.10 Procedure for determining PLL settings

If a particular application uses the PLL, its configuration may be determined as follows:

1. Choose the desired processor operating frequency (CCLK). This may be based on processor throughput requirements, need to support a specific set of UART baud rates, etc. Bear in mind that peripheral devices may be running from a lower clock than the processor (see Section 3–12 “APB divider” on page 35).
2. Choose an oscillator frequency (F_{OSC}). CCLK must be the whole (non-fractional) multiple of F_{OSC}.
3. Calculate the value of M to configure the MSEL bits. M = CCLK / F_{OSC}. M must be in the range of 1 to 32. The value written to the MSEL bits in PLLCFG is M − 1 (see Table 3–24.
4. Find a value for P to configure the PSEL bits, such that F_{CCO} is within its defined frequency limits. F_{CCO} is calculated using the equation given above. P must have one of the values 1, 2, 4, or 8. The value written to the PSEL bits in PLLCFG is 00 for P = 1; 01 for P = 2; 10 for P = 4; 11 for P = 8 (see Table 3–23).

Table 23. PLL Divider values

PSEL Bits (PLLCFG bits [6:5])	Value of P
00	1
01	2
10	4
11	8

Table 24. PLL Multiplier values

MSEL Bits (PLLCFG bits [4:0])	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

9.11 PLL configuring examples

Example: System design asks for F_{OSC}= 10 MHz and requires CCLK = 60 MHz.

Based on these specifications, M = CCLK / Fosc = 60 MHz / 10 MHz = 6. Consequently, M - 1 = 5 will be written as PLLCFG[4:0].

Value for P can be derived from P = F_{CCO} / (CCLK x 2), using condition that F_{CCO} must be in range of 156 MHz to 320 MHz. Assuming the lowest allowed frequency for F_{CCO} = 156 MHz, P = 156 MHz / (2 x 60 MHz) = 1.3. The highest F_{CCO} frequency criteria produces P = 2.67. The only solution for P that satisfies both of these requirements and is listed in Table 3–23 is P = 2. Therefore, PLLCFG[6:5] = 1 will be used.

10. Power control

The LPC2104/05/06 supports two reduced power modes: Idle mode and Power-down mode. In Idle mode, execution of instructions is suspended until either a reset or interrupt occurs. Peripheral functions continue operation during Idle mode and may generate interrupts to cause the processor to resume execution. Idle mode eliminates power used by the processor itself, memory systems and related controllers, and internal buses.

In Power-down mode, the oscillator is shut down and the chip receives no internal clocks. The processor state and registers, peripheral registers, and internal SRAM values are preserved throughout Power-down mode and the logic levels of chip pins remain static. The Power-down mode can be terminated and normal operation resumed by either a reset or certain specific interrupts that are able to function without clocks. Since all dynamic operation of the chip is suspended, Power-down mode reduces chip power consumption to nearly zero.

Entry to Power-down and Idle modes must be coordinated with program execution. Wakeup from Power-down or Idle modes via an interrupt resumes program execution in such a way that no instructions are lost, incomplete, or repeated. Wake up from Power-down mode is discussed further in Section 3–13 “Wakeup timer” on page 36.

A Power Control for Peripherals feature allows individual peripherals to be turned off if they are not needed in the application, resulting in additional power savings.

10.1 Register description

The Power Control function contains two registers, as shown in Table 3–25. More detailed descriptions follow.

Table 25. Power control registers

Name	Description	Access	Reset value[1]	Address
PCON	Power Control Register. This register contains control bits that enable the two reduced power operating modes of the microcontroller. See Table 3–26.	R/W	0x00	0xE01F C0C0
PCONP	Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, Allowing elimination of power consumption by peripherals that are not needed.	R/W	0x0000 1FBE	0xE01F C0C4

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

10.2 Power Control register (PCON - 0xE01F COCO)

The PCON register contains two bits. Writing a one to the corresponding bit causes entry to either the Power-down or Idle mode. If both bits are set, Power-down mode is entered.

Table 26. Power Control register (PCON - address 0xE01F COCO) bit description

Bit	Symbol	Description	Reset value
0	IDL	Idle mode - when 1, this bit causes the processor clock to be stopped, while on-chip peripherals remain active. Any enabled interrupt from a peripheral or an external interrupt source will cause the processor to resume execution.	0
1	PD	Power-down mode - when 1, this bit causes the oscillator and all on-chip clocks to be stopped. A wakeup condition from an external interrupt can cause the oscillator to restart, the PD bit to be cleared, and the processor to resume execution.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

10.3 Power Control for Peripherals register (PCONP - 0xE01F COC4)

The PCONP register allows turning off selected peripheral functions for the purpose of saving power. This is accomplished by gating off the clock source to the specified peripheral blocks. A few peripheral functions cannot be turned off (i.e. the Watchdog timer, GPIO, the Pin Connect block, and the System Control block). Some peripherals, particularly those that include analog functions, may consume power that is not clock dependent. These peripherals may contain a separate disable control that turns off additional circuitry to reduce power. Each bit in PCONP controls one of the peripherals. The bit numbers correspond to the related peripheral number as shown in the APB peripheral map Table 2–3 “APB peripheries and base addresses”.

If a peripheral control bit is 1, that peripheral is enabled. If a peripheral bit is 0, that peripheral is disabled to conserve power. For example, if bit 7 is 1, the I^2C interface is enabled. If bit 7 is 0, the I^2C1 interface is disabled.

Important: valid read from a peripheral register and valid write to a peripheral register is possible only if that peripheral is enabled in the PCONP register!

Table 27. Power Control for Peripherals register (PCONP - address 0xE01F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7	PCI2C	The I^2C interface power/clock control bit.	1
8	PCSPI	The SPI0 interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1

Table 27. Power Control for Peripherals register (PCONP - address 0xE01F C0C4) bit description

Bit	Symbol	Description	Reset value
20:10	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
21	PCSSP	The SSP interface power/clock control bit **Remark:** Setting this bit to 1 and bit 8 (PCSPI) to 0, selects the SSP interface. At reset, SPI is enabled. See Section 13–4 on page 168.	0
31:22	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

10.4 Power control usage notes

After every reset, the PCONP register contains the value that enables all interfaces and peripherals controlled by the PCONP. Therefore, apart from proper configuring via peripheral dedicated registers, the user's application has no need to access the PCONP in order to start using any of the on-board peripherals.

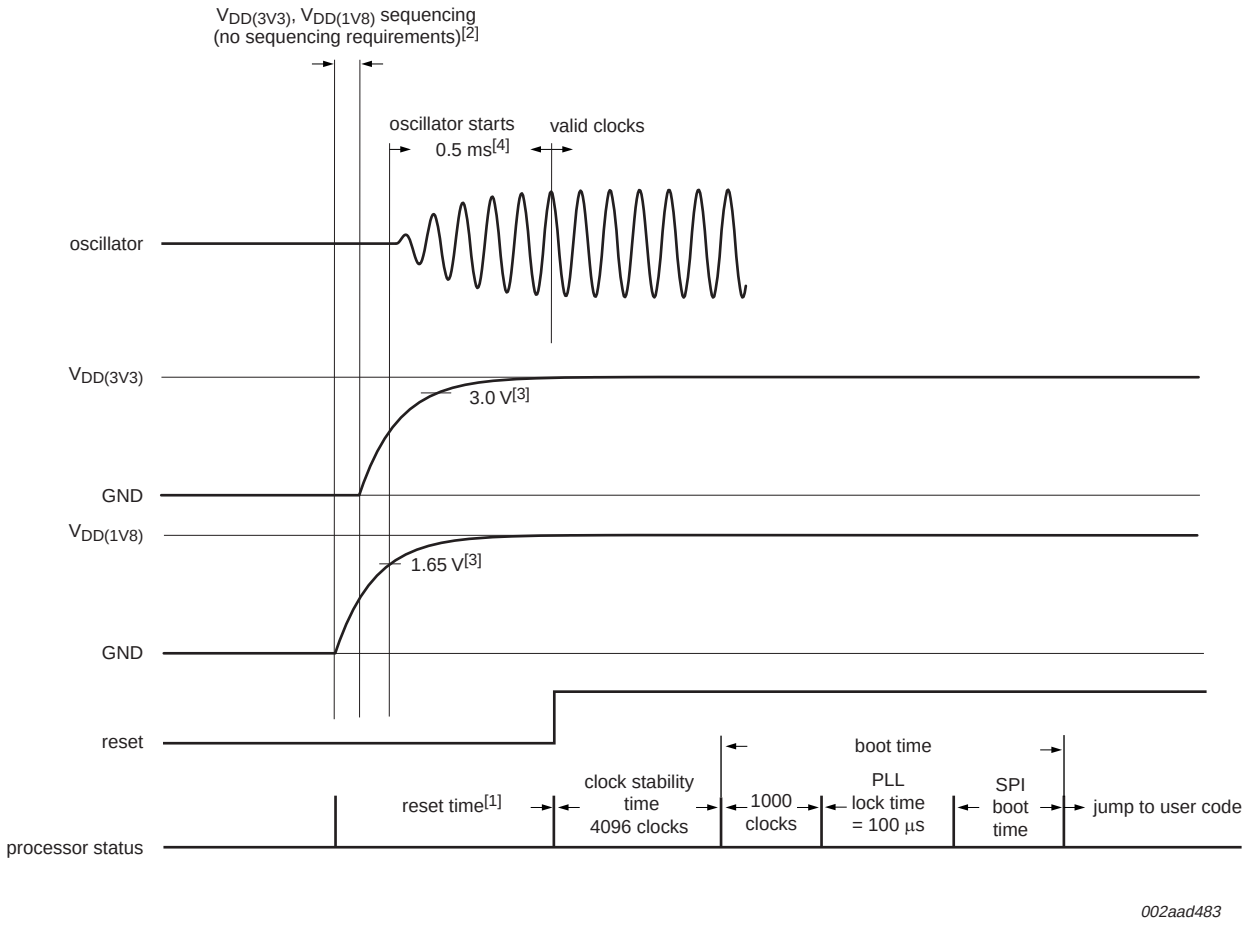
Power saving oriented systems should have 1s in the PCONP register only in positions that match peripherals really used in the application. All other bits, declared to be "Reserved" or dedicated to the peripherals not used in the current application, must be cleared to 0.

11. Reset

Reset has two sources on the LPC2104/05/06: the $\overline{\text{RESET}}$ pin and Watchdog reset. The $\overline{\text{RESET}}$ pin is a Schmitt trigger input pin with an additional glitch filter. Assertion of chip reset by any source starts the wakeup timer (see description in Section 3–13 "Wakeup timer" in this chapter), causing reset to remain asserted until the external reset is de-asserted, the oscillator is running, a fixed number of clocks have passed, and the on-chip circuitry has completed its initialization. The relationship between reset, the oscillator, and the wakeup timer during the startup sequence are shown in Figure 3–9. See Figure 3–10 for a block diagram of the Reset logic.

The reset glitch filter allows the processor to ignore external reset pulses that are very short, and also determines the minimum duration of $\overline{\text{RESET}}$ that must be asserted in order to guarantee a chip reset. Once asserted, $\overline{\text{RESET}}$ pin can be deasserted only when crystal oscillator is fully running and an adequate signal is present on the XTAL1 pin of the microcontroller. Assuming that an external crystal is used in the crystal oscillator subsystem, after power on, the $\overline{\text{RESET}}$ pin should be asserted for 10 ms. For all subsequent resets, when the crystal oscillator is already running and a stable signal is on the XTAL1 pin, the $\overline{\text{RESET}}$ pin needs to be asserted for 300 ns only.

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



(1) Reset time: The time reset needs to be held LOW. This time depends on system parameters such as $V_{DD(1V8)}$, $V_{DD(3V3)}$ risetime, and the oscillator startup time. There are no restrictions from the microcontroller except that $V_{DD(1V8)}$, $V_{DD(3V3)}$, and the oscillator must be within the specific operating range.

(2) There are no sequencing requirements for $V_{DD(3V3)}$ and $V_{DD(1V8)}$.

(3) When $V_{DD(3V3)}$ and $V_{DD(1V8)}$ reach the minimum voltage, a reset is registered within two valid oscillator clocks.

(4) Typical startup time is 0.5 ms for a 12 MHz crystal.

Fig 9. Startup sequence diagram

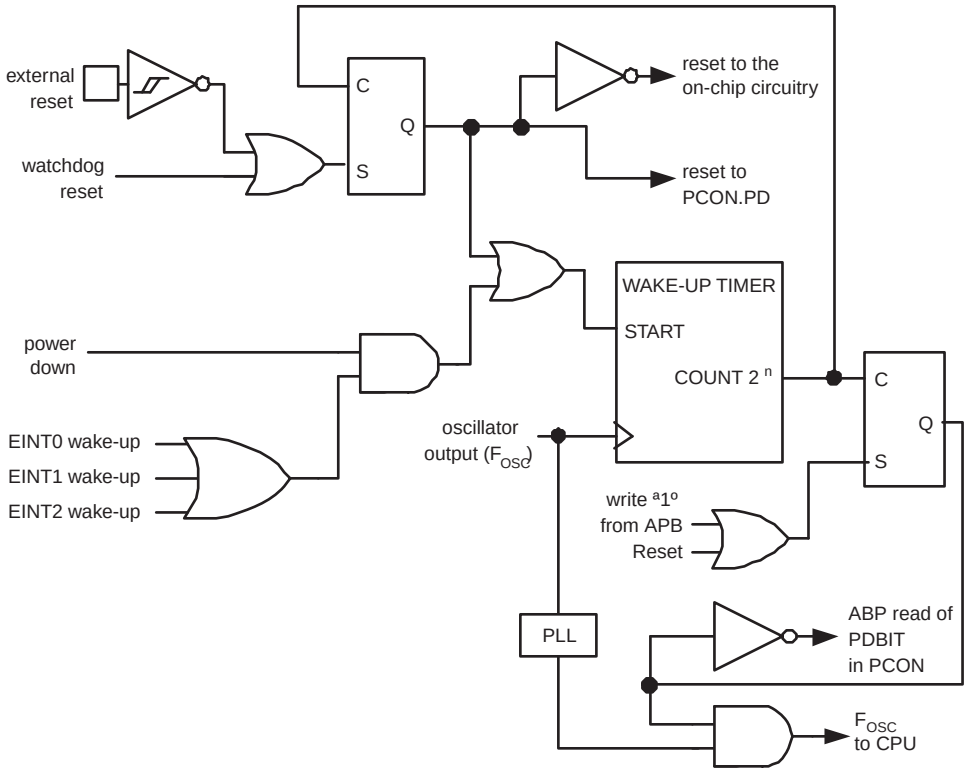


Fig 10. Reset block diagram including the wakeup timer

External and internal resets have some small differences. An external reset causes the value of certain pins to be latched to configure the part. External circuitry cannot determine when an internal reset occurs in order to allow setting up those special pins, so those latches are not reloaded during an internal reset. Pins that are examined during an external reset for various purposes are: P0.26/TRACESYNC and RTCK (see Section 6–3). Pin P0.14 (see Section 18–5) is examined by the on-chip bootloader when this code is executed after every reset.

12. APB divider

The APB Divider determines the relationship between the processor clock (CCLK) and the clock used by peripheral devices (PCLK). The APB Divider serves two purposes.

1. The first purpose is to provide peripherals with desired PCLK via APB bus so that they can operate at the speed chosen for the ARM processor. In order to achieve this, the APB bus may be slowed down to one half or one fourth of the processor clock rate. Because the APB bus must work properly at power up (and its timing cannot be altered if it does not work since the APB divider control registers reside on the APB bus), the default condition at reset is for the APB bus to run at one quarter speed.
2. The second purpose of the APB Divider is to allow power savings when an application does not require any peripherals to run at the full processor rate.

The connection of the APB Divider relative to the oscillator and the processor clock is shown in Figure 3–11. Because the APB Divider is connected to the PLL output, the PLL remains active (if it was running) during Idle mode.

12.1 Register description

Only one register is used to control the APB Divider.

Table 28. APB divider register map

Name	Description	Access	Reset value[1]	Address
APBDIV	Controls the rate of the APB clock in relation to the processor clock.	R/W	0x00	0xE01F C100

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

12.2 APB divider register (APBDIV - 0xE01F C100)

The APB Divider register contains two bits, allowing three divider values, as shown in Table 3–29.

Table 29. APB Divider register (APBDIV - address 0xE01F C100) bit description

Bit	Symbol	Value	Description	Reset value
1:0	APBDIV	00	APB bus clock is one fourth of the processor clock.	00
		01	APB bus clock is the same as the processor clock.	
		10	APB bus clock is one half of the processor clock.	
		11	Reserved. If this value is written to the APBDIV register, it has no effect (the previous setting is retained).	
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

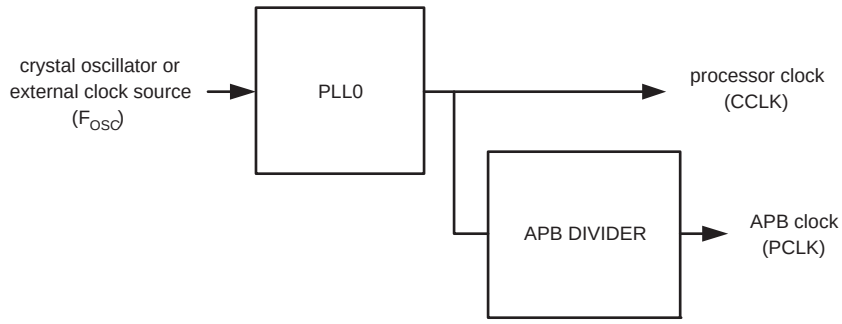


Fig 11. APB divider connections

13. Wakeup timer

On the LPC2104/05/06, the wakeup timer enforces a minimum reset duration based on the crystal oscillator and is activated whenever there is a wakeup from Power-down mode or any type of reset.

The purpose of the wakeup timer is to ensure that the oscillator and other analog functions required for chip operation are fully functional before the processor is allowed to execute instructions. This is important at power on, all types of reset, and whenever any of the aforementioned functions are turned off for any reason. Since the oscillator and other functions are turned off during Power-down mode, any wakeup of the processor from Power-down mode makes use of the wakeup timer.

The wakeup timer monitors the crystal oscillator to check whether it is safe to begin code execution. When power is applied to the chip, or some event caused the chip to exit Power-down mode, some time is required for the oscillator to produce a signal of sufficient amplitude to drive the clock logic. The amount of time depends on many factors, including the rate of V_{DD} ramp (in the case of power on), the type of crystal and its electrical characteristics (if a quartz crystal is used) as well as any other external circuitry (e.g. capacitors), and the characteristics of the oscillator itself under the existing ambient conditions.

Once a clock is detected, the wakeup timer counts 4096 clocks and then enables the flash memory to initialize. When the flash memory initialization is complete, the processor is released to execute instructions if the external reset has been deasserted. If an external clock source is used in the system (as opposed to a crystal connected to the oscillator pins), the possibility that there could be little or no delay for oscillator start-up must be considered. The wakeup timer design then ensures that any other required chip functions will be operational prior to the beginning of program execution.

Any of the various resets can bring the microcontroller out of power-down mode, as can the external interrupts EINT2:0. When one of these interrupts is enabled for wakeup and its selected event occurs, an oscillator wakeup cycle is started. The actual interrupt (if any) occurs after the wakeup timer expires and is handled by the Vectored Interrupt Controller.

The pin multiplexing on the LPC2104/05/06 (see Section 6–3) allows peripherals that share pins with external interrupts to, in effect, bring the device out of Power-down mode. The following pin-function pairings allow interrupts from events relating to UART0 or 1, SPI, or the I^2C: RXD0 / EINT0, SDA / EINT1, SSEL / EINT2, DCD1 / EINT1, RI1 / EINT2.

To put the device in Power-down mode and allow activity on one or more of these buses or lines to power it back up, software should reprogram the pin function to External Interrupt, select the appropriate mode and polarity for the Interrupt, and then select Power-down mode. Upon wakeup software should restore the pin multiplexing to the peripheral function.

14. Code security vs. debugging

Applications in development typically need the debugging and tracing facilities in the LPC2104/05/06. Later in the life cycle of an application, it may be more important to protect the application code from observation by hostile or competitive eyes. The Code Read Protection feature of the LPC2104/05/06 allows an application to control whether it can be debugged or protected from observation.

Details on the way Code Read Protection works can be found in Section 18–8 “Code Read Protection (CRP)”.

1. Introduction

The MAM block in the LPC2104/05/06 maximizes the performance of the ARM processor when it is running code in flash memory using a dual flash bank.

2. Operation

Simply put, the Memory Accelerator Module (MAM) attempts to have the next ARM instruction that will be needed in its latches in time to prevent CPU fetch stalls. The method used is to split the flash memory into two banks, each capable of independent accesses. Each of the two flash banks has its own prefetch buffer and branch trail buffer. The branch trail buffers for the two banks capture two 128-bit lines of flash data when an instruction fetch is not satisfied by either the prefetch buffer or branch trail buffer for its bank, and for which a prefetch has not been initiated. Each prefetch buffer captures one 128-bit line of instructions from its flash bank at the conclusion of a prefetch cycle initiated speculatively by the MAM.

Each 128 bit value includes four 32-bit ARM instructions or eight 16-bit Thumb instructions. During sequential code execution, typically one flash bank contains or is fetching the current instruction and the entire flash line that contains it. The other bank contains or is prefetching the next sequential code line. After a code line delivers its last instruction, the bank that contained it begins to fetch the next line in that bank.

Timing of flash read operations is programmable and is described in Section 4–8.

Branches and other program flow changes cause a break in the sequential flow of instruction fetches described above. When a backward branch occurs, there is a distinct possibility that a loop is being executed. In this case the branch trail buffers may already contain the target instruction. If so, execution continues without the need for a flash read cycle. For a forward branch, there is also a chance that the new address is already contained in one of the prefetch buffers. If it is, the branch is again taken with no delay. When a branch outside the contents of the branch trail and prefetch buffers is taken, one flash access cycle is needed to load the branch trail buffers. Subsequently, there will typically be no further fetch delays until another such “Instruction Miss” occurs.

The flash memory controller detects data accesses to the flash memory and uses a separate buffer to store the results in a manner similar to that used during code fetches. This allows faster access to data if it is accessed sequentially. A single line buffer is provided for data accesses, as opposed to the two buffers per flash bank that are provided for code accesses. There is no prefetch function for data accesses.

3. MAM blocks

The Memory Accelerator Module is divided into several functional blocks:

- A flash address latch for each bank: An incrementor function is associated with the bank 0 flash address latch.

- Two flash memory banks
- Instruction latches, data latches, address comparison latches
- Control and wait logic

Figure 4–12 shows a simplified block diagram of the Memory Accelerator Module data paths.

In the following descriptions, the term “fetch” applies to an explicit flash read request from the ARM. “Pre-fetch” is used to denote a flash read of instructions beyond the current processor fetch address.

3.1 Flash memory bank

There are two banks of flash memory in order to allow parallel access and eliminate delays for sequential access.

Flash programming operations are not controlled by the MAM but are handled as a separate function. A “boot block” sector contains flash programming algorithms that may be called as part of the application program and a loader that may be run to allow serial programming of the flash memory.

The flash memories are wired so that each sector exists in both banks and that a sector erase operation acts on part of both banks simultaneously. In effect, the existence of two banks is transparent to the programming functions.

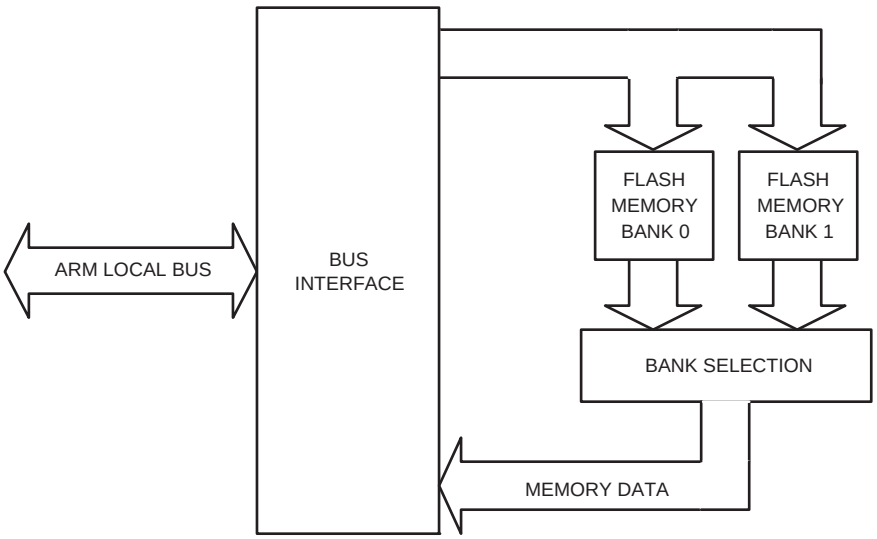


Fig 12. Simplified block diagram of the Memory Accelerator Module (MAM)

3.2 Instruction latches and data latches

Code and data accesses are treated separately by the Memory Accelerator Module.There are two sets of 128-bit instruction latches and 12-bit comparison address latches associated with each flash bank. One of the two sets, called the branch trail buffer, holds the data and comparison address for that bank from the last instruction miss. The other

set, called the prefetch buffer, holds the data and comparison address from prefetches undertaken speculatively by the MAM. Each instruction latch holds 4 words of code (4 ARM instructions, or 8 Thumb instructions).

Similarly, there is a 128-bit data latch and 13-bit data address latch, that are used during data cycles. This single set of latches is shared by both flash banks. Each data access that is not in the data latch causes a flash fetch of 4 words of data, which are captured in the data latch. This speeds up sequential data operations, but has little or no effect on random accesses.

3.3 Flash programming Issues

Since the flash memory does not allow access during programming and erase operations, it is necessary for the MAM to force the CPU to wait if a memory access to a flash address is requested while the flash module is busy. Under some conditions, this delay could result in a Watchdog time-out. The user will need to be aware of this possibility and take steps to insure that an unwanted Watchdog reset does not cause a system failure while programming or erasing the flash memory.

In order to preclude the possibility of stale data being read from the flash memory, the LPC2104/05/06 MAM holding latches are automatically invalidated at the beginning of any flash programming or erase operation. Any subsequent read from a flash address will cause a new fetch to be initiated after the flash operation has completed.

4. MAM operating modes

Three modes of operation are defined for the MAM, trading off performance for ease of predictability:

Mode 0: MAM off. All memory requests result in a flash read operation (see Table note 4–2). There are no instruction prefetches.

Mode 1: MAM partially enabled. Sequential instruction accesses are fulfilled from the holding latches if the data is present. Instruction prefetch is enabled. Non-sequential instruction accesses initiate flash read operations (see Table note 4–2). This means that all branches cause memory fetches. All data operations cause a flash read because buffered data access timing is hard to predict and is very situation dependent.

Mode 2: MAM fully enabled. Any memory request (code or data) for a value that is contained in one of the corresponding holding latches is fulfilled from the latch. Instruction prefetch is enabled. Flash read operations are initiated for instruction prefetch and code or data values not available in the corresponding holding latches.

Table 30. MAM responses to program accesses of various types

Program Memory Request Type	MAM Mode		
	0	1	2
Sequential access, data in latches	Initiate Fetch[2]	Use Latched Data[1]	Use Latched Data[1]
Sequential access, data not in latches	Initiate Fetch	Initiate Fetch[1]	Initiate Fetch[1]
Non-sequential access, data in latches	Initiate Fetch[2]	Initiate Fetch[1][2]	Use Latched Data[1]
Non-sequential access, data not in latches	Initiate Fetch	Initiate Fetch[1]	Initiate Fetch[1]

[1] Instruction prefetch is enabled in modes 1 and 2.

[2] The MAM actually uses latched data if it is available, but mimics the timing of a flash read operation. This saves power while resulting in the same execution timing. The MAM can truly be turned off by setting the fetch timing value in MAMTIM to one clock.

Table 31. MAM responses to data accesses of various types

Data Memory Request Type	MAM Mode		
	0	1	2
Sequential access, data in latches	Initiate Fetch[1]	Initiate Fetch[1]	Use Latched Data
Sequential access, data not in latches	Initiate Fetch	Initiate Fetch	Initiate Fetch
Non-sequential access, data in latches	Initiate Fetch[1]	Initiate Fetch[1]	Use Latched Data
Non-sequential access, data not in latches	Initiate Fetch	Initiate Fetch	Initiate Fetch

[1] The MAM actually uses latched data if it is available, but mimics the timing of a flash read operation. This saves power while resulting in the same execution timing. The MAM can truly be turned off by setting the fetch timing value in MAMTIM to one clock.

5. MAM configuration

After reset the MAM defaults to the disabled state. Software can turn memory access acceleration on or off at any time. This allows most of an application to be run at the highest possible performance, while certain functions can be run at a somewhat slower but more predictable rate if more precise timing is required.

6. Register description

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

Table 32. Summary of MAM registers

Name	Description	Access	Reset value[1]	Address
MAMCR	Memory Accelerator Module Control Register. Determines the MAM functional mode, that is, to what extent the MAM performance enhancements are enabled. See Table 4–33.	R/W	0x0	0xE01F C000
MAMTIM	Memory Accelerator Module Timing control. Determines the number of clocks used for flash memory fetches (1 to 7 processor clocks).	R/W	0x07	0xE01F C004

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

7. MAM Control Register (MAMCR - 0xE01F C000)

Two configuration bits select the three MAM operating modes, as shown in Table 4–33. Following Reset, MAM functions are disabled. Changing the MAM operating mode causes the MAM to invalidate all of the holding latches, resulting in new reads of flash information as required.

Table 33. MAM Control Register (MAMCR - address 0xE01F C000) bit description

Bit	Symbol	Value	Description	Reset value
1:0	MAM_mode_control	00	MAM functions disabled	0
		01	MAM functions partially enabled	
		10	MAM functions fully enabled	
		11	Reserved. Not to be used in the application.	
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8. MAM Timing register (MAMTIM - 0xE01F C004)

The MAM Timing register determines how many CCLK cycles are used to access the flash memory. This allows tuning MAM timing to match the processor operating frequency. flash access times from 1 clock to 7 clocks are possible. Single clock flash accesses would essentially remove the MAM from timing calculations. In this case the MAM mode may be selected to optimize power usage.

Table 34. MAM Timing register (MAMTIM - address 0xE01F C004) bit description

Bit	Symbol	Value	Description	Reset value
2:0	MAM_fetch_cycle_timing	000	0 - Reserved.	07
		001	1 - MAM fetch cycles are 1 processor clock (CCLK) in duration	
		010	2 - MAM fetch cycles are 2 CCLKs in duration	
		011	3 - MAM fetch cycles are 3 CCLKs in duration	
		100	4 - MAM fetch cycles are 4 CCLKs in duration	
		101	5 - MAM fetch cycles are 5 CCLKs in duration	
		110	6 - MAM fetch cycles are 6 CCLKs in duration	
		111	7 - MAM fetch cycles are 7 CCLKs in duration	
			Warning: These bits set the duration of MAM flash fetch operations as listed here. Improper setting of this value may result in incorrect operation of the device.	
7:3	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

9. MAM usage notes

When changing MAM timing, the MAM must first be turned off by writing a zero to MAMCR. A new value may then be written to MAMTIM. Finally, the MAM may be turned on again by writing a value (1 or 2) corresponding to the desired operating mode to MAMCR.

For a system clock slower than 20 MHz, MAMTIM can be 001. For a system clock between 20 MHz and 40 MHz, flash access time is suggested to be 2 CCLKs, while in systems with a system clock faster than 40 MHz, 3 CCLKs are proposed. For system clocks of 60 MHz and above, 4CCLK's are needed.

Table 35. Suggestions for MAM timing selection

system clock	Number of MAM fetch cycles in MAMTIM
< 20 MHz	1 CCLK
20 MHz to 40 MHz	2 CCLK
40 MHz to 60 MHz	3 CCLK
>60 MHz	4 CCLK

1. How to read this chapter

The following interrupts are used on **LPC2104/01, LPC2105/01, and LPC2106/01 only** (see Table 5–53):

- For the SSP interface:
  TX FIFO at least half empty (TXRIS)
  Rx FIFO at least half full (RXRIS)
  Receive Timeout condition (RTRIS)
  Receive overrun (RORRIS)
- For the UART autobauding capability:
  Auto-Baud Time-Out (ABTO)
  End of Auto-Baud (ABEO)

2. Features

- ARM PrimeCell Vectored Interrupt Controller
- 32 interrupt request inputs
- 16 vectored IRQ interrupts
- 16 priority levels dynamically assigned to interrupt requests
- Software interrupt generation

3. Description

The Vectored Interrupt Controller (VIC) takes 32 interrupt request inputs and programmably assigns them into 3 categories, FIQ, vectored IRQ, and non-vectored IRQ. The programmable assignment scheme means that priorities of interrupts from the various peripherals can be dynamically assigned and adjusted.

Fast Interrupt reQuest (FIQ) requests have the highest priority. If more than one request is assigned to FIQ, the VIC ORs the requests to produce the FIQ signal to the ARM processor. The fastest possible FIQ latency is achieved when only one request is classified as FIQ because then the FIQ service routine can simply start dealing with that device. But if more than one request is assigned to the FIQ class, the FIQ service routine can read a word from the VIC that identifies which FIQ source(s) is (are) requesting an interrupt.

Vectored IRQs have the middle priority, but only 16 of the 32 requests can be assigned to this category. Any of the 32 requests can be assigned to any of the 16 vectored IRQ slots, among which slot 0 has the highest priority and slot 15 has the lowest.

Non-vectored IRQs have the lowest priority.

The VIC ORs the requests from all the vectored and non-vectored IRQs to produce the IRQ signal to the ARM processor. The IRQ service routine can start by reading a register from the VIC and jumping there. If any of the vectored IRQs are requesting, the VIC provides the address of the highest-priority requesting IRQs service routine, otherwise it provides the address of a default routine that is shared by all the non-vectored IRQs. The default routine can read another VIC register to see what IRQs are active.

All registers in the VIC are word registers. Byte and halfword reads and write are not supported.

Additional information on the Vectored Interrupt Controller is available in the ARMPrimeCell Vectored Interrupt Controller (PL190) documentation.

4. Register description

The VIC implements the registers shown in Table 5–36. More detailed descriptions follow.

Table 36. VIC register map

Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	0	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	0	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICVectAddr	Vector Address Register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.	R/W	0	0xFFFF F030

Table 36. VIC register map

Name	Description	Access	Reset value[1]	Address
VICDefVectAddr	Default Vector Address Register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-15 hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.	R/W	0	0xFFFF F100
VICVectAddr1	Vector address 1 register.	R/W	0	0xFFFF F104
VICVectAddr2	Vector address 2 register.	R/W	0	0xFFFF F108
VICVectAddr3	Vector address 3 register.	R/W	0	0xFFFF F10C
VICVectAddr4	Vector address 4 register.	R/W	0	0xFFFF F110
VICVectAddr5	Vector address 5 register.	R/W	0	0xFFFF F114
VICVectAddr6	Vector address 6 register.	R/W	0	0xFFFF F118
VICVectAddr7	Vector address 7 register.	R/W	0	0xFFFF F11C
VICVectAddr8	Vector address 8 register.	R/W	0	0xFFFF F120
VICVectAddr9	Vector address 9 register.	R/W	0	0xFFFF F124
VICVectAddr10	Vector address 10 register.	R/W	0	0xFFFF F128
VICVectAddr11	Vector address 11 register.	R/W	0	0xFFFF F12C
VICVectAddr12	Vector address 12 register.	R/W	0	0xFFFF F130
VICVectAddr13	Vector address 13 register.	R/W	0	0xFFFF F134
VICVectAddr14	Vector address 14 register.	R/W	0	0xFFFF F138
VICVectAddr15	Vector address 15 register.	R/W	0	0xFFFF F13C
VICVectCntl0	Vector control 0 register. Vector Control Registers 0-15 each control one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest.	R/W	0	0xFFFF F200
VICVectCntl1	Vector control 1 register.	R/W	0	0xFFFF F204
VICVectCntl2	Vector control 2 register.	R/W	0	0xFFFF F208
VICVectCntl3	Vector control 3 register.	R/W	0	0xFFFF F20C
VICVectCntl4	Vector control 4 register.	R/W	0	0xFFFF F210
VICVectCntl5	Vector control 5 register.	R/W	0	0xFFFF F214
VICVectCntl6	Vector control 6 register.	R/W	0	0xFFFF F218
VICVectCntl7	Vector control 7 register.	R/W	0	0xFFFF F21C
VICVectCntl8	Vector control 8 register.	R/W	0	0xFFFF F220
VICVectCntl9	Vector control 9 register.	R/W	0	0xFFFF F224
VICVectCntl10	Vector control 10 register.	R/W	0	0xFFFF F228
VICVectCntl11	Vector control 11 register.	R/W	0	0xFFFF F22C
VICVectCntl12	Vector control 12 register.	R/W	0	0xFFFF F230
VICVectCntl13	Vector control 13 register.	R/W	0	0xFFFF F234
VICVectCntl14	Vector control 14 register.	R/W	0	0xFFFF F238
VICVectCntl15	Vector control 15 register.	R/W	0	0xFFFF F23C

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

5. VIC registers

The following section describes the VIC registers in the order in which they are used in the VIC logic, from those closest to the interrupt request inputs to those most abstracted for use by software. For most people, this is also the best order to read about the registers when learning the VIC.

5.1 Software Interrupt register (VICSoftInt - 0xFFFF F018)

The contents of this register are ORed with the 32 interrupt requests from the various peripherals, before any other logic is applied.

Table 37. Software Interrupt Register (VICSoftInt - address 0xFFFF F018) bit allocation

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	-	-	-	EINT2
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	-	SPI/SSP	I2C	PW\M0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 38. Software Interrupt Register (VICSoftInt - address 0xFFFF F018) bit description

Bit	Symbol	Reset value	Value	Description
31-0	See VICSoftInt bit allocation table.	0	0	Do not force the interrupt request with this bit number. Writing zeroes to bits in VICSoftInt has no effect, see VICSoftIntClear (Section 5–5.2).
			1	Force the interrupt request with this bit number.

5.2 Software Interrupt Clear Register (VICSoftIntClear - 0xFFFF F01C)

This register allows software to clear one or more bits in the Software Interrupt register, without having to first read it.

Table 39. Software Interrupt Clear Register (VICSoftIntClear - 0xFFFF F01C)

VICSoftIntClear	Description	Reset Value
31:0	1: writing a 1 clears the corresponding bit in the Software Interrupt register, thus releasing the forcing of this request. 0: writing a 0 leaves the corresponding bit in VICSoftInt unchanged.	0

Table 40. Software Interrupt Clear Register (VICSoftIntClear - address 0xFFFF F01C) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	-	-	-	EINT2
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	-	SPI/SSP	I2C	PWM
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT
Access	WO	WO	WO	WO	WO	WO	WO	WO

Table 41. Software Interrupt Clear Register (VICSoftIntClear - address 0xFFFF F01C) bit description

Bit	Symbol	Reset value	Value	Description
31-0	See VICSoftIntClear bit allocation table.	0	0	Writing a 0 leaves the corresponding bit in VICSoftInt unchanged.
			1	Writing a 1 clears the corresponding bit in the Software Interrupt register, thus releasing the forcing of this request.

5.3 Raw Interrupt Status Register (VICRawIntr - 0xFFFF F008)

This is a read only register. This register reads out the state of the 32 interrupt requests and software interrupts, regardless of enabling or classification.

Table 42. Raw Interrupt Status Register (VICRawIntr - address 0xFFFF F008) bit description

VICRawIntr	Description	Reset value
31:0	1:The hardware or software interrupt request with this bit number is asserted. 0: Neither the hardware nor software interrupt request with this bit number is asserted.	0

5.4 Interrupt Enable Register (VICIntEnable - 0xFFFF F010)

This is a read/write accessible register. This register controls which of the 32 interrupt requests and software interrupts contribute to FIQ or IRQ.

Table 43. Interrupt Enable Register (VICINtEnable - address 0xFFFF F010) bit description

VICIntEnable	Description	Reset value
31:0	When this register is read, 1s indicate interrupt requests or software interrupts that are enabled to contribute to FIQ or IRQ. When this register is written, ones enable interrupt requests or software interrupts to contribute to FIQ or IRQ, zeroes have no effect. See Section 5–5.5 “Interrupt Enable Clear Register (VICIntEnClear - 0xFFFF F014)” on page 49 and Table 5–44 below for how to disable interrupts.	0

5.5 Interrupt Enable Clear Register (VICIntEnClear - 0xFFFF F014)

This is a write only register. This register allows software to clear one or more bits in the Interrupt Enable register (Section 5–5.4), without having to first read it.

Table 44. Software Interrupt Clear Register (VICIntEnClear - address 0xFFFF F014) bit description

VICIntEnClear	Description	Reset value
31:0	1: writing a 1 clears the corresponding bit in the Interrupt Enable register, thus disabling interrupts for this request. 0: writing a 0 leaves the corresponding bit in VICIntEnable unchanged.	0

5.6 Interrupt Select Register (VICIntSelect - 0xFFFF F00C)

This is a read/write accessible register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.

Table 45. Interrupt Select Register (VICIntSelect - address 0xFFFF F00C) bit description

VICIntSelect	Description	Reset value
31:0	1: the interrupt request with this bit number is assigned to the FIQ category. 0: the interrupt request with this bit number is assigned to the IRQ category.	0

5.7 IRQ Status Register (VICIRQStatus - 0xFFFF F000)

This is a read only register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ. It does not differentiate between vectored and non-vectored IRQs.

Table 46. IRQ Status Register (VICIRQStatus - address 0xFFFF F000) bit description

VICIRQStatus	Description	Reset value
31:0	1: the interrupt request with this bit number is enabled, classified as IRQ, and asserted.	0

5.8 FIQ Status Register (VICFIQStatus - 0xFFFF F004)

This is a read only register. This register reads out the state of those interrupt requests that are enabled and classified as FIQ. If more than one request is classified as FIQ, the FIQ service routine can read this register to see which request(s) is (are) active.

Table 47. FIQ Status Register (VICFIQStatus - address 0xFFFF F004) bit description

VICFIQStatus	Description	Reset value
31:0	1: the interrupt request with this bit number is enabled, classified as FIQ, and asserted.	0

5.9 Vector Control registers 0-15 (VICvectCntl0-15 - 0xFFFF F200-23C)

These are a read/write accessible registers. Each of these registers controls one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest. Note that disabling a vectored IRQ slot in one of the VICVectCntl registers does not disable the interrupt itself, the interrupt is simply changed to the non-vectored form.

Table 48. Vector Control registers (VICVectCntl0-15 - addresses 0xFFFF F200-23C) bit description

VICVectCntl0-15	Description	Reset value
4:0	The number of the interrupt request or software interrupt assigned to this vectored IRQ slot. As a matter of good programming practice, software should not assign the same interrupt number to more than one enabled vectored IRQ slot. But if this does occur, the lower numbered slot will be used when the interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
5	1: this vectored IRQ slot is enabled, and can produce a unique ISR address when its assigned interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
31:6	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.10 Vector Address registers 0-15 (VICVectAddr0-15 - 0xFFFF F100-13C)

These are a read/write accessible registers. These registers hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.

Table 49. Vector Address registers (VICVectAddr0-15 - addresses 0xFFFF F100-13C) bit description

VICVectAddr0-15	Description	Reset value
31:0	When one or more interrupt request or software interrupt is (are) enabled, classified as IRQ, asserted, and assigned to an enabled vectored IRQ slot, the value from this register for the highest-priority such slot will be provided when the IRQ service routine reads the Vector Address register -VICVectAddr (Section 5–5.10).	0

5.11 Default Vector Address register (VICDefVectAddr - 0xFFFF F034)

This is a read/write accessible register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.

Table 50. Default Vector Address register (VICDefVectAddr - address 0xFFFF F034) bit description

VICDefVectAddr	Description	Reset value
31:0	When an IRQ service routine reads the Vector Address register (VICVectAddr), and no IRQ slot responds as described above, this address is returned.	0

5.12 Vector Address register (VICVectAddr - 0xFFFF F030)

This is a read/write accessible register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.

Table 51. Vector Address register (VICVectAddr - address 0xFFFF F030) bit description

VICVectAddr	Description	Reset value
31:0	If any of the interrupt requests or software interrupts that are assigned to a vectored IRQ slot is (are) enabled, classified as IRQ, and asserted, reading from this register returns the address in the Vector Address Register for the highest-priority such slot (lowest-numbered) such slot. Otherwise it returns the address in the Default Vector Address Register. Writing to this register does not set the value for future reads from it. Rather, this register should be written near the end of an ISR, to update the priority hardware.	0

5.13 Protection Enable register (VICProtection - 0xFFFF F020)

This is a read/write accessible register. This one-bit register controls access to the VIC registers by software running in User mode.

Table 52. Protection Enable register (VICProtection - address 0xFFFF F020) bit description

VICProtection	Description	Reset value
0	1: the VIC registers can only be accessed in privileged mode. 0: VIC registers can be accessed in User or privileged mode.	0
31:1	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

6. Interrupt sources

Table 5–53 lists the interrupt sources for each peripheral function. Each peripheral device has one interrupt line connected to the Vectored Interrupt Controller, but may have several internal interrupt flags. Individual interrupt flags may also represent more than one interrupt source. See Table 5–53 for which flags are implemented for which parts.

Table 53. Connection of interrupt sources to the Vectored Interrupt Controller

Block	Flag(s)	VIC Hex	VIC Channel # and Mask
WDT	Watchdog Interrupt (WDINT)	0	0x0000 0001
-	Reserved for software interrupts only	1	0x0000 0002
ARM Core	Embedded ICE, DbgCommRx	2	0x0000 0004
ARM Core	Embedded ICE, DbgCommTX	3	0x0000 0008
TIMER0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 2 (CR0, CR1, CR2)	4	0x0000 0010
TIMER1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5	0x0000 0020
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Auto-Baud Time-Out (ABTO) End of Auto-Baud (ABEO)	6	0x0000 0040
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI) Auto-Baud Time-Out (ABTO) End of Auto-Baud (ABEO)	7	0x0000 0080
PWM	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8	0x0000 0100
I^2C	SI (state change)	9	0x0000 0200
SPI/SSP	**Source: SPI** SPI Interrupt Flag (SPIF) Mode Fault (MODF) **Source: SSP** TX FIFO at least half empty (TXRIS) Rx FIFO at least half full (RXRIS) Receive Timeout condition (RTRIS) Receive overrun (RORRIS)	10	0x0000 0400
-	reserved	11	0x0000 0800
PLL	PLL Lock (PLOCK)	12	0x0000 1000
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	13	0x0000 2000
System Control	External Interrupt 0 (EINT0)	14	0x0000 4000
	External Interrupt 1 (EINT1)	15	0x0000 8000
	External Interrupt 2 (EINT2)	16	0x0001 0000

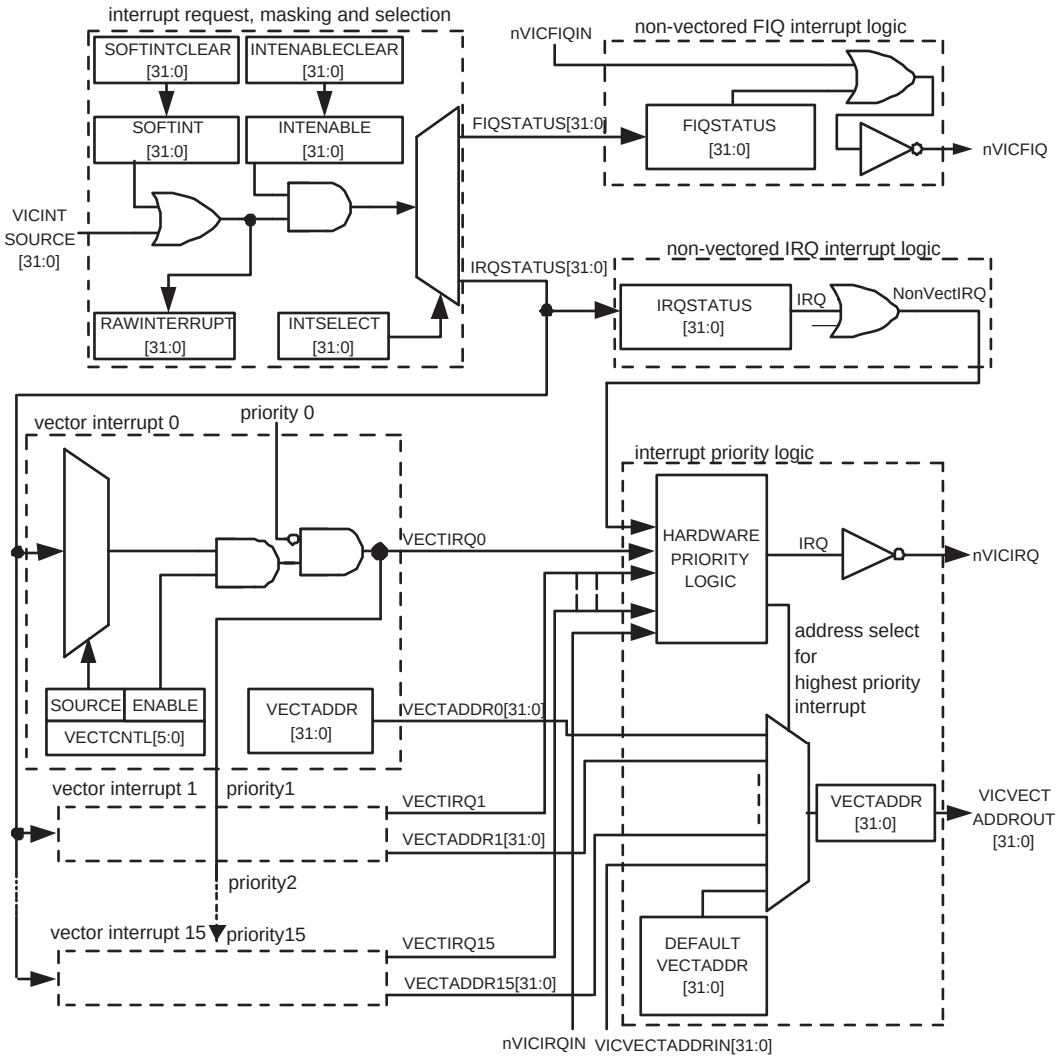


Fig 13. Block diagram of the Vectored Interrupt Controller

7. Spurious interrupts

Spurious interrupts are possible in the ARM7TDMI based microcontrollers such as the LPC2104/05/06 due to asynchronous interrupt handling. The asynchronous character of the interrupt processing has its roots in the interaction of the core and the VIC. If the VIC state is changed between the moments when the core detects an interrupt, and the core actually processes an interrupt, problems may be generated.

Real-life applications may experience the following scenarios:

1. VIC decides there is an IRQ interrupt and sends the IRQ signal to the core.
2. Core latches the IRQ state.
3. Processing continues for a few cycles due to pipelining.
4. Core loads IRQ address from VIC.

Furthermore, It is possible that the VIC state has changed during step 3. For example, VIC was modified so that the interrupt that triggered the sequence starting with step 1) is no longer pending -interrupt got disabled in the executed code. In this case, the VIC will not be able to clearly identify the interrupt that generated the interrupt request, and as a result the VIC will return the default interrupt VicDefVectAddr (0xFFFF F034).

This potentially disastrous chain of events can be prevented in two ways:

1. Application code should be set up in a way to prevent the spurious interrupts from occurring. Simple guarding of changes to the VIC may not be enough since, for example, glitches on level sensitive interrupts can also cause spurious interrupts.
2. VIC default handler should be set up and tested properly.

7.1 Details and case studies on spurious interrupts

This chapter contains details that can be obtained from the official ARM website , FAQ section under the "Technical Support":

What happens if an interrupt occurs as it is being disabled?

Applies to: ARM7TDMI

If an interrupt is received by the core during execution of an instruction that disables interrupts, the ARM7 family will still take the interrupt. This occurs for both IRQ and FIQ interrupts.

For example, consider the following instruction sequence:

```
MRS  r0, cpsr
ORR  r0, r0, #I_Bit:OR:F_Bit     ;disable IRQ and FIQ interrupts
MSR  cpsr_c, r0
```

If an IRQ interrupt is received during execution of the MSR instruction, then the behavior will be as follows:

- The IRQ interrupt is latched.
- The MSR cpsr, r0 executes to completion setting both the I bit and the F bit in the CPSR.
- The IRQ interrupt is taken because the core was committed to taking the interrupt exception before the I bit was set in the CPSR.
- The CPSR (with the I bit and F bit set) is moved to the SPSR_IRQ.

This means that, on entry to the IRQ interrupt service routine, you can see the unusual effect that an IRQ interrupt has just been taken while the I bit in the SPSR is set. In the example above, the F bit will also be set in both the CPSR and SPSR. This means that FIQs are disabled upon entry to the IRQ service routine, and will remain so until explicitly re-enabled. FIQs will not be reenabled automatically by the IRQ return sequence.

Although the example shows both IRQ and FIQ interrupts being disabled, similar behavior occurs when only one of the two interrupt types is being disabled. The fact that the core processes the IRQ after completion of the MSR instruction which disables IRQs does not normally cause a problem, since an interrupt arriving just one cycle earlier would be expected to be taken. When the interrupt routine returns with an instruction like:

```
SUBS  pc, lr, #4
```

The SPSR_IRQ is restored to the CPSR. The CPSR will now have the I bit and F bit set, and therefore execution will continue with all interrupts disabled. However, this can cause problems in the following cases:

Problem 1: A particular routine maybe called as an IRQ handler, or as a regular subroutine. In the latter case, the system guarantees that IRQs would have been disabled prior to the routine being called. The routine exploits this restriction to determine how it was called (by examining the I bit of the SPSR), and returns using the appropriate instruction. If the routine is entered due to an IRQ being received during execution of the MSR instruction which disables IRQs, then the I bit in the SPSR will be set. The routine would therefore assume that it could not have been entered via an IRQ.

Problem 2: FIQs and IRQs are both disabled by the same write to the CPSR. In this case, if an IRQ is received during the CPSR write, FIQs will be disabled for the execution time of the IRQ handler. This may not be acceptable in a system where FIQs must not be disabled for more than a few cycles.

7.1.1 Workaround

There are 3 suggested workarounds. Which of these is most applicable will depend upon the requirements of the particular system.

7.1.1.1 Solution 1: Test for an IRQ received during a write to disable IRQs

Add code similar to the following at the start of the interrupt routine.

```
SUB      lr, lr, #4       ; Adjust LR to point to return
STMFD    sp!, {..., lr}   ; Get some free regs
MRS      lr, SPSR         ; See if we got an interrupt while
TST      lr, #I_Bit       ; interrupts were disabled.
LDMNEFD  sp!, {..., pc}^  ; If so, just return immediately.
                          ; The interrupt will remain pending since we haven't
                          ; acknowledged it and will be reissued when interrupts
                          ; are next enabled.
                          ; Rest of interrupt routine
```

This code will test for the situation where the IRQ was received during a write to disable IRQs. If this is the case, the code returns immediately - resulting in the IRQ not being acknowledged (cleared), and further IRQs being disabled.

Similar code may also be applied to the FIQ handler, in order to resolve the first issue.

This is the recommended workaround, as it overcomes both problems mentioned above. However, in the case of problem two, it does add several cycles to the maximum length of time FIQs will be disabled.

7.1.1.2 Solution 2: Disable IRQs and FIQs using separate writes to the CPSR

```
MRS  r0, cpsr
ORR  r0, r0, #I_Bit   ;disable IRQs
MSR  cpsr_c, r0
ORR  r0, r0, #F_Bit   ;disable FIQs
MSR  cpsr_c, r0
```

This is the best workaround where the maximum time for which FIQs are disabled is critical (it does not increase this time at all). However, it does not solve problem one, and requires extra instructions at every point where IRQs and FIQs are disabled together.

7.1.1.3 Solution 3: Re-enable FIQs at the beginning of the IRQ handler

As the required state of all bits in the c field of the CPSR are known, this can be most efficiently be achieved by writing an immediate value to CPSR_C, for example:

```
MSR  cpsr_c, #I_Bit:OR:irq_MODE    ;IRQ should be disabled
                                   ;FIQ enabled
                                   ;ARM state, IRQ mode
```

This requires only the IRQ handler to be modified, and FIQs may be re-enabled more quickly than by using workaround 1. However, this should only be used if the system can guarantee that FIQs are never disabled while IRQs are enabled. It does not address problem one.

8. VIC usage notes

If user code is running from an on-chip RAM and an application uses interrupts, interrupt vectors must be re-mapped to on-chip address 0x0. This is necessary because all the exception vectors are located at addresses 0x0 and above. This is easily achieved by configuring the MEMMAP register (see Table 2–5) to User RAM mode. Application code should be linked such that at 0x4000 0000 the Interrupt Vector Table (IVT) will reside.

Although multiple sources can be selected (VICIntSelect) to generate FIQ request, only one interrupt service routine should be dedicated to service all available/present FIQ request(s). Therefore, if more than one interrupt sources are classified as FIQ the FIQ interrupt service routine must read VICFIQStatus to decide based on this content what to do and how to process the interrupt request. However, it is recommended that only one interrupt source should be classified as FIQ. Classifying more than one interrupt sources as FIQ will increase the interrupt latency.

Following the completion of the desired interrupt service routine, clearing of the interrupt flag on the peripheral level will propagate to corresponding bits in VIC registers (VICRawIntr, VICFIQStatus and VICIRQStatus). Also, before the next interrupt can be serviced, it is necessary that write is performed into the VICVectAddr register before the return from interrupt is executed. This write will clear the respective interrupt flag in the internal interrupt priority hardware.

In order to disable the interrupt at the VIC you need to clear corresponding bit in the VICIntEnClr register, which in turn clears the related bit in the VICIntEnable register. This also applies to the VICSoftInt and VICSoftIntClear in which VICSoftIntClear will clear the respective bits in VICSoftInt. For example, if VICSoftInt = 0x0000 0005 and bit 0 has to be cleared, VICSoftIntClear = 0x0000 0001 will accomplish this. Before the new clear operation on the same bit in VICSoftInt using writing into VICSoftIntClear is performed in the future, VICSoftIntClear = 0x0000 0000 must be assigned. Therefore writing 1 to any bit in Clear register will have one-time-effect in the destination register.

If the watchdog is enabled for interrupt on underflow or invalid feed sequence only then there is no way of clearing the interrupt. The only way you could perform return from interrupt is by disabling the interrupt at the VIC (using VICIntEnClr).

Example: Assuming that UART0 and SPI0 are generating interrupt requests that are classified as vectored IRQs (UART0 being on the higher level than SPI0), while UART1 and I^2C are generating non-vectored IRQs, the following could be one possibility for VIC setup:

```
VICIntSelect = 0x0000 0000   ; SPI0, I2C, UART1 and UART0 are IRQ =>
                             ; bit10, bit9, bit7 and bit6=0
VICIntEnable = 0x0000 06C0   ; SPI0, I2C, UART1 and UART0 are enabled interrupts =>
                             ; bit10, bit9, bit 7 and bit6=1
VICDefVectAddr = 0x...       ; holds address at what routine for servicing
                             ; non-vectored IRQs (i.e. UART1 and I2C) starts
VICVectAddr0 = 0x...         ; holds address where UART0 IRQ service routine starts
VICVectAddr1 = 0x...         ; holds address where SPI0 IRQ service routine starts
VICVectCntl0 = 0x0000 0026   ; interrupt source with index 6 (UART0) is enabled as
                             ; the one with priority 0 (the highest)
VICVectCntl1 = 0x0000 002A   ; interrupt source with index 10 (SPI0) is enabled
                             ; as the one with priority 1
```

After any of IRQ requests (SPI0, I^2C, UART0 or UART1) is made, microcontroller will redirect code execution to the address specified at location 0x0000 0018. For vectored and non-vectored IRQ's the following instruction could be placed at 0x0000 0018:

```
LDR pc, [pc,#-0xFF0]
```

This instruction loads PC with the address that is present in VICVectAddr register.

In case UART0 request has been made, VICVectAddr will be identical to VICVectAddr0, while in case SPI0 request has been made value from VICVectAddr1 will be found here. If neither UART0 nor SPI0 have generated IRQ request but UART1 and/or I^2C were the reason, content of VICVectAddr will be identical to VICDefVectAddr.

1. How to read this chapter

The pin configuration is identical for all LPC2104/05/06 parts.

2. Pin configuration

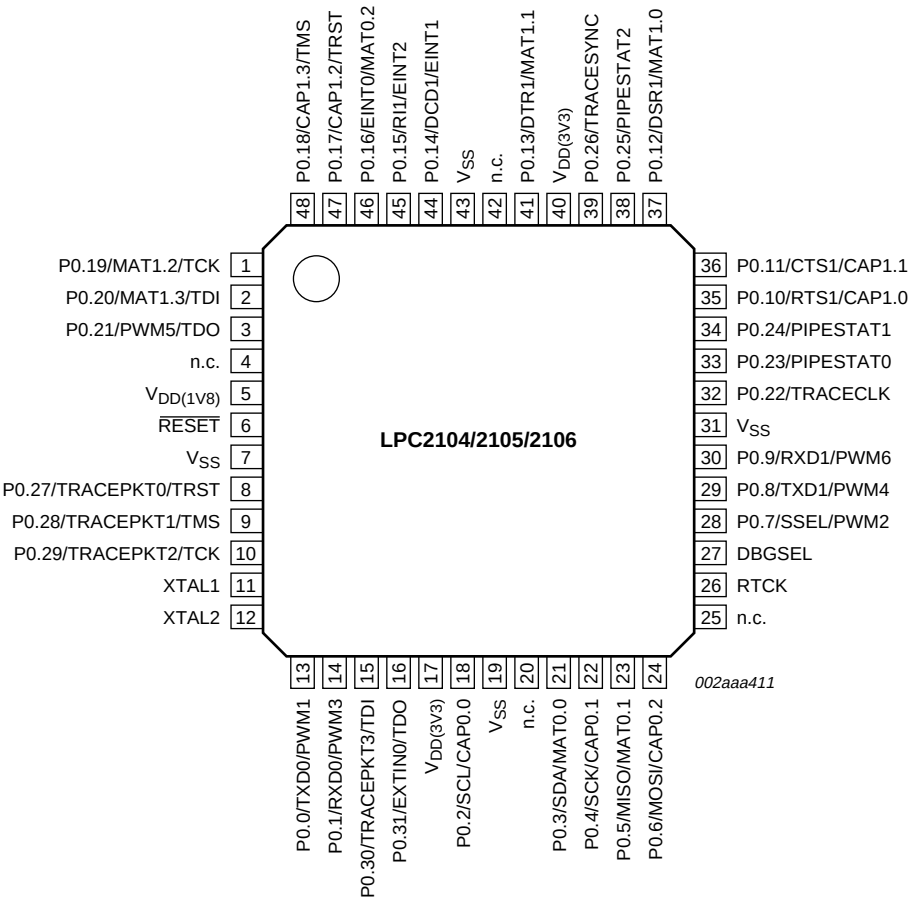


Fig 14. LPC2104/05/06 LQFP48 pin package

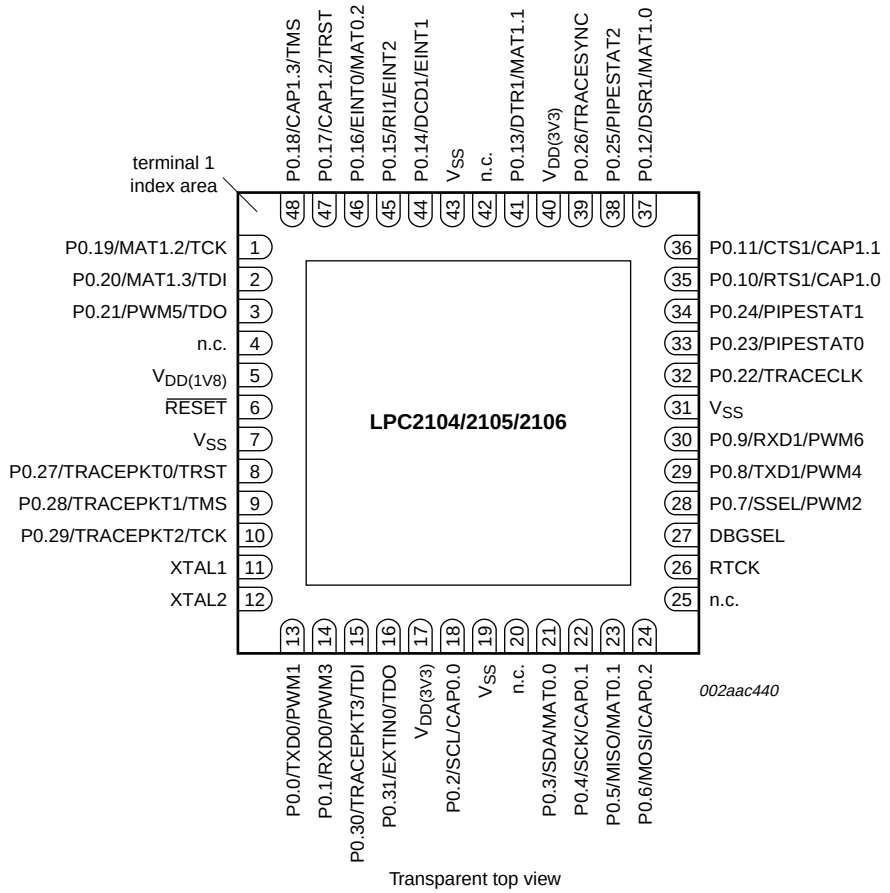


Fig 15. LPC2104/05/06 HVQFN48 pin package

3. LPC2104/05/06 pin description

Table 54. Pin description

Symbol	Pin	Type	Description
P0.0 to P0.31		I/O	**Port 0:** Port 0 is a 32-bit bidirectional I/O port with individual direction controls for each bit. The operation of port 0 pins depends upon the pin function selected via the Pin Connect Block.
P0.0/TXD0/PWM1	13[1]	I/O	**P0.0 —** Port 0 bit 0.
		O	**TXD0 —** Transmitter output for UART 0.
		O	**PWM1 —** Pulse Width Modulator output 1.
P0.1/RXD0/PWM3	14[1]	I/O	**P0.1 —** Port 0 bit 1.
		I	**RXD0 —** Receiver input for UART 0.
		O	**PWM3 —** Pulse Width Modulator output 3.
P0.2/SCL/CAP0.0	18[2]	I/O	**P0.2 —** Port 0 bit 2. The output is open-drain.
		I/O	**SCL —** I^2C-bus clock input/output. Open-drain output (for I^2C-bus compliance).
		I	**CAP0.0 —** Capture input for Timer 0, channel 0.

Table 54. Pin description *…continued*

Symbol	Pin	Type	Description
P0.3/SDA/MAT0.0	21[2]	I/O	**P0.3 —** Port 0 bit 3. The output is open-drain.
		I/O	**SDA —** I^2C-bus data input/output. Open-drain output (for I^2C-bus compliance).
		O	**MAT0.0 —** Match output for Timer 0, channel 0. The output is open-drain.
P0.4/SCK/CAP0.1	22[1]	I/O	**P0.4 —** Port 0 bit 4.
		I/O	**SCK —** Serial clock for SPI/SSP[3]. Clock output from master or input to slave.
		I	**CAP0.1 —** Capture input for Timer 0, channel 1.
P0.5/MISO/MAT0.1	23[1]	I/O	**P0.5 —** Port 0 bit 5.
		I/O	**MISO —** Master In Slave Out for SPI/SSP[3]. Data input to SPI/SSP master or data output from SPI/SSP slave.
		O	**MAT0.1 —** Match output for Timer 0, channel 1.
P0.6/MOSI/CAP0.2	24[1]	I/O	**P0.6 —** Port 0 bit 6.
		I/O	**MOSI —** Master Out Slave In for SPI/SSP[3]. Data output from SPI/SSP master or data input to SPI/SSP slave.
		I	**CAP0.2 —** Capture input for Timer 0, channel 2.
P0.7/SSEL/PWM2	28[1]	I/O	**P0.7 —** Port 0 bit 7.
		I	**SSEL —** Slave Select for SPI/SSP[3]. Selects the SPI/SSP interface as a slave.
		O	**PWM2 —** Pulse Width Modulator output 2.
P0.8/TXD1/PWM4	29[1]	I/O	**P0.8 —** Port 0 bit 8.
		O	**TXD1 —** Transmitter output for UART 1.
		O	**PWM4 —** Pulse Width Modulator output 4.
P0.9/RXD1/PWM6	30[1]	I/O	**P0.9 —** Port 0 bit 9.
		I	**RXD1 —** Receiver input for UART 1.
		O	**PWM6 —** Pulse Width Modulator output 6.
P0.10/RTS1/CAP1.0	35[1]	I/O	**P0.10 —** Port 0 bit 10.
		O	**RTS1 —** Request to Send output for UART 1.
		I	**CAP1.0 —** Capture input for Timer 1, channel 0.
P0.11/CTS1/CAP1.1	36[1]	I/O	**P0.11 —** Port 0 bit 11.
		I	**CTS1 —** Clear to Send input for UART 1.
		I	**CAP1.1 —** Capture input for Timer 1, channel 1.
P0.12/DSR1/MAT1.0	37[1]	I/O	**P0.12 —** Port 0 bit 12.
		I	**DSR1 —** Data Set Ready input for UART 1.
		O	**MAT1.0 —** Match output for Timer 1, channel 0.
P0.13/DTR1/MAT1.1	41[1]	I/O	**P0.13 —** Port 0 bit 13.
		O	**DTR1 —** Data Terminal Ready output for UART 1.
		O	**MAT1.1 —** Match output for Timer 1, channel 1.
P0.14/DCD1/EINT1	44[1]	I/O	**P0.14 —** Port 0 bit 14.
		I	**DCD1 —** Data Carrier Detect input for UART 1.
		I	**EINT1 —** External interrupt 1 input.
P0.15/RI1/EINT2	45[1]	I/O	**P0.15 —** Port 0 bit 15.
		I	**RI1 —** Ring Indicator input for UART 1.
		O	**EINT2 —** External interrupt 2 input.

Table 54. Pin description *…continued*

Symbol	Pin	Type	Description
P0.16/EINT0/MAT0.2	46[1]	I/O	**P0.16 —** Port 0 bit 16.
		I	**EINT0 —** External interrupt 0 input.
		O	**MAT0.2 —** Match output for Timer 0, channel 2.
P0.17/CAP1.2/TRST	47[1]	I/O	**P0.17 —** Port 0 bit 17.
		I	**CAP1.2 —** Capture input for Timer 1, channel 2.
		I	**TRST —** Test Reset for JTAG interface, primary JTAG pin group.
P0.18/CAP1.3/TMS	48[1]	I/O	**P0.18 —** Port 0 bit 18.
		I	**CAP1.3 —** Capture input for Timer 1, channel 3.
		I	**TMS —** Test Mode Select for JTAG interface, primary JTAG pin group.
P0.19/MAT1.2/TCK	1[1]	I/O	**P0.19 —** Port 0 bit 19.
		O	**MAT1.2 —** Match output for Timer 1, channel 2.
		I	**TCK —** Test Clock for JTAG interface, primary JTAG pin group.
P0.20/MAT1.3/TDI	2[1]	I/O	**P0.20 —** Port 0 bit 20.
		O	**MAT1.3 —** Match output for Timer 1, channel 3.
		I	**TDI —** Test Data In for JTAG interface, primary JTAG pin group.
P0.21/PWM5/TDO	3[1]	I/O	**P0.21 —** Port 0 bit 21.
		O	**PWM5 —** Pulse Width Modulator output 5.
		O	**TDO —** Test Data Out for JTAG interface, primary JTAG pin group.
P0.22/TRACECLK	32[4]	I/O	**P0.22 —** Port 0 bit 22.
		O	**TRACECLK —** Trace Clock. Standard I/O port with internal pull-up.
P0.23/PIPESTAT0	33[4]	I/O	**P0.23 —** Port 0 bit 23.
		O	**PIPESTAT0 —** Pipeline Status, bit 0. Standard I/O port with internal pull-up.
P0.24/PIPESTAT1	34[4]	I/O	**P0.24 —** Port 0 bit 24.
		O	**PIPESTAT1 —** Pipeline Status, bit 1. Standard I/O port with internal pull-up.
P0.25/PIPESTAT2	38[4]	I/O	**P0.25 —** Port 0 bit 25.
		O	**PIPESTAT2 —** Pipeline Status, bit 2. Standard I/O port with internal pull-up.
P0.26/TRACESYNC	39[4]	I/O	**P0.26 —** Port 0 bit 26.
		O	**TRACESYNC —** Trace Synchronization Standard I/O port with internal pull-up.
P0.27/TRACEPKT0/ TRST	8[4]	I/O	**P0.27 —** Port 0 bit 27.
		O	**TRACEPKT0 —** Trace Packet, bit 0. Standard I/O port with internal pull-up.
		I	**TRST —** Test Reset for JTAG interface, secondary JTAG pin group.
P0.28/TRACEPKT1/ TMS	9[4]	I/O	**P0.28 —** Port 0 bit 28.
		O	**TRACEPKT1 —** Trace Packet, bit 1. Standard I/O port with internal pull-up.
		I	**TMS —** Test Mode Select for JTAG interface, secondary JTAG pin group.
P0.29/TRACEPKT2/ TCK	10[4]	I/O	**P0.29 —** Port 0 bit 29.
		O	**TRACEPKT2 —** Trace Packet, bit 2. Standard I/O port with internal pull-up.
		I	**TCK —** Test Clock for JTAG interface, secondary JTAG pin group. This clock must be slower than 1/6 of the CPU clock (CCLK) for the JTAG interface to operate.

Table 54. Pin description *…continued*

Symbol	Pin	Type	Description
P0.30/TRACEPKT3/ TDI	15[4]	I/O	**P0.30 —** Port 0 bit 30.
		O	**TRACEPKT3 —** Trace Packet, bit 3. Standard I/O port with internal pull-up.
		I	**TDI —** Test Data In for JTAG interface, secondary JTAG pin group.
P0.31/EXTIN0/TDO	16[4]	I/O	**P0.31 —** Port 0 bit 31.
		I	**EXTIN0 —** External Trigger Input. Standard I/O port with internal pull-up.
		O	**TDO —** Test Data out for JTAG interface, secondary JTAG pin group.
RTCK	26[4]	I/O	Returned Test Clock output: Extra signal added to the JTAG port. Assists debugger synchronization when processor frequency varies. Also used during debug mode entry to select primary or secondary JTAG pins with the 48-pin package. Bidirectional pin with internal pull-up.
DBGSEL	27	I	Debug Select: When LOW, the part operates normally. When HIGH, debug mode is entered. Input pin with internal pull-down.
$\overline{\text{RESET}}$	6[5]	I	external reset input; a LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. TTL with hysteresis, 5 V tolerant.
XTAL1	11	I	input to the oscillator circuit and internal clock generator circuits.
XTAL2	12	O	output from the oscillator amplifier.
V_{SS}	7, 19, 31, 43	I	ground: 0 V reference.
$V_{DD(1V8)}$	5	I	1.8 V core power supply; this is the power supply voltage for internal circuitry.
$V_{DD(3V3)}$	17, 40	I	3.3 V pad power supply; this is the power supply voltage for the I/O ports.
n.c.	4, 20, 25, 42	-	not connected; these pins are not connected in the 48 pin package.

[1] 5 V tolerant pad providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control.

[2] Open-drain 5 V tolerant digital I/O pad, compatible with I^2C-bus 400 kHz specification. It requires external pull-up to provide an output functionality. Open-drain functionality applies to all functions on this pin.

[3] SSP interface available on LPC2104/05/06/01 only.

[4] 5 V tolerant pad with built-in pull-up resistor providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control. The pull-up resistor's value ranges from 60 kΩ to 300 kΩ.

[5] 5 V tolerant pad providing digital input (with TTL levels and hysteresis) function only.

1. Description

The pin connect block allows selected pins of the microcontroller to have more than one function. Configuration registers control the multiplexers to allow connection between the pin and the on chip peripherals.

Peripherals should be connected to the appropriate pins prior to being activated, and prior to any related interrupt(s) being enabled. Activity of any enabled peripheral function that is not mapped to a related pin should be considered undefined.

Selection of a single function on a port pin completely excludes all other functions otherwise available on the same pin.

2. Register description

The Pin Control Module contains 2 registers as shown in Table 7–55 below.

Table 55. Pin connect block register map

Name	Description	Access	Reset value[1]	Address
PINSEL0	Pin function select register 0.	Read/Write	0x0000 0000	0xE002 C000
PINSEL1	Pin function select register 1.	Read/Write	0x0000 0000	0xE002 C004

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

2.1 Pin function select register 0 (PINSEL0 - 0xE002 C000)

The PINSEL0 register controls the functions of the pins as per the settings listed in Table 7–56. The direction control bit in the IODIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically. Settings other than those shown in Table 7–56 are reserved, and should not be used

Table 56. Pin function select register 0 (PINSEL0 - 0xE002 C000)

PINSEL0	Pin name	Value		Function	Value after reset
1:0	P0.0	0	0	GPIO Port 0.0	0
		0	1	TXD (UART 0)	
		1	0	PWM1	
3:2	P0.1	0	0	GPIO Port 0.1	0
		0	1	RXD (UART 0)	
		1	0	PWM3	
5:4	P0.2	0	0	GPIO Port 0.2	0
		0	1	SCL (I^2C-bus)	
		1	0	Capture 0.0 (Timer 0)	

Table 56. Pin function select register 0 (PINSEL0 - 0xE002 C000) *...continued*

PINSEL0	Pin name	Value		Function	Value after reset
7:6	P0.3	0	0	GPIO Port 0.3	0
		0	1	SDA (I^2C-bus)	
		1	0	Match 0.0 (Timer 0)	
9:8	P0.4	0	0	GPIO Port 0.4	0
		0	1	SCK (SPI/SSP)	
		1	0	Capture 0.1 (Timer 0)	
11:10	P0.5	0	0	GPIO Port 0.5	0
		0	1	MISO (SPI/SSP)	
		1	0	Match 0.1 (Timer 0)	
13:12	P0.6	0	0	GPIO Port 0.6	0
		0	1	MOSI (SPI/SSP)	
		1	0	Capture 0.2 (Timer 0)	
15:14	P0.7	0	0	GPIO Port 0.7	0
		0	1	SSEL (SPI/SSP)	
		1	0	PWM2	
17:16	P0.8	0	0	GPIO Port 0.8	0
		0	1	TXD UART 1	
		1	0	PWM4	
19:18	P0.9	0	0	GPIO Port 0.9	0
		0	1	RXD (UART 1)	
		1	0	PWM6	
21:20	P0.10	0	0	GPIO Port 0.10	0
		0	1	RTS (UART1)	
		1	0	Capture 1.0 (Timer 1)	
23:22	P0.11	0	0	GPIO Port 0.11	0
		0	1	CTS (UART1)	
		1	0	Capture 1.1 (Timer 1)	
25:24	P0.12	0	0	GPIO Port 0.12	0
		0	1	DSR (UART1)	
		1	0	Match 1.0 (Timer 1)	
27:26	P0.13	0	0	GPIO Port 0.13	0
		0	1	DTR (UART 1)	
		1	0	Match 1.1 (Timer 1)	
29:28	P0.14	0	0	GPIO Port 0.14	0
		0	1	DCD (UART 1)	
		1	0	EINT1	
31:30	P0.15	0	0	GPIO Port 0.15	0
		0	1	RI (UART1)	
		1	0	EINT2	

2.2 Pin function select register 1 (PINSEL1 - 0xE002 C004)

The PINSEL1 register controls the functions of the pins as per the settings listed in Table 7–57. The direction control bit in the IODIR register is effective only when the GPIO function is selected for a pin. For other functions direction is controlled automatically. Function control for the pins P0[31:17] is effective only when the DBGSEL input is pulled LOW during reset.

Table 57. Pin function select register 1 (PINSEL1 - 0xE002 C004)

PINSEL1	Pin Name	Value		Function	Value after reset
1:0	P0.16	0	0	GPIO Port 0.16	0
		0	1	EINT0	
		1	0	Match 0.2 (Timer 0)	
3:2	P0.17	0	0	GPIO Port 0.17	0
		0	1	Capture 1.2 (Timer 1)	
5:4	P0.18	0	0	GPIO Port 0.18	0
		0	1	Capture 1.3 (Timer 1)	
7:6	P0.19	0	0	GPIO Port 0.19	0
		0	1	Match 1.2 (Timer 1)	
9:8	P0.20	0	0	GPIO Port 0.20	0
		0	1	Match 1.3 (Timer 1)	
11:10	P0.21	0	0	GPIO Port 0.21	0
		0	1	PWM5	
13:12	P0.22	0	0	GPIO Port 0.22	0
15:14	P0.23	0	0	GPIO Port 0.23	0
17:16	P0.24	0	0	GPIO Port 0.24	0
19:18	P0.25	0	0	GPIO Port 0.25	0
21:20	P0.26	0	0	GPIO Port 0.26	0
23:22	P0.27	0	0	GPIO Port 0.27	0
		0	1	TRST	
25:24	P0.28	0	0	GPIO Port 0.28	0
		0	1	TMS	
27:26	P0.29	0	0	GPIO Port 0.29	0
		0	1	TCK	
29:28	P0.30	0	0	GPIO Port 0.30	0
		0	1	TDI	
31:30	P0.31	0	0	GPIO Port 0.31	0
		0	1	TDO	

2.3 Pin function select register values

The PINSEL registers control the functions of device pins as shown below. Pairs of bits in these registers correspond to specific device pins.

Table 58. Pin function select register bits

PINSEL0 and PINSEL1 Values	Function	Value after Reset
00	Primary (default) function, typically GPIO port	00
01	First alternate function	
10	Second alternate function	
11	Third alternate function	

The direction control bit in the IO0DIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically. Each derivative typically has a different pinout and therefore a different set of functions possible for each pin. Details for a specific derivative may be found in the appropriate data sheet.

UM10275

Chapter 8: LPC2104/05/06 General Purpose Input/Output (GPIO)

Rev. 02 — 8 April 2009 User manual

1. How to read this chapter

Enhanced GPIO functions and all Fast GPIO registers (FIOxxx) are available on **LPC2104/01, LPC2105/01, and LPC2106/01 only**.

2. Basic configuration

GPIOs are configured using the following registers:

- Power: always enabled.
- Pins: select GPIO pins using PINSEL0/1 registers (see Table 7–55).
- Enable Fast GPIO (LPC2104/05/06/01 only): see Table 3–14.

3. Features

- Every physical GPIO port is accessible through two independent sets of registers. One set provides enhanced features and higher speed port access. The other set of registers is the legacy group of registers to ensure backward compatibilty to older NXP LPC2000 devices.
- Enhanced GPIO functions:
  - GPIO registers are relocated to the ARM local bus to achieve the fastest possible I/O timing.
  - Mask registers allow treating sets of port bits as a group, leaving other bits unchanged.
  - All registers are byte and half-word addressable.
  - Entire port value can be written in one instruction.
- Bit-level set and clear registers allow a single instruction set or clear of any number of bits in one port.
- Individual bits can be direction controlled.
- All I/O pins default to inputs after reset.
- Backward compatibility with other earlier devices is maintained with legacy registers appearing at the original addresses on the APB bus.

4. Applications

- General purpose I/O
- Driving LEDs or other indicators
- Controlling off-chip devices
- Sensing digital inputs

5. Pin description

Table 59. GPIO pin description

Pin	Type	Description
P0.0-P0.31	Input/ Output	General purpose input/output. The number of GPIOs actually available depends on the use of alternate functions.

6. Register description

LPC2104/05/06 has one 32-bit General Purpose I/O port. A total of 32 input/output pins are available on PORT0. PORT0 is controlled by the registers shown in Table 8–60 and Table 8–61.

Legacy registers shown in Table 8–60 allow backward compatibility with earlier family devices using existing code. The functions and relative timing of older GPIO implementations is preserved.

The registers in Table 8–61 represent the enhanced GPIO features available on the LPC2104/05/06. All of these registers are located directly on the local bus of the CPU for the fastest possible read and write timing and are byte, half-word, and word accessible. A mask register allows writing to individual pins of the GPIO port without the overhead of software masking.

The user must select in the System Control and Status flags register (SCS) whether a GPIO will be accessed via registers that provide enhanced features or a legacy set of registers (see Section 3–7.1 “System Control and Status flags register (SCS - 0xE01F C1A0)” on page 23). While both of a port’s fast and legacy GPIO registers are controlling the same physical pins, these two port control branches are mutually exclusive and operate independently. For example, changing a pin’s output via a fast register will not be observable via the corresponding legacy register.

The following text will refer to the legacy GPIO as "the slow" GPIO, while GPIO with the enhanced features selected will be referred to as "the fast" GPIO.

The "slow", legacy registers are word accessible only. The fast GPIO registers are byte, half-word, and word accessible. In the following two tables, bit 0 corresponds to port0.0, and bit 31 corresponds to port0.31.

Table 60. GPIO register map (legacy APB accessible registers)

Generic Name	Description	Access	Reset value[1]	PORT0 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction.	R/W	NA	0xE002 8000 IO0PIN
IOSET	GPIO Port Output Set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces HIGHs at the corresponding port pins. Writing zeroes has no effect.	R/W	0x0000 0000	0xE002 8004 IO0SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0x0000 0000	0xE002 8008 IO0DIR
IOCLR	GPIO Port Output Clear register. This register controls the state of output pins. Writing ones produces LOW at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	WO	0x0000 0000	0xE002 800C IO0CLR

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

Table 61. GPIO register map (local bus accessible registers - enhanced GPIO features)

Generic Name	Description	Access	Reset value[1]	PORT0 Address & Name
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0x0000 0000	0x3FFF C000 FIO0DIR
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN). Only the bits enabled by zeroes in this register are altered or returned. **Remark:** Bits in the FIOMASK register are active LOW.	R/W	0x0000 0000	0x3FFF C010 FIO0MASK

Table 61. GPIO register map (local bus accessible registers - enhanced GPIO features)

Generic Name	Description	Access	Reset value[1]	PORT0 Address & Name
FIOPIN	Fast GPIO Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins is not configured as an input to ADC). The value read is masked by ANDing with FIOMASK. Writing to this register places corresponding values in all bits enabled by zeroes in FIOMASK.	R/W	0x0000 0000	0x3FFF C014 FIO0PIN
FIOSET	Fast GPIO Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces HIGH at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by zeroes in FIOMASK can be altered.	R/W	0x0000 0000	0x3FFF C018 FIO0SET
FIOCLR	Fast GPIO Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces LOW at the corresponding port pins. Writing 0s has no effect. Only bits enabled by zeroes in FIOMASK can be altered.	WO	0x0000 0000	0x3FFF C01C FIO0CLR

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

6.1 GPIO port 0 Direction register (IODIR, Port 0: IO0DIR - 0xE002 8008; FIODIR, Port 0: FIO0DIR - 0x3FFF C000)

This word accessible register is used to control the direction of the pins when they are configured as GPIO port pins. Direction bit for any pin must be set according to the pin functionality.

IO0DIR is the legacy register while the enhanced GPIO functions are supported via the FIO0DIR register.

Table 62. GPIO port 0 Direction register (IO0DIR - address 0xE002 8008) bit description

Bit	Symbol	Value	Description	Reset value
31:0	P0xDIR		Slow GPIO Direction control bits. Bit 0 controls P0.0 ... bit 30 controls P0.30.	0x0000 0000
		0	Controlled pin is input.	
		1	Controlled pin is output.	

Table 63. Fast GPIO port 0 Direction register (FIO0DIR - address 0x3FFF C000) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FP0xDIR		Fast GPIO Direction control bits. Bit 0 in FIO0DIR controls P0.0 ... Bit 30 in FIO0DIR controls P0.30.	0x0000 0000
		0	Controlled pin is input.	
		1	Controlled pin is output.	

Aside from the 32-bit long and word only accessible FIODIR register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in Table 8–64. Next to providing the same functions as the FIODIR register, these additional registers allow easier and faster access to the physical port pins.

Table 64. Fast GPIO port 0 Direction control byte and half-word accessible register description

Register name	Register length (bits) & access	Address	Description	Reset value
FIO0DIR0	8 (byte)	0x3FFF C000	Fast GPIO Port 0 Direction control register 0. Bit 0 in FIO0DIR0 register corresponds to P0.0 ... bit 7 to P0.7.	0x00
FIO0DIR1	8 (byte)	0x3FFF C001	Fast GPIO Port 0 Direction control register 1. Bit 0 in FIO0DIR1 register corresponds to P0.8 ... bit 7 to P0.15.	0x00
FIO0DIR2	8 (byte)	0x3FFF C002	Fast GPIO Port 0 Direction control register 2. Bit 0 in FIO0DIR2 register corresponds to P0.16 ... bit 7 to P0.23.	0x00
FIO0DIR3	8 (byte)	0x3FFF C003	Fast GPIO Port 0 Direction control register 3. Bit 0 in FIO0DIR3 register corresponds to P0.24 ... bit 7 to P0.31.	0x00
FIO0DIRL	16 (half-word)	0x3FFF C000	Fast GPIO Port 0 Direction control Lower half-word register. Bit 0 in FIO0DIRL register corresponds to P0.0 ... bit 15 to P0.15.	0x0000
FIO0DIRU	16 (half-word)	0x3FFF C002	Fast GPIO Port 0 Direction control Upper half-word register. Bit 0 in FIO0DIRU register corresponds to P0.16 ... bit 15 to P0.31.	0x0000

6.2 Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)

This register is available in the enhanced group of registers only. It is used to select the port's pins that will and will not be affected by a write accesses to the FIOPIN, FIOSET or FIOCLR register. The mask register also filters out the port's content when the FIOPIN register is read.

A zero in this register's bit enables an access to the corresponding physical pin via a read or write access. If a bit in this register is one, the corresponding pin will not be changed with write access and if read, will not be reflected in the updated FIOPIN register. For software examples, see Section 8–7 "GPIO usage notes" on page 75

Table 65. Fast GPIO port 0 Mask register (FIO0MASK - address 0x3FFF C010) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FP0xMASK		Fast GPIO physical pin access control.	0x0000 0000
		0	Pin is affected by writes to the FIOSET, FIOCLR, and FIOPIN registers. Current state of the pin will be observable in the FIOPIN register.	
		1	Physical pin is unaffected by writes into the FIOSET, FIOCLR and FIOPIN registers. When the FIOPIN register is read, this bit will not be updated with the state of the physical pin.	

Aside from the 32-bit long and word only accessible FIOMASK register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in Table 8–66. Next to providing the same functions as the FIOMASK register, these additional registers allow easier and faster access to the physical port pins.

Table 66. Fast GPIO port 0 Mask byte and half-word accessible register description

Register name	Register length (bits) & access	Address	Description	Reset value
FIO0MASK0	8 (byte)	0x3FFF C010	Fast GPIO Port 0 Mask register 0. Bit 0 in FIO0MASK0 register corresponds to P0.0 ... bit 7 to P0.7.	0x00
FIO0MASK1	8 (byte)	0x3FFF C011	Fast GPIO Port 0 Mask register 1. Bit 0 in FIO0MASK1 register corresponds to P0.8 ... bit 7 to P0.15.	0x00
FIO0MASK2	8 (byte)	0x3FFF C012	Fast GPIO Port 0 Mask register 2. Bit 0 in FIO0MASK2 register corresponds to P0.16 ... bit 7 to P0.23.	0x00
FIO0MASK3	8 (byte)	0x3FFF C013	Fast GPIO Port 0 Mask register 3. Bit 0 in FIO0MASK3 register corresponds to P0.24 ... bit 7 to P0.31.	0x00
FIO0MASKL	16 (half-word)	0x3FFF C001	Fast GPIO Port 0 Mask Lower half-word register. Bit 0 in FIO0MASKL register corresponds to P0.0 ... bit 15 to P0.15.	0x0000
FIO0MASKU	16 (half-word)	0x3FFF C012	Fast GPIO Port 0 Mask Upper half-word register. Bit 0 in FIO0MASKU register corresponds to P0.16 ... bit 15 to P0.31.	0x0000

6.3 GPIO port 0 Pin value register (IOPIN, Port 0: IO0PIN - 0xE002 8000; FIOPIN, Port 0: FIO0PIN - 0x3FFF C014)

This register provides the value of port pins that are configured to perform only digital functions. The register will give the logic value of the pin regardless of whether the pin is configured for input or output, or as GPIO or an alternate digital function. As an example, a particular port pin may have GPIO input or GPIO output, UART receive, and PWM output as selectable functions. Any configuration of that pin will allow its current logic state to be read from the IOPIN register.

If a pin has an analog function as one of its options, the pin state cannot be read if the analog configuration is selected. Selecting the pin as an A/D input disconnects the digital features of the pin. In that case, the pin value read in the IOPIN register is not valid.

Writing to the IOPIN register stores the value in the port output register, bypassing the need to use both the IOSET and IOCLR registers to obtain the entire written value. This feature should be used carefully in an application since it affects the entire port.

The legacy register is the IO0PIN, while the enhanced GPIOs are supported via the FIO0PIN register. Access to a port pins via the FIOPIN register is conditioned by the corresponding FIOMASK register (see Section 8–6.2 “Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)”).

Only pins masked with zeros in the Mask register (see Section 8–6.2 “Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)”) will be correlated to the current content of the Fast GPIO port pin value register.

Table 67. GPIO port 0 Pin value register (IO0PIN - address 0xE002 8000) bit description

Bit	Symbol	Description	Reset value
31:0	P0xVAL	Slow GPIO pin value bits. Bit 0 in IO0PIN corresponds to P0.0 ... Bit 31 in IO0PIN corresponds to P0.31.	NA

Table 68. Fast GPIO port 0 Pin value register (FIO0PIN - address 0x3FFF C014) bit description

Bit	Symbol	Description	Reset value
31:0	FP0xVAL	Fast GPIO pin value bits. Bit 0 in FIO0PIN corresponds to P0.0 ... Bit 31 in FIO0PIN corresponds to P0.31.	NA

Aside from the 32-bit long and word only accessible FIOPIN register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in Table 8–69. Next to providing the same functions as the FIOPIN register, these additional registers allow easier and faster access to the physical port pins.

Table 69. Fast GPIO port 0 Pin value byte and half-word accessible register description

Register name	Register length (bits) & access	Address	Description	Reset value
FIO0PIN0	8 (byte)	0x3FFF C014	Fast GPIO Port 0 Pin value register 0. Bit 0 in FIO0PIN0 register corresponds to P0.0 ... bit 7 to P0.7.	0x00
FIO0PIN1	8 (byte)	0x3FFF C015	Fast GPIO Port 0 Pin value register 1. Bit 0 in FIO0PIN1 register corresponds to P0.8 ... bit 7 to P0.15.	0x00
FIO0PIN2	8 (byte)	0x3FFF C016	Fast GPIO Port 0 Pin value register 2. Bit 0 in FIO0PIN2 register corresponds to P0.16 ... bit 7 to P0.23.	0x00
FIO0PIN3	8 (byte)	0x3FFF C017	Fast GPIO Port 0 Pin value register 3. Bit 0 in FIO0PIN3 register corresponds to P0.24 ... bit 7 to P0.31.	0x00
FIO0PINL	16 (half-word)	0x3FFF C014	Fast GPIO Port 0 Pin value Lower half-word register. Bit 0 in FIO0PINL register corresponds to P0.0 ... bit 15 to P0.15.	0x0000
FIO0PINU	16 (half-word)	0x3FFF C016	Fast GPIO Port 0 Pin value Upper half-word register. Bit 0 in FIO0PINU register corresponds to P0.16 ... bit 15 to P0.31.	0x0000

6.4 GPIO port 0 output Set register (IOSET, Port 0: IO0SET - 0xE002 8004; FIOSET, Port 0: FIO0SET - 0x3FFF C018)

This register is used to produce a HIGH level output at the port pins configured as GPIO in an OUTPUT mode. Writing 1 produces a HIGH level at the corresponding port pins. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing 1 to the corresponding bit in the IOSET has no effect.

Reading the IOSET register returns the value of this register, as determined by previous writes to IOSET and IOCLR (or IOPIN as noted above). This value does not reflect the effect of any outside world influence on the I/O pins.

IO0SET is the legacy register while the enhanced GPIOs are supported via the FIO0SET register. Access to a port pins via the FIOSET register is conditioned by the corresponding FIOMASK register (see Section 8–6.2 “Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)”).

Table 70. GPIO port 0 output Set register (IO0SET - address 0xE002 8004 bit description

Bit	Symbol	Description	Reset value
31:0	P0xSET	Slow GPIO output value Set bits. Bit 0 in IO0SET corresponds to P0.0 ... Bit 31 in IO0SET corresponds to P0.31.	0x0000 0000

Table 71. Fast GPIO port 0 output Set register (FIO0SET - address 0x3FFF C018) bit description

Bit	Symbol	Description	Reset value
31:0	FP0xSET	Fast GPIO output value Set bits. Bit 0 in FIO0SET corresponds to P0.0 ... Bit 31 in FIO0SET corresponds to P0.31.	0x0000 0000

Aside from the 32-bit long and word only accessible FIOSET register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in Table 8–72. Next to providing the same functions as the FIOSET register, these additional registers allow easier and faster access to the physical port pins.

Table 72. Fast GPIO port 0 output Set byte and half-word accessible register description

Register name	Register length (bits) & access	Address	Description	Reset value
FIO0SET0	8 (byte)	0x3FFF C018	Fast GPIO Port 0 output Set register 0. Bit 0 in FIO0SET0 register corresponds to P0.0 ... bit 7 to P0.7.	0x00
FIO0SET1	8 (byte)	0x3FFF C019	Fast GPIO Port 0 output Set register 1. Bit 0 in FIO0SET1 register corresponds to P0.8 ... bit 7 to P0.15.	0x00
FIO0SET2	8 (byte)	0x3FFF C01A	Fast GPIO Port 0 output Set register 2. Bit 0 in FIO0SET2 register corresponds to P0.16 ... bit 7 to P0.23.	0x00
FIO0SET3	8 (byte)	0x3FFF C01B	Fast GPIO Port 0 output Set register 3. Bit 0 in FIO0SET3 register corresponds to P0.24 ... bit 7 to P0.31.	0x00
FIO0SETL	16 (half-word)	0x3FFF C018	Fast GPIO Port 0 output Set Lower half-word register. Bit 0 in FIO0SETL register corresponds to P0.0 ... bit 15 to P0.15.	0x0000
FIO0SETU	16 (half-word)	0x3FFF C01A	Fast GPIO Port 0 output Set Upper half-word register. Bit 0 in FIO0SETU register corresponds to P0.16 ... bit 15 to P0.31.	0x0000

6.5 GPIO port 0 output Clear register (IOCLR, Port 0: IO0CLR - 0xE002 800C; FIOCLR, Port 0: FIO0CLR - 0x3FFF C01C)

This register is used to produce a LOW level output at port pins configured as GPIO in an OUTPUT mode. Writing 1 produces a LOW level at the corresponding port pin and clears the corresponding bit in the IOSET register. Writing 0 has no effect. If any pin is configured as an input or a secondary function, writing to IOCLR has no effect.

IO0CLR is the legacy register while the enhanced GPIOs are supported via the FIO0CLR register. Access to a port pins via the FIOCLR register is conditioned by the corresponding FIOMASK register (see Section 8–6.2 “Fast GPIO port 0 Mask register (FIOMASK, Port 0: FIO0MASK - 0x3FFF C010)”).

Table 73. GPIO port 0 output Clear register 0 (IO0CLR - address 0xE002 800C) bit description

Bit	Symbol	Description	Reset value
31:0	P0xCLR	Slow GPIO output value Clear bits. Bit 0 in IO0CLR corresponds to P0.0 ... Bit 31 in IO0CLR corresponds to P0.31.	0x0000 0000

Table 74. Fast GPIO port 0 output Clear register 0 (FIO0CLR - address 0x3FFF C01C) bit description

Bit	Symbol	Description	Reset value
31:0	FP0xCLR	Fast GPIO output value Clear bits. Bit 0 in FIO0CLR corresponds to P0.0 ... Bit 31 in FIO0CLR corresponds to P0.31.	0x0000 0000

Aside from the 32-bit long and word only accessible FIOCLR register, every fast GPIO port can also be controlled via several byte and half-word accessible registers listed in Table 8–75. Next to providing the same functions as the FIOCLR register, these additional registers allow easier and faster access to the physical port pins.

Table 75. Fast GPIO port 0 output Clear byte and half-word accessible register description

Register name	Register length (bits) & access	Address	Description	Reset value
FIO0CLR0	8 (byte)	0x3FFF C01C	Fast GPIO Port 0 output Clear register 0. Bit 0 in FIO0CLR0 register corresponds to P0.0 ... bit 7 to P0.7.	0x00
FIO0CLR1	8 (byte)	0x3FFF C01D	Fast GPIO Port 0 output Clear register 1. Bit 0 in FIO0CLR1 register corresponds to P0.8 ... bit 7 to P0.15.	0x00
FIO0CLR2	8 (byte)	0x3FFF C01E	Fast GPIO Port 0 output Clear register 2. Bit 0 in FIO0CLR2 register corresponds to P0.16 ... bit 7 to P0.23.	0x00
FIO0CLR3	8 (byte)	0x3FFF C01F	Fast GPIO Port 0 output Clear register 3. Bit 0 in FIO0CLR3 register corresponds to P0.24 ... bit 7 to P0.31.	0x00
FIO0CLRL	16 (half-word)	0x3FFF C01C	Fast GPIO Port 0 output Clear Lower half-word register. Bit 0 in FIO0CLRL register corresponds to P0.0 ... bit 15 to P0.15.	0x0000
FIO0CLRU	16 (half-word)	0x3FFF C01E	Fast GPIO Port 0 output Clear Upper half-word register. Bit 0 in FIO0SETU register corresponds to P0.16 ... bit 15 to P0.31.	0x0000

7. GPIO usage notes

7.1 Example 1: sequential accesses to IOSET and IOCLR affecting the same GPIO pin/bit

State of the output configured GPIO pin is determined by writes into the pin's port IOSET and IOCLR registers. Last of these accesses to the IOSET/IOCLR register will determine the final output of a pin.

In case of a code:

```
IO0DIR = 0x0000 0080 ;pin P0.7 configured as output
IO0CLR = 0x0000 0080 ;P0.7 goes LOW
IO0SET = 0x0000 0080 ;P0.7 goes HIGH
IO0CLR = 0x0000 0080 ;P0.7 goes LOW
```

pin P0.7 is configured as an output (write to IO0DIR register). After this, P0.7 output is set to LOW (first write to IO0CLR register). Short high pulse follows on P0.7 (write access to IO0SET), and the final write to IO0CLR register sets pin P0.7 back to LOW level.

7.2 Example 2: an immediate output of 0s and 1s on a GPIO port

Write access to port's IOSET followed by write to the IOCLR register results with pins outputting 0s being slightly later then pins outputting 1s. There are systems that can tolerate this delay of a valid output, but for some applications simultaneous output of a binary content (mixed 0s and 1s) within a group of pins on a single GPIO port is required. This can be accomplished by writing to the port's IOPIN register.

The following code will preserve existing output on PORT0 pins P0.[31:16] and P0.[7:0] and at the same time set P0.[15:8] to 0xA5, regardless of the previous value of pins P0.[15:8]:

```
IO0PIN = (IO0PIN && 0xFFFF00FF) || 0x0000A500
```

The same outcome can be obtained using the fast port access.

Solution 1: using 32-bit (word) accessible fast GPIO registers

```
FIO0MASK = 0xFFFF00FF;
FIO0PIN  = 0x0000A500;
```

Solution 2: using 16-bit (half-word) accessible fast GPIO registers

```
FIO0MASKL = 0x00FF;
FIO0PINL  = 0xA500;
```

Solution 3: using 8-bit (byte) accessible fast GPIO registers

```
FIO0PIN1  = 0xA5;
```

7.3 Writing to IOSET/IOCLR vs. IOPIN

Write to the IOSET/IOCLR register allows easy change of the port's selected output pin(s) to HIGH/LOW level at a time. Only pin/bit(s) in the IOSET/IOCLR written with 1 will be set to HIGH/LOW level, while those written as 0 will remain unaffected. However, by just writing to either IOSET or IOCLR register it is not possible to instantaneously output arbitrary binary data containing mixture of 0s and 1s on a GPIO port.

Write to the IOPIN register enables instantaneous output of a desired content on the parallel GPIO. Binary data written into the IOPIN register will affect all output configured pins of that parallel port: 0s in the IOPIN will produce LOW level pin outputs and 1s in IOPIN will produce HIGH level pin outputs. In order to change output of only a group of port's pins, application must logically AND readout from the IOPIN with mask containing 0s in bits corresponding to pins that will be changed, and 1s for all others. Finally, this result has to be logically ORred with the desired content and stored back into the IOPIN register. Example 2 from above illustrates output of 0xA5 on PORT0 pins 15 to 8 while preserving all other PORT0 output pins as they were before.

7.4 Output signal frequency considerations when using the legacy and enhanced GPIO registers

The enhanced features of the fast GPIO ports available on this microcontroller make GPIO pins more responsive to the code that has task of controlling them. In particular, software access to a GPIO pin is 3.5 times faster via the fast GPIO registers than it is when the legacy set of registers is used. As a result of the access speed increase, the

maximum output frequency of the digital pin is increased 3.5 times, too. This tremendous increase of the output frequency is not always that visible when a plain C code is used. To gain full benefit from the fast GPIO features, write the portion of the application handling the fast port output in assembly code and execute in the ARM mode.

The following example shows a code in which the pin control section is written in assembly language for ARM. First, port 0 is configured as slow port, and the program generates two pulses on P0.20. Then port 0 is configured as fast port, and two pulses are generated on P0.16. This illustrates the difference between the fast and slow GPIO port output capabilities. Once this code is compiled in the ARM mode, its execution from the on-chip Flash will yield the best results when the MAM module is configured as described in Section 4–9 “MAM usage notes” on page 42. Execution from the on-chip SRAM is independent from the MAM setup.

```
          /*set port 0 to slow GPIO */
     ldr  r0,=0xe01fc1a0  /*register address--SCS register*/
     mov  r1,#0x0         /*set bit 0 to 0*/
     str  r1,[r0]         /*enable slow port*/
     ldr  r1,=0xffffffff  /* */
     ldr  r0,=0xe0028008  /*register address--IODIR*/
     str  r1,[r0]         /*set port 0 to output*/
     ldr  r2,=0x00100000  /*select P0.20*/
     ldr  r0,=0xe0028004  /*register address--IOSET*/
     ldr  r1,=0xe002800C  /*register address--IOCLR*/
          /*generate 2 pulses using slow GPIO on P0.20*/
     str  r2,[r0]         /*HIGH*/
     str  r2,[r1]         /*LOW*/
     str  r2,[r0]         /*HIGH*/
     str  r2,[r1]         /*LOW*/
          /*set port 0 to fast GPIO */
     ldr  r0,=0xe01fc1a0  /*register address--enable fast port*/
     mov  r1,#0x1
     str  r1,[r0]         /*enable fast port0*/
     ldr  r1,=0xffffffff
     ldr  r0,=0x3fffc000  /*direction of fast port0*/
     str  r1,[r0]
     ldr  r0,=0x3fffc018  /*FIO0SET -- fast port0 register*/
     ldr  r1,=0x3fffc01c  /*FIO0CLR0 -- fast port0 register*/
     ldr  r2,=0x00001000  /*select fast port 0.12 for toggle*/
          /*generate 2 pulses on the fast port*/
     str  r2,[r0]
     str  r2,[r1]
     str  r2,[r0]
     str  r2,[r1]
loop:b    loop
```

Figure 8–16 illustrates the code from above executed from the LPC2104/05/06 Flash memory. The PLL generated F_{CCLK} =60 MHz out of external F_{OSC} = 12 MHz. The MAM was fully enabled with MEMCR = 2 and MEMTIM = 3, and APBDIV = 1 (PCLK = CCLK).

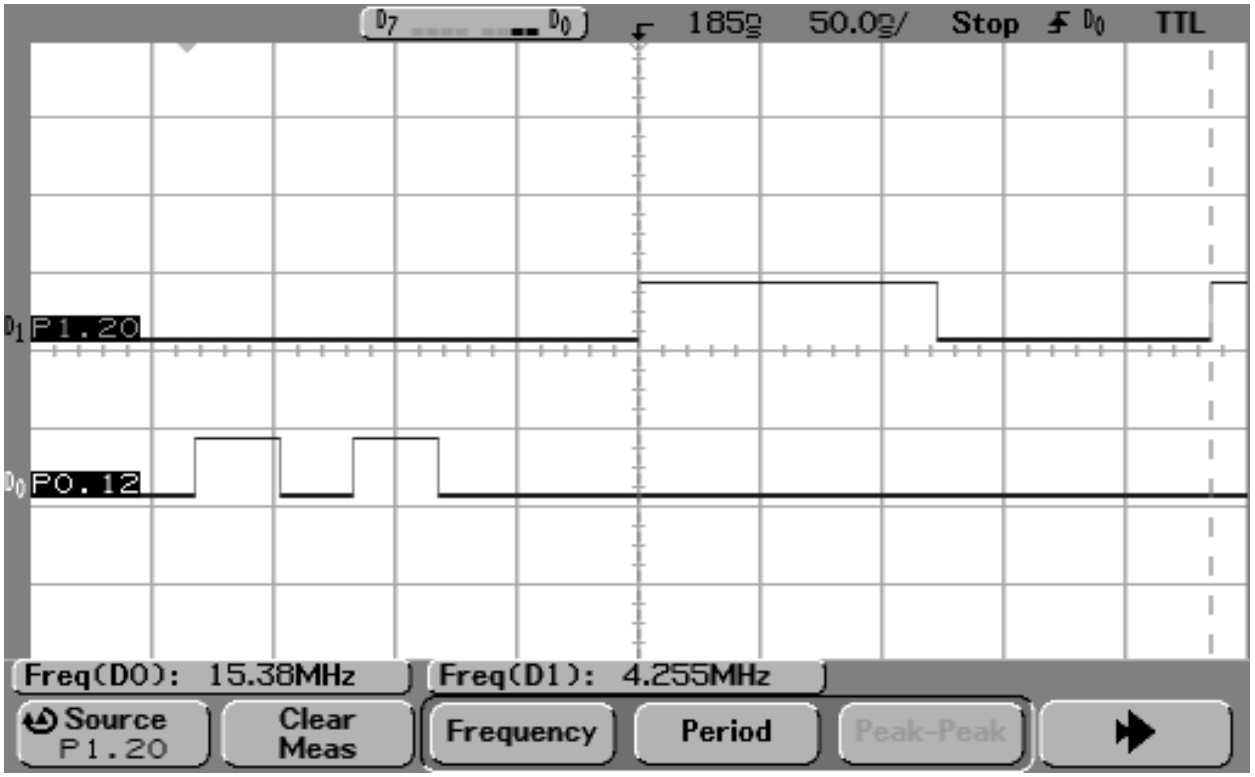


Fig 16. Illustration of the fast and slow GPIO access and output showing 3.5 x increase of the pin output frequency

UM10275

Chapter 9: LPC2104/05/06 Universal Asynchronous Receiver/Transmitter (UART) 0

Rev. 02 — 8 April 2009 User manual

1. How to read this chapter

The following features and registers are available in LPC2104/01, LPC2105/01, and LPC2106/01 only:

- Fractional baud rate controller: U0FDR (Table 9–82).
- Auto-baud control: U0ACR (Table 9–91) and U0IIR/U0IER bits 9:8 (Table 9–84 and Table 9–85).
- Software flow control: U0TER (Table 9–92).

2. Basic configuration

The UART0 peripheral is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCUART0.

   Remark: On reset, UART0 is enabled (PCUART0 = 1).
2. Baud rate: In register U0LCR (Table 9–88), set bit DLAB =1. This enables access to registers DLL (Table 9–80) and DLM (Table 9–81) for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register (Table 9–82).
3. UART FIFO: Use bit FIFO enable (bit 0) in register U0FCR (Table 9–87) to enable FIFO.
4. Pins: Select UART pins in registers PINSEL0/1 (see Section 7–2).
5. Interrupts: To enable UART interrupts set bit DLAB =0 in register U0LCR (Table 9–88). This enables access to U0IER (Table 9–84). Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

3. Features

- 16 byte Receive and Transmit FIFOs
- Register locations conforming to ‘550 industry standard
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes
- Built-in fractional baud rate generator with autobauding capabilities.
- Mechanism that enables software and hardware flow control implementation

4. Pin description

Table 76: UART0 pin description

Pin	Type	Description
RXD0	Input	**Serial Input.** Serial receive data.
TXD0	Output	**Serial Output.** Serial transmit data.

5. Register description

UART0 contains registers organized as shown in Table 9–77. The Divisor Latch Access Bit (DLAB) is contained in U0LCR[7] and enables access to the Divisor Latches.

The divisor latches are used to determine the baud rate for all UART transfers. When setting up the part, follow these steps:

1. Set DLAB = 1 in U0LCR (Section 9–5.8).
2. Set baud rate by writing values to registers DLL and DLM at address 0xE000 C000 Section 9–5.3).
3. Set DLAB = 0 in U0LCR (Section 9–5.8).
4. Read at address 0xE000 C000 accesses the U0RBR register (Section 9–5.1).
5. Write at address 0xE000 C000 accesses the U0THR register (Section 9–5.2).

Table 77. UART0 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U0RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE000 C000 (DLAB=0)
U0THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE000 C000 (DLAB=0)
U0DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE000 C000 (DLAB=1)
U0DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE000 C004 (DLAB=1)
U0IER	Interrupt Enable Register	-	-	-	-	-	-	En.ABTO	En.ABEO	R/W	0x00	0xE000 C004 (DLAB=0)
		-	-	-	-	-	En.RX Lin.St.Int	Enable THRE Int	En.RX Dat.Av.Int			
U0IIR	Interrupt ID Reg.	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE000 C008
		FIFOs Enabled		-	-	IIR3	IIR2	IIR1	IIR0			
U0FCR	FIFO Control Register	RX Trigger		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE000 C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000 C014
U0SCR	Scratch Pad Reg.	8-bit Data								R/W	0x00	0xE000 C01C
U0ACR	Auto-baud Control Register	-	-	-	-	-	-	ABTO Int.Clr	ABEO Int.Clr	R/W	0x00	0xE000 C020
		-	-	-	-	-	Aut.Rstrt.	Mode	Start			
U0FDR	Fractional Divider Register	Reserved[31:8]									0x10	0xE000 C028
		MulVal					DivAddVal					
U0TER	TX. Enable Reg.	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE000 C030

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

5.1 UART0 Receiver Buffer register (U0RBR - 0xE000 C000, when DLAB = 0, Read Only)

The U0RBR is the top byte of the UART0 Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the “oldest” received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

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

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

Table 78: UART0 Receiver Buffer Register (U0RBR - address 0xE000 C000, when DLAB = 0, Read Only) bit description

Bit	Symbol	Description	Reset value
7:0	RBR	The UART0 Receiver Buffer Register contains the oldest received byte in the UART0 Rx FIFO.	undefined

5.2 UART0 Transmit Holding Register (U0THR - 0xE000 C000, when DLAB = 0, Write Only)

The U0THR is the top byte of the UART0 TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The LSB represents the first bit to transmit.

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

Table 79: UART0 Transmit Holding Register (U0THR - address 0xE000 C000, when DLAB = 0, Write Only) bit description

Bit	Symbol	Description	Reset value
7:0	THR	Writing to the UART0 Transmit Holding Register causes the data to be stored in the UART0 transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA

5.3 UART0 Divisor Latch registers (U0DLL - 0xE000 C000 and U0DLM - 0xE000 C004, when DLAB = 1)

The UART0 Divisor Latch is part of the UART0 Baud Rate Generator and holds the value used to divide the clock in order to produce the baud rate clock, which must be 16x the desired baud rate (Equation 9–1). The U0DLL and U0DLM registers together form a 16 bit divisor where U0DLL contains the lower 8 bits of the divisor and U0DLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART0 Divisor Latches.

(1)

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL)}$$

Details on how to select the right value for U0DLL and U0DLM if the part includes a fractional divider (see Section 9–1) can be found in Section 9–5.4.

Table 80: UART0 Divisor Latch LSB register (U0DLL - address 0xE000 C000, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLL	The UART0 Divisor Latch LSB Register, along with the U0DLM register, determines the baud rate of the UART0.	0x01

Table 81: UART0 Divisor Latch MSB register (U0DLM - address 0xE000 C004, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLM	The UART0 Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UART0.	0x00

5.4 UART0 Fractional Divider Register (U0FDR - 0xE000 C028)

The UART0 Fractional Divider Register (U0FDR) controls the clock pre-scaler for the baud rate generation and can be read and written at the user's discretion. This pre-scaler takes the APB clock and generates an output clock according to the specified fractional requirements.

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

Table 82: UARTn Fractional Divider Register (U0FDR - address 0xE000 C028, U2FDR - 0xE007 8028, U3FDR - 0xE007 C028) bit description

Bit	Function	Value	Description	Reset value
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
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
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of UART0 disabled making sure that UART0 is fully software and hardware compatible with UARTs not equipped with this feature.

The UART0 baudrate can be calculated as (n = 0):

(2)

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Where PCLK is the peripheral clock, U0DLM and U0DLL are the standard UART0 baud rate divider registers, and DIVADDVAL and MULVAL are UART0 fractional baudrate generator specific parameters.

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

1. 0 < MULVAL ≤ 15
2. 0 ≤ DIVADDVAL < 15
3. DIVADDVAL<MULVAL

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

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

5.4.1 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.

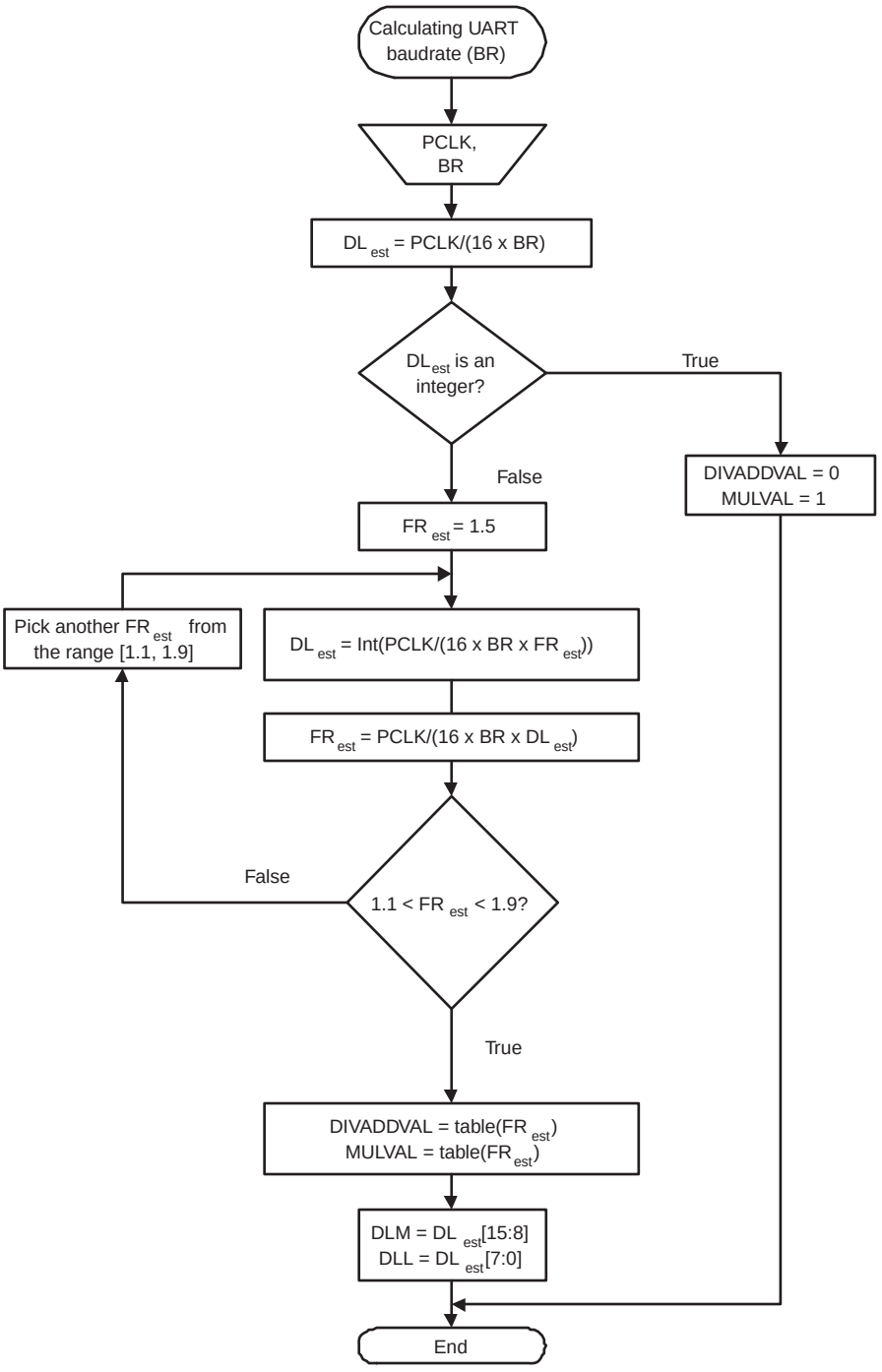


Fig 17. Algorithm for setting UART dividers

Table 83. 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

5.4.1.1 Example 1: PCLK = 14.7456 MHz, BR = 9600

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

5.4.1.2 Example 2: PCLK = 12 MHz, BR = 115200

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

The closest value for FRest = 1.628 in the look-up Table 9–83 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 9–2 the UART's baud rate is 115384. This rate has a relative error of 0.16% from the originally specified 115200.

5.5 UART0 Interrupt Enable Register (U0IER - 0xE000 C004, when DLAB = 0)

The U0IER is used to enable UART0 interrupt sources.

Table 84. UART0 Interrupt Enable Register (U0IER - address 0xE000 C004, when DLAB = 0) bit description

Bit	Symbol	Value	Description	Reset value
0	RBR Interrupt Enable		U0IER[0] enables the Receive Data Available interrupt for UART0. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	
1	THRE Interrupt Enable		U0IER[1] enables the THRE interrupt for UART0. The status of this can be read from U0LSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Status Interrupt Enable		U0IER[2] enables the UART0 RX line status interrupts. The status of this interrupt can be read from U0LSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
7:3	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	ABEOIntEn		Enables the end of auto-baud interrupt.	0
		0	Disable End of Auto-baud Interrupt.	
		1	Enable End of Auto-baud Interrupt.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable Auto-baud Time-out Interrupt.	
		1	Enable Auto-baud Time-out Interrupt.	
31:10	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.6 UART0 Interrupt Identification Register (U0IIR - 0xE000 C008, Read Only)

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

Table 85: UART0 Interrupt Identification Register (U0IIR - address 0xE000 C008, read only) bit description

Bit	Symbol	Value	Description	Reset value
0	Interrupt Pending		Note that U0IIR[0] is active LOW. The pending interrupt can be determined by evaluating U0IIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No pending interrupts.	

Table 85: UART0 Interrupt Identification Register (U0IIR - address 0xE000 C008, read only) bit description

Bit	Symbol	Value	Description	Reset value
3:1	Interrupt Identification		U0IER[3:1] identifies an interrupt corresponding to the UART0 Rx FIFO. All other combinations of U0IER[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	
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFO Enable		These bits are equivalent to U0FCR[0].	0
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Interrupts are handled as described in Table 9–86. Given the status of U0IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The U0IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UART0 RLS interrupt (U0IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART0 Rx input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART0 Rx error condition that set the interrupt can be observed via U0LSR[4:1]. The interrupt is cleared upon an U0LSR read.

The UART0 RDA interrupt (U0IIR[3:1] = 010) shares the second level priority with the CTI interrupt (U0IIR[3:1] = 110). The RDA is activated when the UART0 Rx FIFO reaches the trigger level defined in U0FCR[7:6] and is reset when the UART0 Rx FIFO depth falls below the trigger level. When the RDA interrupt goes active, the CPU can read a block of data defined by the trigger level.

The CTI interrupt (U0IIR[3:1] = 110) is a second level interrupt and is set when the UART0 Rx FIFO contains at least one character and no UART0 Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART0 Rx FIFO activity (read or write of UART0 RSR) will clear the interrupt. This interrupt is intended to flush the UART0 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 86: UART0 interrupt handling

U0IIR[3:0] value[1]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	None	None	-
0110	Highest	RX Line Status / Error	OE[2] or PE[2] or FE[2] or BI[2]	U0LSR Read[2]
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (U0FCR0=1)	U0RBR Read[3] or UART0 FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times). The exact time will be: [(word length) × 7 − 2] × 8 + [(trigger level − number of characters) × 8 + 1] RCLKs	U0RBR Read[3]
0010	Third	THRE	THRE[2]	U0IIR 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 9–5.9 “UART0 Line Status Register (U0LSR - 0xE000 C014, Read Only)”

[3] For details see Section 9–5.1 “UART0 Receiver Buffer register (U0RBR - 0xE000 C000, when DLAB = 0, Read Only)”

[4] For details see Section 9–5.6 “UART0 Interrupt Identification Register (U0IIR - 0xE000 C008, Read Only)” and Section 9–5.2 “UART0 Transmit Holding Register (U0THR - 0xE000 C000, when DLAB = 0, Write Only)”

The UART0 THRE interrupt (U0IIR[3:1] = 001) is a third level interrupt and is activated when the UART0 THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART0 THR FIFO a chance to fill up with data to eliminate many THRE interrupts from occurring at system start-up. The initialization conditions implement a one character delay minus the stop bit whenever THRE=1 and there have not been at least two characters in the U0THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to U0THR without a THRE interrupt to decode and service. A THRE interrupt is set immediately if the UART0 THR FIFO has held two or more characters at one time and currently, the U0THR is empty. The THRE interrupt is reset when a U0THR write occurs or a read of the U0IIR occurs and the THRE is the highest interrupt (U0IIR[3:1] = 001).

5.7 UART0 FIFO Control Register (U0FCR - 0xE000 C008)

The U0FCR controls the operation of the UART0 Rx and TX FIFOs.

Table 87: UART0 FIFO Control Register (U0FCR - address 0xE000 C008) bit description

Bit	Symbol	Value	Description	Reset value
0	FIFO Enable	0	UART0 FIFOs are disabled. Must not be used in the application.	0
		1	Active HIGH enable for both UART0 Rx and TX FIFOs and U0FCR[7:1] access. This bit must be set for proper UART0 operation. Any transition on this bit will automatically clear the UART0 FIFOs.	

Table 87: UART0 FIFO Control Register (U0FCR - address 0xE000 C008) bit description

Bit	Symbol	Value	Description	Reset value
1	RX FIFO Reset	0	No impact on either of UART0 FIFOs.	0
		1	Writing a logic 1 to U0FCR[1] will clear all bytes in UART0 Rx FIFO and reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UART0 FIFOs.	0
		1	Writing a logic 1 to U0FCR[2] will clear all bytes in UART0 TX FIFO and reset the pointer logic. This bit is self-clearing.	
5:3	-	0	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UART0 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).	

5.8 UART0 Line Control Register (U0LCR - 0xE000 C00C)

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

Table 88: UART0 Line Control Register (U0LCR - address 0xE000 C00C) bit description

Bit	Symbol	Value	Description	Reset value
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	
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if U0LCR[1:0]=00).	
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
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.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UART0 TXD is forced to logic 0 when U0LCR[6] is active HIGH.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	

5.9 UART0 Line Status Register (U0LSR - 0xE000 C014, Read Only)

The U0LSR is a read-only register that provides status information on the UART0 TX and RX blocks.

Table 89: UART0 Line Status Register (U0LSR - address 0xE000 C014, read only) bit description

Bit	Symbol	Value	Description	Reset value
0	Receiver Data Ready (RDR)		U0LSR0 is set when the U0RBR holds an unread character and is cleared when the UART0 RBR FIFO is empty.	0
		0	U0RBR is empty.	
		1	U0RBR contains valid data.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An U0LSR read clears U0LSR1. U0LSR1 is set when UART0 RSR has a new character assembled and the UART0 RBR FIFO is full. In this case, the UART0 RBR FIFO will not be overwritten and the character in the UART0 RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun 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 U0LSR read clears U0LSR[2]. Time of parity error detection is dependent on U0FCR[0]. **Note:** A parity error is associated with the character at the top of the UART0 RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An U0LSR read clears U0LSR[3]. The time of the framing error detection is dependent on U0FCR0. Upon detection of a framing error, the Rx will attempt to 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 UART0 RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXD0 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 RXD0 goes to marking state (all 1's). An U0LSR read clears this status bit. The time of break detection is dependent on U0FCR[0]. **Note:** The break interrupt is associated with the character at the top of the UART0 RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE))		THRE is set immediately upon detection of an empty UART0 THR and is cleared on a U0THR write.	1
		0	U0THR contains valid data.	
		1	U0THR is empty.	

Table 89: UART0 Line Status Register (U0LSR - address 0xE000 C014, read only) bit description

Bit	Symbol	Value	Description	Reset value
6	Transmitter Empty (TEMT)		TEMT is set when both U0THR and U0TSR are empty; TEMT is cleared when either the U0TSR or the U0THR contain valid data.	1
		0	U0THR and/or the U0TSR contains valid data.	
		1	U0THR and the U0TSR are empty.	
7	Error in RX FIFO (RXFE)		U0LSR[7] is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the U0RBR. This bit is cleared when the U0LSR register is read and there are no subsequent errors in the UART0 FIFO.	0
		0	U0RBR contains no UART0 RX errors or U0FCR[0]=0.	
		1	UART0 RBR contains at least one UART0 RX error.	

5.10 UART0 Scratch Pad Register (U0SCR - 0xE000 C01C)

The U0SCR has no effect on the UART0 operation. This register can be written and/or read at user's discretion. There is no provision in the interrupt interface that would indicate to the host that a read or write of the U0SCR has occurred.

Table 90: UART0 Scratch Pad Register (U0SCR - address 0xE000 C01C) bit description

Bit	Symbol	Description	Reset value
7:0	Pad	A readable, writable byte.	0x00

5.11 UART0 Auto-baud Control Register (U0ACR - 0xE000 C020)

The UART0 Auto-baud Control Register (U0ACR) controls the process of measuring the incoming clock/data rate for the baud rate generation and can be read and written at user's discretion.

Table 91: Auto-baud Control Register (U0ACR - 0xE000 C020) bit description

Bit	Symbol	Value	Description	Reset value
0	Start		This bit is automatically cleared after auto-baud completion.	0
		0	Auto-baud stop (auto-baud is not running).	
		1	Auto-baud start (auto-baud is running).Auto-baud run bit. This bit is automatically cleared after auto-baud completion.	
1	Mode		Auto-baud mode select bit.	0
		0	Mode 0.	
		1	Mode 1.	
2	AutoRestart	0	No restart	0
		1	Restart in case of time-out (counter restarts at next UART0 Rx falling edge)	
7:3	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

Table 91: Auto-baud Control Register (U0ACR - 0xE000 C020) bit description

Bit	Symbol	Value	Description	Reset value
8	ABEOIntClr		End of auto-baud interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the U0IIR. Writing a 0 has no impact.	0
9	ABTOIntClr		Auto-baud time-out interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the U0IIR. Writing a 0 has no impact.	0
31:10	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

5.11.1 Auto-baud

The UART0 auto-baud function can be used to measure the incoming baud-rate based on the "AT" protocol (Hayes command). If enabled the auto-baud feature will measure the bit time of the receive data stream and set the divisor latch registers U0DLM and U0DLL accordingly.

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

Two auto-baud measuring modes are available which can be selected by the U0ACR Mode bit. In mode 0 the baud-rate is measured on two subsequent falling edges of the UART0 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 UART0 Rx pin (the length of the start bit).

The U0ACR AutoRestart bit can be used to automatically restart baud-rate measurement if a time-out occurs (the rate measurement counter overflows). If this bit is set the rate measurement will restart at the next falling edge of the UART0 Rx pin.

The auto-baud function can generate two interrupts.

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

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

Typically the fractional baud-rate generator is disabled (DIVADDVAL = 0) during auto-baud. However, if the fractional baud-rate generator is enabled (DIVADDVAL > 0), it is going to impact the measuring of UART0 Rx pin baud-rate, but the value of the U0FDR register is not going to be modified after rate measurement. Also, when auto-baud is used, any write to U0DLM and U0DLL registers should be done before U0ACR register write. The minimum and the maximum baudrates supported by UART0 are function of PCLK, number of data bits, stop-bits and parity bits.

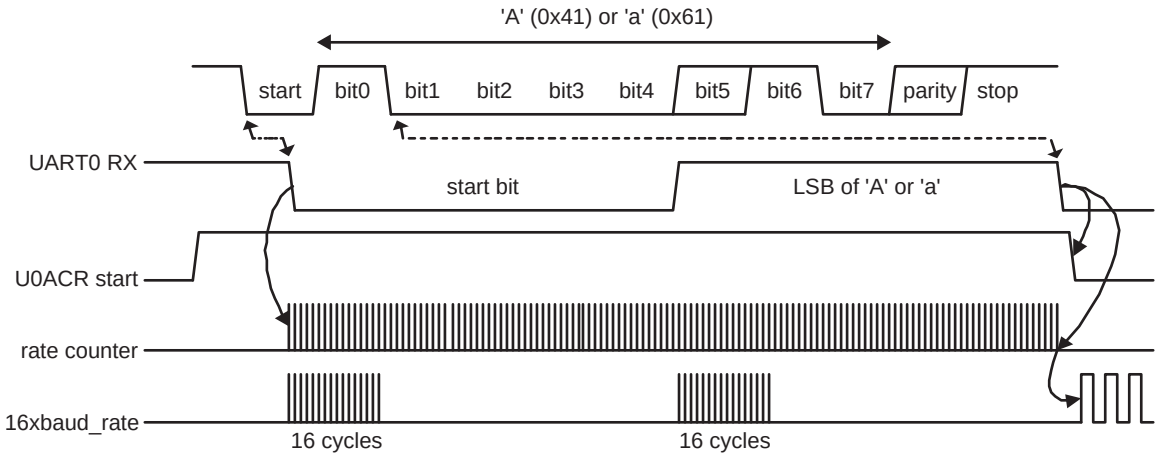
(3)

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART0_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

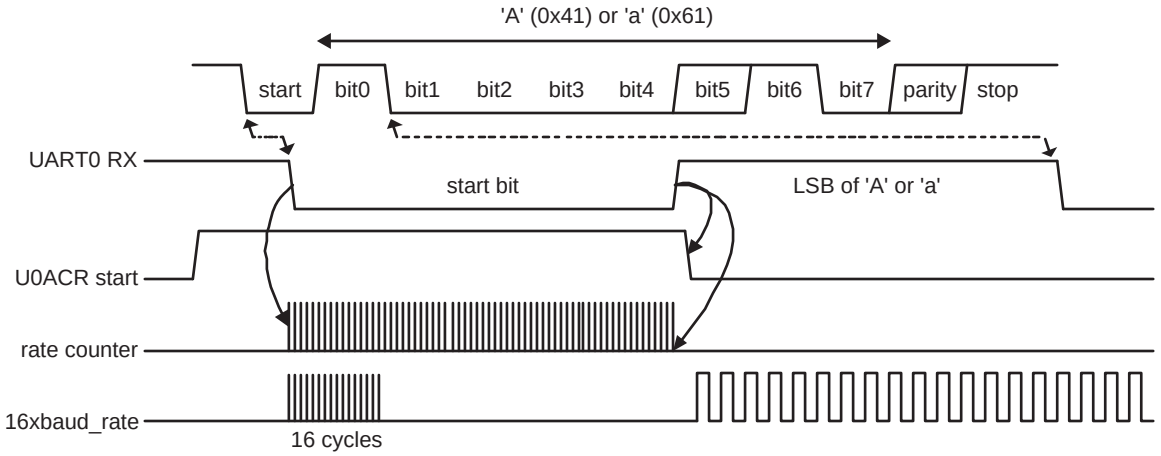
5.11.2 Auto-baud modes

When the software is expecting an ”AT" command, it configures the UART0 with the expected character format and sets the U0ACR Start bit. The initial values in the divisor latches U0DLM and U0DLM don‘t care. Because of the ”A" or ”a" ASCII coding (”A" = 0x41, ”a" = 0x61), the UART0 Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the U0ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On U0ACR Start bit setting, the baud-rate measurement counter is reset and the UART0 U0RSR is reset. The U0RSR baud rate is switch to the highest rate.
2. A falling edge on UART0 Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting PCLK cycles optionally pre-scaled by the fractional baud-rate generator.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the (fractional baud-rate pre-scaled) UART0 input clock, guaranteeing the start bit is stored in the U0RSR.
4. During the receipt of the start bit (and the character LSB for mode = 0) the rate counter will continue incrementing with the pre-scaled UART0 input clock (PCLK).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UART0 Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UART0 Rx pin.
6. The rate counter is loaded into U0DLM/U0DLL and the baud-rate will be switched to normal operation. After setting the U0DLM/U0DLL the end of auto-baud interrupt U0IIR ABEOInt will be set, if enabled. The U0RSR will now continue receiving the remaining bits of the ”A/a" character.



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



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

Fig 18. Autobaud a) mode 0 and b) mode 1 waveform.

5.12 UART0 Transmit Enable Register (U0TER - 0xE000 C030)

The U0TER enables implementation of software flow control. When TXEn=1, UART0 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART0 transmission will stop.

Table 9–92 describes how to use TXEn bit in order to achieve software flow control.

Table 92: UART0 Transmit Enable Register (U0TER - address 0xE000 C030) bit description

Bit	Symbol	Description	Reset value
6:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7	TXEN	When this bit is 1, as it is after a Reset, data written to the THR is output on the TXD pin as soon as any preceding data has been sent. If this bit 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. Architecture

The architecture of the UART0 is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART0.

The UART0 receiver block, U0RX, monitors the serial input line, RXD0, for valid input. The UART0 RX Shift Register (U0RSR) accepts valid characters via RXD0. After a valid character is assembled in the U0RSR, it is passed to the UART0 RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART0 transmitter block, U0TX, accepts data written by the CPU or host and buffers the data in the UART0 TX Holding Register FIFO (U0THR). The UART0 TX Shift Register (U0TSR) reads the data stored in the U0THR and assembles the data to transmit via the serial output pin, TXD0.

The UART0 Baud Rate Generator block, U0BRG, generates the timing enables used by the UART0 TX block. The U0BRG clock input source is the APB clock (PCLK). The main clock is divided down per the divisor specified in the U0DLL and U0DLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

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

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

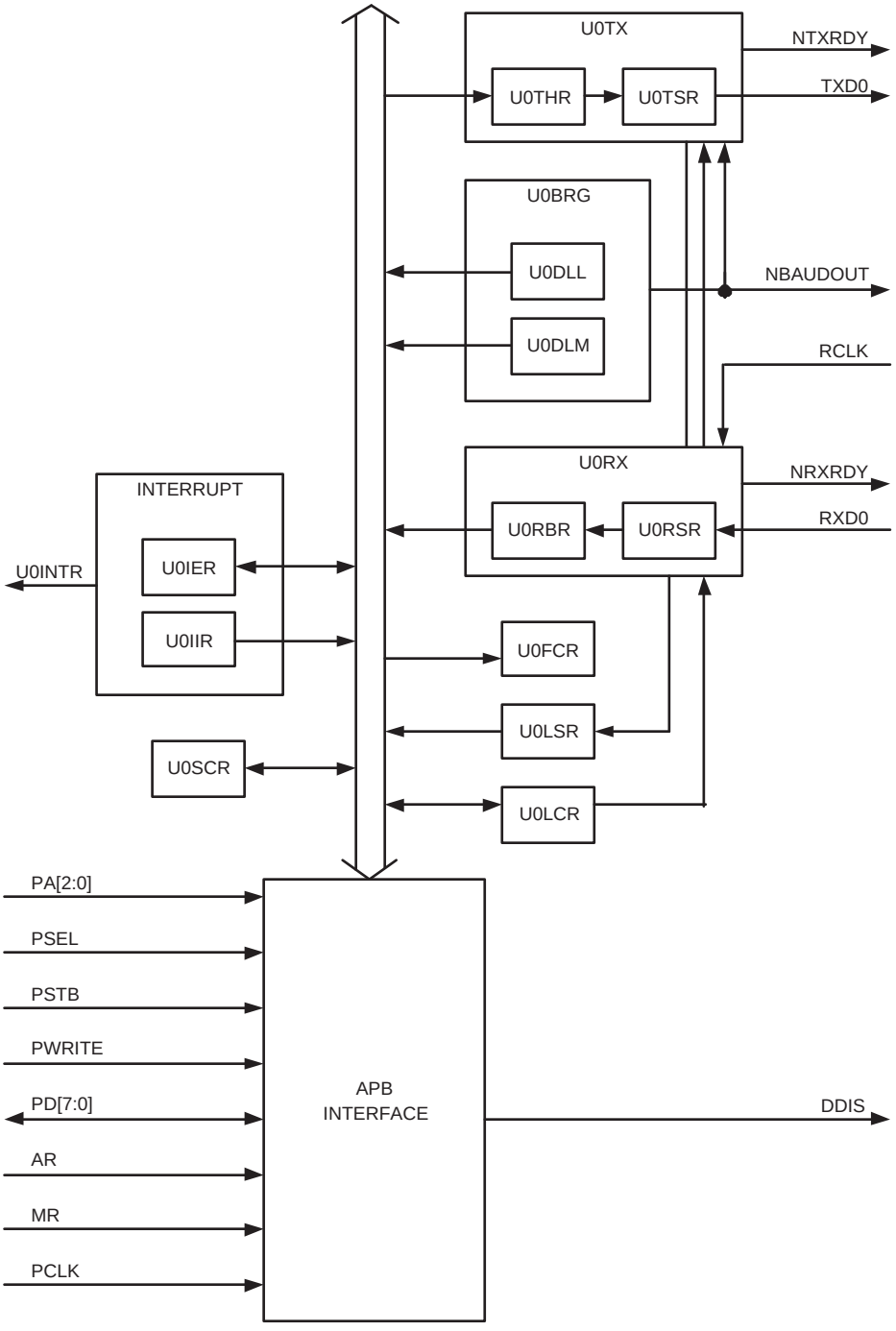


Fig 19. UART0 block diagram

UM10275

Chapter 10: LPC2104/05/06 Universal Asynchronous Receiver/Transmitter (UART) 1

Rev. 02 — 8 April 2009 User manual

1. How to read this chapter

The following features and registers are available in LPC2104/01, LPC2105/01, and LPC2106/01 only:

- Fractional baud rate controller: U1FDR (Table 10–99).
- Auto-baud control: U1ACR (Table 10–111) and U1IIR/U1IER bits 9:8 (Table 10–101 and Table 10–102).
- Software flow control: U1TER (Table 10–112).

2. Basic configuration

The UART0 peripheral is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCUART0.

   Remark: On reset, UART1 is enabled (PCUART1 = 1).
2. Baud rate: In register U1LCR (Table 10–105), set bit DLAB =1. This enables access to registers DLL (Table 10–97) and DLM (Table 10–98) for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register (Table 10–99).
3. UART FIFO: Use bit FIFO enable (bit 0) in register U1FCR (Table 10–104) to enable FIFO.
4. Pins: Select UART pins in registers PINSEL0/1 (see Section 7–2).
5. Interrupts: To enable UART interrupts set bit DLAB =0 in register U1LCR (Table 10–105). This enables access to U1IER (Table 10–101). Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

3. Features

- UART1 is identical to UART0 with the addition of a modem interface.
- UART1 contains 16 byte Receive and Transmit FIFOs.
- Register locations conform to ‘550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Fractional baud rate generator with autobauding capabilities is built-in.
- Mechanism enables software and hardware flow control implementation.
- Standard modem interface signals are included, and flow control (auto-CTS/RTS) is fully supported in hardware.

4. Pin description

Table 93. UART1 pin description

Pin	Type	Description
RXD1	Input	**Serial Input.** Serial receive data.
TXD1	Output	**Serial Output.** Serial transmit data.
CTS1	Input	**Clear To Send.** Active LOW signal indicates if the external modem is ready to accept transmitted data via TXD1 from the UART1. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[4]. State change information is stored in U1MSR[0] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).
DCD1	Input	**Data Carrier Detect.** Active LOW signal indicates if the external modem has established a communication link with the UART1 and data may be exchanged. In normal operation of the modem interface (U1MCR[4]=0), the complement value of this signal is stored in U1MSR[7]. State change information is stored in U1MSR3 and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).
DSR1	Input	**Data Set Ready.** Active LOW signal indicates if the external modem is ready to establish a communications link with the UART1. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[5]. State change information is stored in U1MSR[1] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).
DTR1	Output	Data Terminal Ready. Active LOW signal indicates that the UART1 is ready to establish connection with external modem. The complement value of this signal is stored in U1MCR[0].
RI1	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 (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[6]. State change information is stored in U1MSR[2] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).
RTS1	Output	**Request To Send.** Active LOW signal indicates that the UART1 would like to transmit data to the external modem. The complement value of this signal is stored in U1MCR[1].

5. Register description

UART1 contains registers organized as shown in Table 76. The Divisor Latch Access Bit (DLAB) is contained in U1LCR[7] and enables access to the Divisor Latches.

The divisor latches are used to determine the baud rate for all UART transfers. When setting up the part, follow these steps:

1. Set DLAB = 1 in U1LCR (Section 10–5.10).
2. Set baud rate by writing values to registers DLL and DLM at address 0xE000 C000 Section 10–5.3).
3. Set DLAB = 0 in U1LCR (Section 10–5.8).
4. Read at address 0xE000 C000 accesses the U1RBR register (Section 10–5.1).
5. Write at address 0xE000 C000 accesses the U1THR register (Section 10–5.2).

Table 94. UART1 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U1RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE001 0000 (DLAB=0)
U1DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE001 0000 (DLAB=1)
U1DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE001 0004 (DLAB=1)
U1IER	Interrupt Enable Register	-	-	-	-	-	-	En.ABTO	En.ABEO	R/W	0x00	0xE001 0004 (DLAB=0)
		En.CTS Int	-	-	-	E.Modem St.Int	En. RX Lin.St. Int	Enable THRE Int	En. RX Dat.Av.Int			
U1IIR	Interrupt ID Reg.	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE001 0008
		FIFOs Enabled		-	-	IIR3	IIR2	IIR1	IIR0			
U1FCR	FIFO Control Register	RX Trigger		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE001 000C
U1MCR	Modem Ctrl. Reg.	CTSen	RTSen	-	LoopBck.	-	-	RTS	DTR	R/W	0x00	0xE001 0010
U1LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE001 0014
U1MSR	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing Edge RI	Delta DSR	Delta CTS	RO	0x00	0xE001 0018
U1SCR	Scratch Pad Reg.	8-bit Data								R/W	0x00	0xE001 001C
U1ACR	Auto-baud Control Register	-	-	-	-	-	-	ABTO IntClr	ABEO IntClr	R/W	0x00	0xE001 0020
		-	-	-	-	-	Aut.Rstrt.	Mode	Start			
U1FDR	Fractional Divider Register	Reserved[31:8]								R/W	0x10	0xE001 0028
		MulVal					DivAddVal					
U1TER	TX. Enable Reg.	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE001 0030

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

5.1 UART1 Receiver Buffer Register (U1RBR - 0xE001 0000, when DLAB = 0 Read Only)

The U1RBR is the top byte of the UART1 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 U1LCR must be zero in order to access the U1RBR. The U1RBR 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 U1LSR register, and then to read a byte from the U1RBR.

Table 95. UART1 Receiver Buffer Register (U1RBR - address 0xE001 0000, when DLAB = 0 Read Only) bit description

Bit	Symbol	Description	Reset value
7:0	RBR	The UART1 Receiver Buffer Register contains the oldest received byte in the UART1 RX FIFO.	undefined

5.2 UART1 Transmitter Holding Register (U1THR - 0xE001 0000, when DLAB = 0 Write Only)

The U1THR is the top byte of the UART1 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 U1LCR must be zero in order to access the U1THR. The U1THR is always Write Only.

Table 96. UART1 Transmitter Holding Register (U1THR - address 0xE001 0000, when DLAB = 0 Write Only) bit description

Bit	Symbol	Description	Reset value
7:0	THR	Writing to the UART1 Transmit Holding Register causes the data to be stored in the UART1 transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.	NA

5.3 UART1 Divisor Latch registers 0 and 1 (U1DLL - 0xE001 0000 and U1DLM - 0xE001 0004, when DLAB = 1)

The UART0 Divisor Latch is part of the UART0 Baud Rate Generator and holds the value used to divide the clock in order to produce the baud rate clock, which must be 16x the desired baud rate (Equation 10–4). The U1DLL and U1DLM registers together form a 16 bit divisor where U1DLL contains the lower 8 bits of the divisor and U1DLM 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 U1LCR must be one in order to access the UART1 Divisor Latches.

(4)

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL)}$$

Details on how to select the right value for U1DLL and U1DLM if the part includes a fractional divider (see Section 10–1) can be found in Section 10–5.4.

Table 97: UART1 Divisor Latch LSB register (U1DLL - address 0xE001 C000, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLL	The UART0 Divisor Latch LSB Register, along with the U1DLM register, determines the baud rate of the UART1.	0x01

Table 98: UART0 Divisor Latch MSB register (U1DLM - address 0xE001 C004, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLM	The UART1 Divisor Latch MSB Register, along with the U1DLL register, determines the baud rate of the UART1.	0x00

5.4 UART1 Fractional Divider Register (U1FDR - 0xE001 0028)

The UART1 Fractional Divider Register (U1FDR) 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 99. UART1 Fractional Divider Register (U1FDR - address 0xE001 0028) bit description

Bit	Function	Value	Description	Reset value
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
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
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

This register controls the clock pre-scaler for the baud rate generation. The reset value of the register keeps the fractional capabilities of UART1 disabled making sure that UART1 is fully software and hardware compatible with UARTs not equipped with this feature.

UART1 baudrate can be calculated as (n = 1):

(5)

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

Where PCLK is the peripheral clock, U1DLM and U1DLL are the standard UART1 baud rate divider registers, and DIVADDVAL and MULVAL are UART1 fractional baudrate generator specific parameters.

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

1. 0 < MULVAL ≤ 15
2. 0 ≤ DIVADDVAL < 15
3. DIVADDVAL<MULVAL

The value of the U1FDR should not be modified while transmitting/receiving data or data may be lost or corrupted.

If the U1FDR 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.

5.4.1 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.

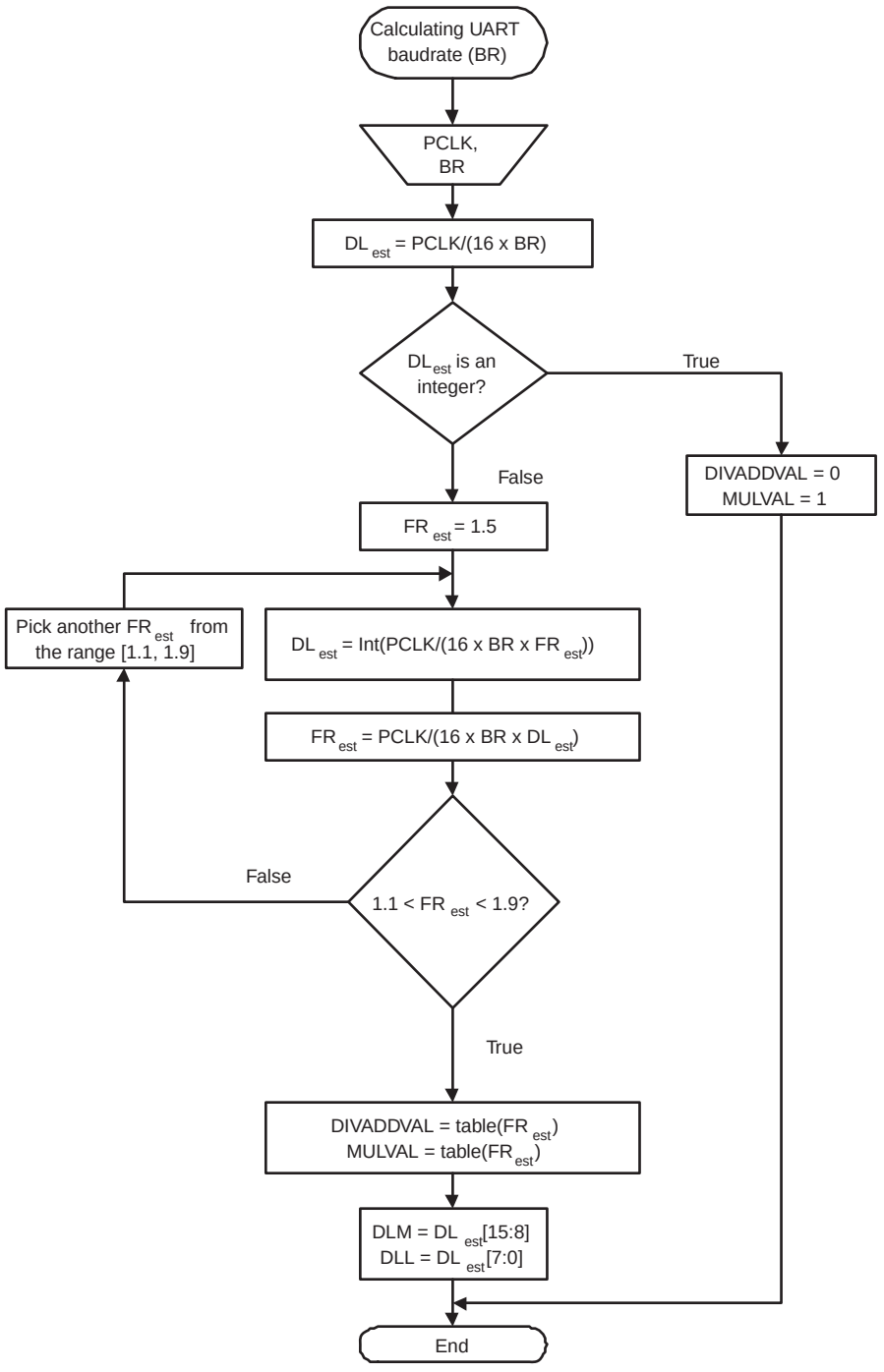


Fig 20. Algorithm for setting UART dividers

Table 100. 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

5.4.1.1 Example 1: PCLK = 14.7456 MHz, BR = 9600 Bd

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

5.4.1.2 Example 2: PCLK = 12 MHz, BR = 115200 Bd

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

The closest value for FRest = 1.628 in the look-up Table 10–100 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 10–5 the UART's baud rate is 115384 Bd. This rate has a relative error of 0.16% from the originally specified 115200 Bd.

5.5 UART1 Interrupt Enable Register (U1IER - 0xE001 0004, when DLAB = 0)

The U1IER is used to enable UART1 interrupt sources.

Table 101. UART1 Interrupt Enable Register (U1IER - address 0xE001 0004, when DLAB = 0) bit description

Bit	Symbol	Value	Description	Reset value
0	RBR Interrupt Enable		U1IER[0] enables the Receive Data Available interrupt for UART1. It also controls the Character Receive Time-out interrupt.	0
		0	Disable the RDA interrupts.	
		1	Enable the RDA interrupts.	
1	THRE Interrupt Enable		U1IER[1] enables the THRE interrupt for UART1. The status of this interrupt can be read from U1LSR[5].	0
		0	Disable the THRE interrupts.	
		1	Enable the THRE interrupts.	
2	RX Line Interrupt Enable		U1IER[2] enables the UART1 RX line status interrupts. The status of this interrupt can be read from U1LSR[4:1].	0
		0	Disable the RX line status interrupts.	
		1	Enable the RX line status interrupts.	
3	Modem Status Interrupt Enable		U1IER[3] enables the modem interrupt. The status of this interrupt can be read from U1MSR[3:0].	0
		0	Disable the modem interrupt.	
		1	Enable the modem interrupt.	
6:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7	CTS Interrupt Enable		If auto-CTS mode is enabled this bit enables/disables the modem status interrupt generation on a CTS1 signal transition. If auto-CTS mode is disabled a CTS1 transition will generate an interrupt if Modem Status Interrupt Enable (U1IER[3]) is set. In normal operation a CTS1 signal transition will generate a Modem Status Interrupt unless the interrupt has been disabled by clearing the U1IER[3] bit in the U1IER register. In auto-CTS mode a transition on the CTS1 bit will trigger an interrupt only if both the U1IER[3] and U1IER[7] bits are set.	0
		0	Disable the CTS interrupt.	
		1	Enable the CTS 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.	
9	ABTOIntEn		Enables the auto-baud time-out interrupt.	0
		0	Disable Auto-baud Time-out Interrupt.	
		1	Enable Auto-baud Time-out Interrupt.	
31:10	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.6 UART1 Interrupt Identification Register (U1IIR - 0xE001 0008, Read Only)

The U1IIR provides a status code that denotes the priority and source of a pending interrupt. The interrupts are frozen during an U1IIR access. If an interrupt occurs during an U1IIR access, the interrupt is recorded for the next U1IIR access.

Table 102. UART1 Interrupt Identification Register (U1IIR - address 0xE001 0008, read only) bit description

Bit	Symbol	Value	Description	Reset value
0	Interrupt Pending		Note that U1IIR[0] is active LOW. The pending interrupt can be determined by evaluating U1IIR[3:1].	1
		0	At least one interrupt is pending.	
		1	No interrupt is pending.	
3:1	Interrupt Identification		U1IER[3:1] identifies an interrupt corresponding to the UART1 Rx FIFO. All other combinations of U1IER[3:1] not listed above are reserved (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.	
		000	4 - Modem Interrupt.	
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFO Enable		These bits are equivalent to U1FCR[0].	0
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.	0
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.	0
31:10	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Interrupts are handled as described in Table 10–103. Given the status of U1IIR[3:0], an interrupt handler routine can determine the cause of the interrupt and how to clear the active interrupt. The U1IIR must be read in order to clear the interrupt prior to exiting the Interrupt Service Routine.

The UART1 RLS interrupt (U1IIR[3:1] = 011) is the highest priority interrupt and is set whenever any one of four error conditions occur on the UART1RX input: overrun error (OE), parity error (PE), framing error (FE) and break interrupt (BI). The UART1 Rx error condition that set the interrupt can be observed via U1LSR[4:1]. The interrupt is cleared upon an U1LSR read.

The UART1 RDA interrupt (U1IIR[3:1] = 010) shares the second level priority with the CTI interrupt (U1IIR[3:1] = 110). The RDA is activated when the UART1 Rx FIFO reaches the trigger level defined in U1FCR7:6 and is reset when the UART1 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 (U1IIR[3:1] = 110) is a second level interrupt and is set when the UART1 Rx FIFO contains at least one character and no UART1 Rx FIFO activity has occurred in 3.5 to 4.5 character times. Any UART1 Rx FIFO activity (read or write of UART1 RSR) will clear the interrupt. This interrupt is intended to flush the UART1 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 103. UART1 interrupt handling

U1IIR[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]	U1LSR Read[2]
0100	Second	RX Data Available	Rx data available or trigger level reached in FIFO (U1FCR0=1)	U1RBR Read[3] or UART1 FIFO drops below trigger level
1100	Second	Character Time-out indication	Minimum of one character in the RX FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times). The exact time will be: [(word length) × 7 - 2] × 8 + [(trigger level - number of characters) × 8 + 1] RCLKs	U1RBR Read[3]
0010	Third	THRE	THRE[2]	U1IIR Read[4] (if source of interrupt) or THR write
0000	Fourth	Modem Status	CTS or DSR or RI or DCD	MSR Read

[1] Values "0000" (see Table note 10–2), “0011”, “0101”, “0111”, “1000”, “1001”, “1010”, “1011”,”1101”,”1110”,”1111” are reserved.

[2] For details see Section 10–5.10 “UART1 Line Status Register (U1LSR - 0xE001 0014, Read Only)”

[3] For details see Section 10–5.1 “UART1 Receiver Buffer Register (U1RBR - 0xE001 0000, when DLAB = 0 Read Only)”

[4] For details see Section 10–5.6 “UART1 Interrupt Identification Register (U1IIR - 0xE001 0008, Read Only)” and Section 10–5.2 “UART1 Transmitter Holding Register (U1THR - 0xE001 0000, when DLAB = 0 Write Only)”

The UART1 THRE interrupt (U1IIR[3:1] = 001) is a third level interrupt and is activated when the UART1 THR FIFO is empty provided certain initialization conditions have been met. These initialization conditions are intended to give the UART1 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 U1THR at one time since the last THRE = 1 event. This delay is provided to give the CPU time to write data to U1THR without a THRE interrupt to decode and service. A THRE interrupt is set

immediately if the UART1 THR FIFO has held two or more characters at one time and currently, the U1THR is empty. The THRE interrupt is reset when a U1THR write occurs or a read of the U1IIR occurs and the THRE is the highest interrupt (U1IIR[3:1] = 001).

The modem interrupt (U1IIR[3:1] = 000) is available in LPC2101/02/03. It is the lowest priority interrupt and is activated whenever there is any state change on modem inputs pins, DCD, DSR or CTS. In addition, a LOW to high transition on modem input RI will generate a modem interrupt. The source of the modem interrupt can be determined by examining U1MSR[3:0]. A U1MSR read will clear the modem interrupt.

5.7 UART1 FIFO Control Register (U1FCR - 0xE001 0008)

The U1FCR controls the operation of the UART1 RX and TX FIFOs.

Table 104. UART1 FIFO Control Register (U1FCR - address 0xE001 0008) bit description

Bit	Symbol	Value	Description	Reset value
0	FIFO Enable	0	UART1 FIFOs are disabled. Must not be used in the application.	0
		1	Active HIGH enable for both UART1 Rx and TX FIFOs and U1FCR[7:1] access. This bit must be set for proper UART1 operation. Any transition on this bit will automatically clear the UART1 FIFOs.	
1	RX FIFO Reset	0	No impact on either of UART1 FIFOs.	0
		1	Writing a logic 1 to U1FCR[1] will clear all bytes in UART1 Rx FIFO and reset the pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	No impact on either of UART1 FIFOs.	0
		1	Writing a logic 1 to U1FCR[2] will clear all bytes in UART1 TX FIFO and reset the pointer logic. This bit is self-clearing.	
5:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RX Trigger Level		These two bits determine how many receiver UART1 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).	

5.8 UART1 Line Control Register (U1LCR - 0xE001 000C)

The U1LCR determines the format of the data character that is to be transmitted or received.

Table 105. UART1 Line Control Register (U1LCR - address 0xE001 000C) bit description

Bit	Symbol	Value	Description	Reset value
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.	
2	Stop Bit Select	0	1 stop bit.	0
		1	2 stop bits (1.5 if U1LCR[1:0]=00).	

Table 105. UART1 Line Control Register (U1LCR - address 0xE001 000C) bit description

Bit	Symbol	Value	Description	Reset value
3	Parity Enable	0	Disable parity generation and checking.	0
		1	Enable parity generation and checking.	
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.	
6	Break Control	0	Disable break transmission.	0
		1	Enable break transmission. Output pin UART1 TXD is forced to logic 0 when U1LCR[6] is active HIGH.	
7	Divisor Latch Access Bit (DLAB)	0	Disable access to Divisor Latches.	0
		1	Enable access to Divisor Latches.	

5.9 UART1 Modem Control Register (U1MCR - 0xE001 0010)

The U1MCR enables the modem loopback mode and controls the modem output signals.

Table 106. UART1 Modem Control Register (U1MCR - address 0xE001 0010) bit description

Bit	Symbol	Value	Description	Reset value
0	DTR Control		Source for modem output pin, DTR. This bit reads as 0 when modem loopback mode is active.	0
1	RTS Control		Source for modem output pin RTS. This bit reads as 0 when modem loopback mode is active.	0
3:2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
4	Loopback Mode Select		The modem loopback mode provides a mechanism to perform diagnostic loopback testing. Serial data from the transmitter is connected internally to serial input of the receiver. Input pin, RXD1, has no effect on loopback and output pin, TXD1 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 U1MSR will be driven by the lower four bits of the U1MCR 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.	
5:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
6	RTSen		Auto-RTS control bit.	0
		0	Disable auto-RTS flow control.	
		1	Enable auto-RTS flow control.	

Table 106. UART1 Modem Control Register (U1MCR - address 0xE001 0010) bit description

Bit	Symbol	Value	Description	Reset value
7	CTSen		Auto-CTS control bit.	0
		0	Disable auto-CTS flow control.	
		1	Enable auto-CTS flow control.	

5.9.1 Auto-flow control

If auto-RTS mode is enabled the UART1‘s receiver FIFO hardware controls the RTS1 output of the UART1. If the auto-CTS mode is enabled the UART1‘s U1TSR hardware will only start transmitting if the CTS1 input signal is asserted.

5.9.1.1 Auto-RTS

The auto-RTS function is enabled by setting the RTSen bit. Auto-RTS data flow control originates in the U1RBR 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, RTS1 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 RTS1 until after it has begun sending the additional byte. RTS1 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 RTS Control bit controls the RTS1 output of the UART1. If Auto-RTS mode is enabled, hardware controls the RTS1 output, and the actual value of RTS1 will be copied in the RTS Control bit of the UART1. As long as Auto-RTS is enabled, the value of the RTS Control bit is read-only for software.

Example: Suppose the UART1 operating in type 550 has trigger level in U1FCR set to 0x2 then if Auto-RTS is enabled the UART1 will deassert the RTS1 output as soon as the receive FIFO contains 8 bytes (Table 10–104). The RTS1 output will be reasserted as soon as the receive FIFO hits the previous trigger level: 4 bytes.

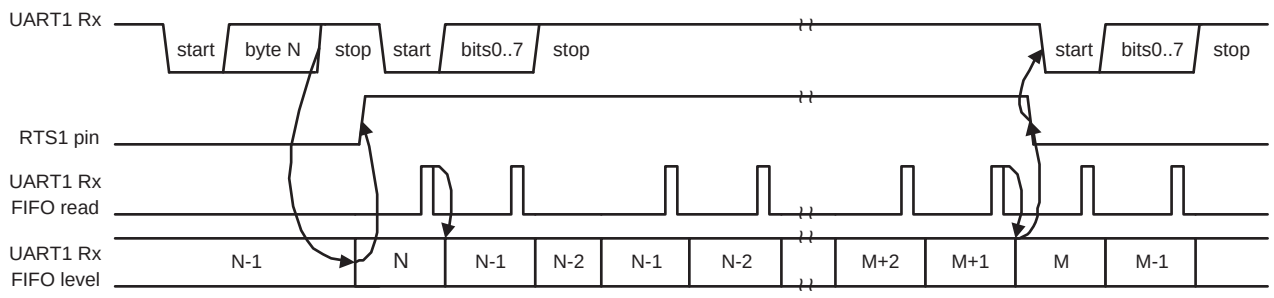


Fig 21. Auto-RTS functional timing

5.9.1.2 Auto-CTS

The auto-CTS function is enabled by setting the CTSen bit. If auto-CTS is enabled the transmitter circuitry in the U1TSR module checks CTS1 input before sending the next data byte. When CTS1 is active (LOW), the transmitter sends the next byte. To stop the

transmitter from sending the following byte, CTS1 must be released before the middle of the last stop bit that is currently being sent. In auto-CTS mode a change of the CTS1 signal does not trigger a modem status interrupt unless the CTS Interrupt Enable bit is set, Delta CTS bit in the U1MSR will be set though. Table 10–107 lists the conditions for generating a Modem Status interrupt.

Table 107. Modem status interrupt generation

Enable Modem Status Interrupt (U1IER[3])	CTSen (U1MCR[7])	CTS Interrupt Enable (U1IER[7])	Delta CTS (U1MSR[0])	Delta DCD or Trailing Edge RI or Delta DSR (U1MSR[3] or U1MSR[2] or (U1MSR[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 10–22 illustrates the auto-CTS functional timing.

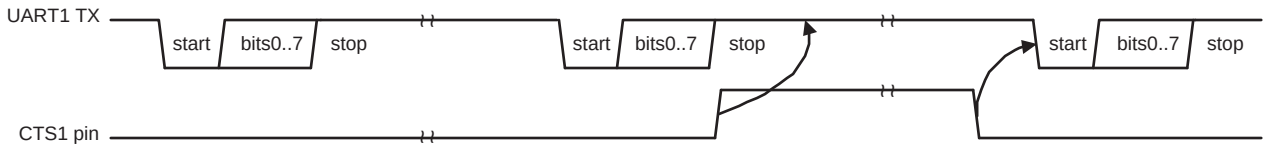


Fig 22. Auto-CTS functional timing

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.

5.10 UART1 Line Status Register (U1LSR - 0xE001 0014, Read Only)

The U1LSR is a read-only register that provides status information on the UART1 TX and RX blocks.

Table 108. UART1 Line Status Register (U1LSR - address 0xE001 0014, read only) bit description

Bit	Symbol	Value	Description	Reset value
0	Receiver Data Ready (RDR)		U1LSR[0] is set when the U1RBR holds an unread character and is cleared when the UART1 RBR FIFO is empty.	0
		0	U1RBR is empty.	
		1	U1RBR contains valid data.	
1	Overrun Error (OE)		The overrun error condition is set as soon as it occurs. An U1LSR read clears U1LSR[1]. U1LSR[1] is set when UART1 RSR has a new character assembled and the UART1 RBR FIFO is full. In this case, the UART1 RBR FIFO will not be overwritten and the character in the UART1 RSR will be lost.	0
		0	Overrun error status is inactive.	
		1	Overrun 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 U1LSR read clears U1LSR[2]. Time of parity error detection is dependent on U1FCR[0]. **Note:** A parity error is associated with the character at the top of the UART1 RBR FIFO.	0
		0	Parity error status is inactive.	
		1	Parity error status is active.	
3	Framing Error (FE)		When the stop bit of a received character is a logic 0, a framing error occurs. An U1LSR read clears U1LSR[3]. The time of the framing error detection is dependent on U1FCR0. 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 UART1 RBR FIFO.	0
		0	Framing error status is inactive.	
		1	Framing error status is active.	
4	Break Interrupt (BI)		When RXD1 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 RXD1 goes to marking state (all 1's). An U1LSR read clears this status bit. The time of break detection is dependent on U1FCR[0]. **Note:** The break interrupt is associated with the character at the top of the UART1 RBR FIFO.	0
		0	Break interrupt status is inactive.	
		1	Break interrupt status is active.	
5	Transmitter Holding Register Empty (THRE)		THRE is set immediately upon detection of an empty UART1 THR and is cleared on a U1THR write.	1
		0	U1THR contains valid data.	
		1	U1THR is empty.	
6	Transmitter Empty (TEMT)		TEMT is set when both U1THR and U1TSR are empty; TEMT is cleared when either the U1TSR or the U1THR contain valid data.	1
		0	U1THR and/or the U1TSR contains valid data.	
		1	U1THR and the U1TSR are empty.	

Table 108. UART1 Line Status Register (U1LSR - address 0xE001 0014, read only) bit description

Bit	Symbol	Value	Description	Reset value
7	Error in RX FIFO (RXFE)		U1LSR[7] is set when a character with a RX error such as framing error, parity error or break interrupt, is loaded into the U1RBR. This bit is cleared when the U1LSR register is read and there are no subsequent errors in the UART1 FIFO.	0
		0	U1RBR contains no UART1 RX errors or U1FCR[0]=0.	
		1	UART1 RBR contains at least one UART1 RX error.	

5.11 UART1 Modem Status Register (U1MSR - 0xE001 0018)

The U1MSR is a read-only register that provides status information on the modem input signals. U1MSR[3:0] is cleared on U1MSR read. Note that modem signals have no direct affect on UART1 operation, they facilitate software implementation of modem signal operations.

Table 109. UART1 Modem Status Register (U1MSR - address 0xE001 0018) bit description

Bit	Symbol	Value	Description	Reset value
0	Delta CTS		Set upon state change of input CTS. Cleared on an U1MSR read.	0
		0	No change detected on modem input, CTS.	
		1	State change detected on modem input, CTS.	
1	Delta DSR		Set upon state change of input DSR. Cleared on an U1MSR read.	0
		0	No change detected on modem input, DSR.	
		1	State change detected on modem input, DSR.	
2	Trailing Edge RI		Set upon LOW to HIGH transition of input RI. Cleared on an U1MSR read.	0
		0	No change detected on modem input, RI.	
		1	LOW-to-HIGH transition detected on RI.	
3	Delta DCD		Set upon state change of input DCD. Cleared on an U1MSR read.	0
		0	No change detected on modem input, DCD.	
		1	State change detected on modem input, DCD.	
4	CTS		Clear To Send State. Complement of input signal CTS. This bit is connected to U1MCR[1] in modem loopback mode.	0
5	DSR		Data Set Ready State. Complement of input signal DSR. This bit is connected to U1MCR[0] in modem loopback mode.	0
6	RI		Ring Indicator State. Complement of input RI. This bit is connected to U1MCR[2] in modem loopback mode.	0
7	DCD		Data Carrier Detect State. Complement of input DCD. This bit is connected to U1MCR[3] in modem loopback mode.	0

5.12 UART1 Scratch Pad Register (U1SCR - 0xE001 001C)

The U1SCR has no effect on the UART1 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 U1SCR has occurred.

Table 110. UART1 Scratch Pad Register (U1SCR - address 0xE001 0014) bit description

Bit	Symbol	Description	Reset value
7:0	Pad	A readable, writable byte.	0x00

5.13 UART1 Auto-baud Control Register (U1ACR - 0xE001 0020)

The UART1 Auto-baud Control Register (U1ACR) 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 111. Auto-baud Control Register (U1ACR - 0xE001 0020) bit description

Bit	Symbol	Value	Description	Reset value
0	Start		This bit is automatically cleared after auto-baud completion.	0
		0	Auto-baud stop (auto-baud is not running).	
		1	Auto-baud start (auto-baud is running).Auto-baud run bit. This bit is automatically cleared after auto-baud completion.	
1	Mode		Auto-baud mode select bit.	0
		0	Mode 0.	
		1	Mode 1.	
2	AutoRestart	0	No restart	0
		1	Restart in case of time-out (counter restarts at next UART1 Rx falling edge)	
7:3	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
8	ABEOIntClr		End of auto-baud interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the U1IIR. Writing a 0 has no impact.	0
9	ABTOIntClr		Auto-baud time-out interrupt clear bit (write only accessible). Writing a 1 will clear the corresponding interrupt in the U1IIR. Writing a 0 has no impact.	0
31:10	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0

5.14 Auto-baud

The UART1 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 U1DLM and U1DLL accordingly.

Auto-baud is started by setting the U1ACR Start bit. Auto-baud can be stopped by clearing the U1ACR 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 U1ACR Mode bit. In mode 0 the baud-rate is measured on two subsequent falling edges of the UART1 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 UART1 Rx pin (the length of the start bit).

The U1ACR 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 UART1 Rx pin.

The auto-baud function can generate two interrupts.

- The U1IIR ABTOInt interrupt will get set if the interrupt is enabled (U1IER ABToIntEn is set and the auto-baud rate measurement counter overflows).
- The U1IIR ABEOInt interrupt will get set if the interrupt is enabled (U1IER ABEOIntEn is set and the auto-baud has completed successfully).

The auto-baud interrupts have to be cleared by setting the corresponding U1ACR ABTOIntClr and ABEOIntEn bits.

Typically the fractional baud-rate generator is disabled (DIVADDVAL = 0) during auto-baud. However, if the fractional baud-rate generator is enabled (DIVADDVAL > 0), it is going to impact the measuring of UART1 Rx pin baud-rate, but the value of the U1FDR register is not going to be modified after rate measurement. Also, when auto-baud is used, any write to U1DLM and U1DLL registers should be done before U1ACR register write. The minimum and the maximum baudrates supported by UART1 are function of PCLK, number of data bits, stop-bits and parity bits.

(6)

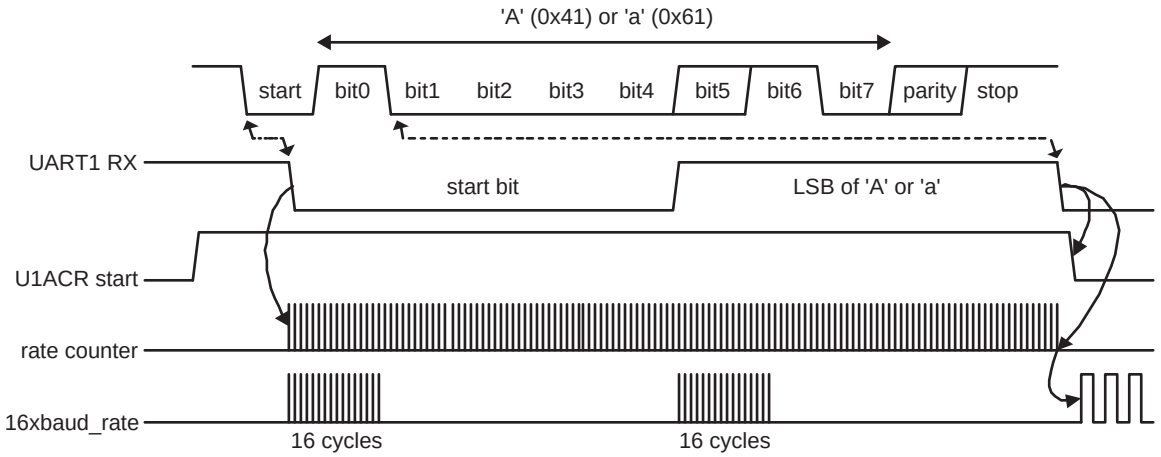
$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART1_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax$$

5.15 Auto-baud modes

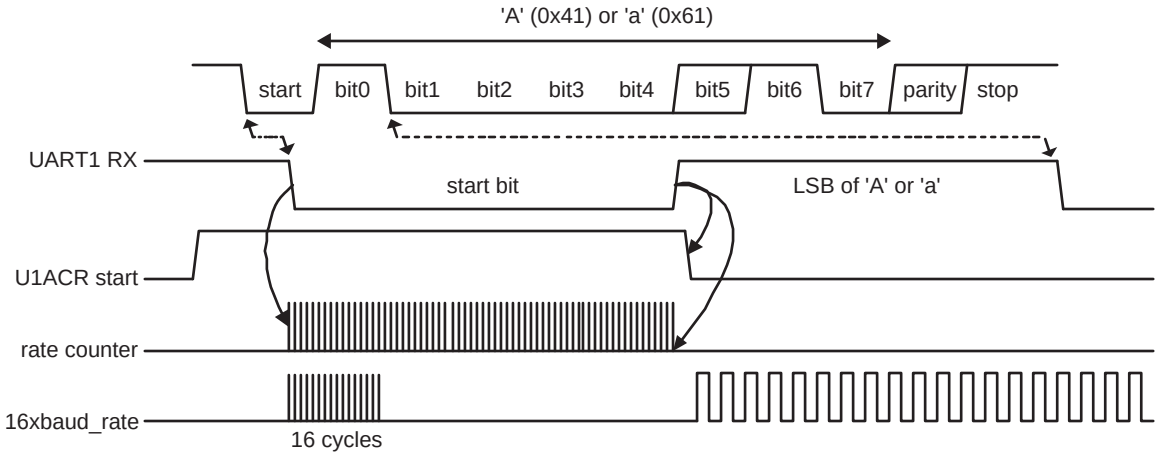
When the software is expecting an ”AT" command, it configures the UART1 with the expected character format and sets the U1ACR Start bit. The initial values in the divisor latches U1DLM and U1DLM don‘t care. Because of the ”A" or ”a" ASCII coding (”A" = 0x41, ”a" = 0x61), the UART1 Rx pin sensed start bit and the LSB of the expected character are delimited by two falling edges. When the U1ACR Start bit is set, the auto-baud protocol will execute the following phases:

1. On U1ACR Start bit setting, the baud-rate measurement counter is reset and the UART1 U1RSR is reset. The U1RSR baud rate is switch to the highest rate.
2. A falling edge on UART1 Rx pin triggers the beginning of the start bit. The rate measuring counter will start counting PCLK cycles optionally pre-scaled by the fractional baud-rate generator.
3. During the receipt of the start bit, 16 pulses are generated on the RSR baud input with the frequency of the (fractional baud-rate pre-scaled) UART1 input clock, guaranteeing the start bit is stored in the U1RSR.
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 UART1 input clock (PCLK).
5. If Mode = 0 then the rate counter will stop on next falling edge of the UART1 Rx pin. If Mode = 1 then the rate counter will stop on the next rising edge of the UART1 Rx pin.

6. The rate counter is loaded into U1DLM/U1DLL and the baud-rate will be switched to normal operation. After setting the U1DLM/U1DLL the end of auto-baud interrupt U1IIR ABEOInt will be set, if enabled. The U1RSR 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 23. Autobaud a) mode 0 and b) mode 1 waveform

5.16 UART1 Transmit Enable Register (U1TER - 0xE001 0030)

LPC2104/05/06’s U1TER enables implementation of software and hardware flow control. When TXEn=1, UART1 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART1 transmission will stop.

Table 10–112 describes how to use TXEn bit in order to achieve software flow control.

Table 112. UART1 Transmit Enable Register (U1TER - address 0xE001 0030) bit description

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

6. Architecture

The architecture of the UART1 is shown below in the block diagram.

The APB interface provides a communications link between the CPU or host and the UART1.

The UART1 receiver block, U1RX, monitors the serial input line, RXD1, for valid input. The UART1 RX Shift Register (U1RSR) accepts valid characters via RXD1. After a valid character is assembled in the U1RSR, it is passed to the UART1 RX Buffer Register FIFO to await access by the CPU or host via the generic host interface.

The UART1 transmitter block, U1TX, accepts data written by the CPU or host and buffers the data in the UART1 TX Holding Register FIFO (U1THR). The UART1 TX Shift Register (U1TSR) reads the data stored in the U1THR and assembles the data to transmit via the serial output pin, TXD1.

The UART1 Baud Rate Generator block, U1BRG, generates the timing enables used by the UART1 TX block. The U1BRG clock input source is the APB clock (PCLK). The main clock is divided down per the divisor specified in the U1DLL and U1DLM registers. This divided down clock is a 16x oversample clock, NBAUDOUT.

The modem interface contains registers U1MCR and U1MSR. This interface is responsible for handshaking between a modem peripheral and the UART1.

The interrupt interface contains registers U1IER and U1IIR. The interrupt interface receives several one clock wide enables from the U1TX and U1RX blocks.

Status information from the U1TX and U1RX is stored in the U1LSR. Control information for the U1TX and U1RX is stored in the U1LCR.

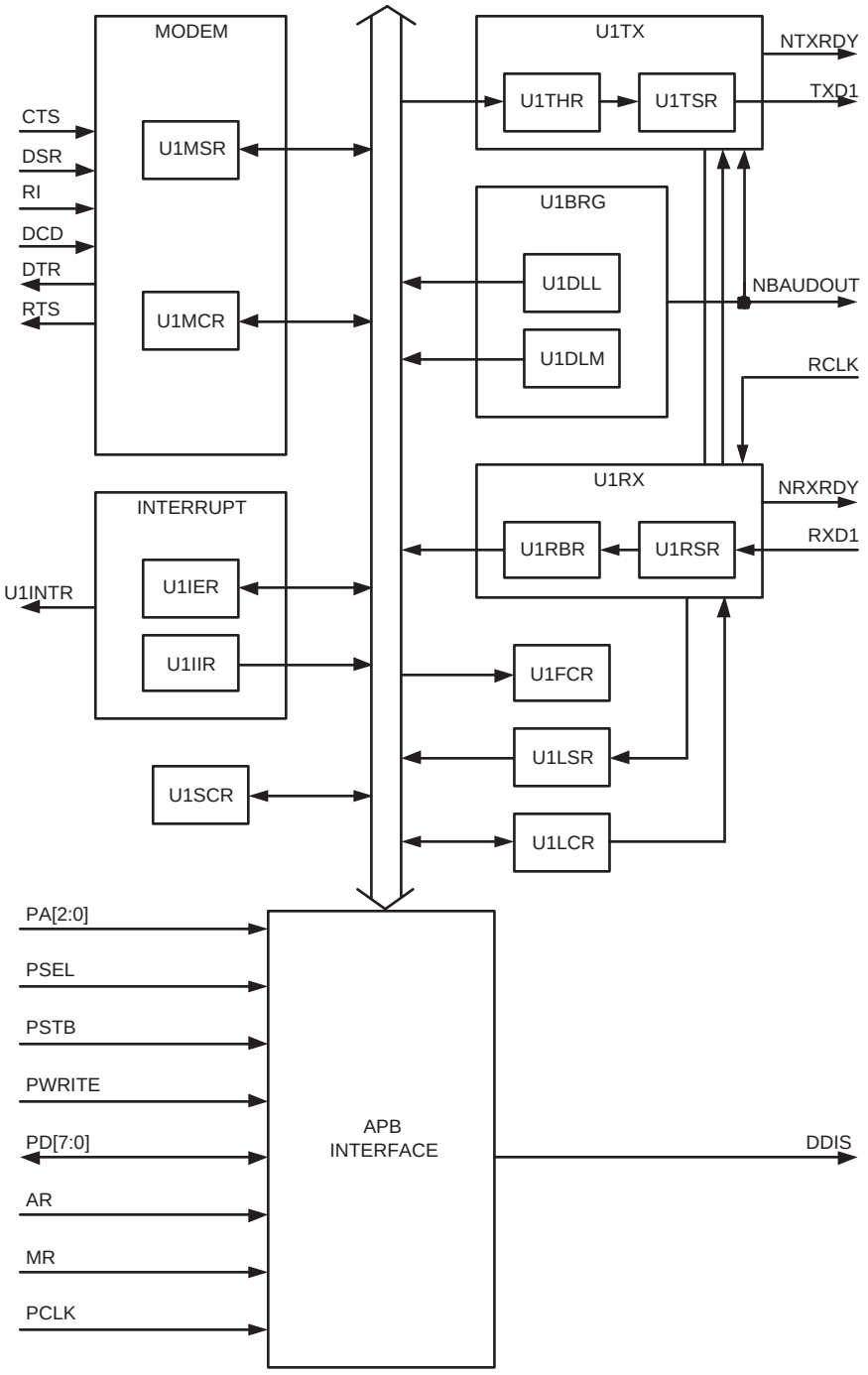


Fig 24. UART1 block diagram

1. Basic configuration

The I2C peripheral is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCI2C.

   Remark: On reset, I2C is enabled (PCI2C = 1).

2. Pins: Select I2C pins in registers PINSEL0 (see Section 7–2).

   Remark: The I2C pins SDA and SCL are open-drain pins.

3. Interrupts: Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

4. Initialization: see Section 11–9.15 and Section 11–10.1.

2. Features

- Standard I2C compliant bus interfaces that may be configured as Master, Slave, or Master/Slave.
- Arbitration between simultaneously transmitting masters without corruption of serial data on the bus.
- Programmable clock to allow adjustment of I2C transfer rates.
- Bidirectional data transfer between masters and slaves.
- 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 I2C-bus may be used for test and diagnostic purposes.

3. Applications

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

4. Description

A typical I2C-bus configuration is shown in Figure 11–25. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I2C-bus:

- Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte.
- Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. Next follows the data bytes transmitted by the slave to the master. The master returns

an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a “not acknowledge” is returned. The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the I2C-bus will not be released.

The LPC2104/05/06 I2C interface is byte oriented, and have four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

The I2C interface complies with entire I2C specification, supporting the ability to turn power off to the LPC2104/05/06 without causing a problem with other devices on the same I2C-bus. This is sometimes a useful capability, but intrinsically limits alternate uses for the same pins if the I2C interface is not used.

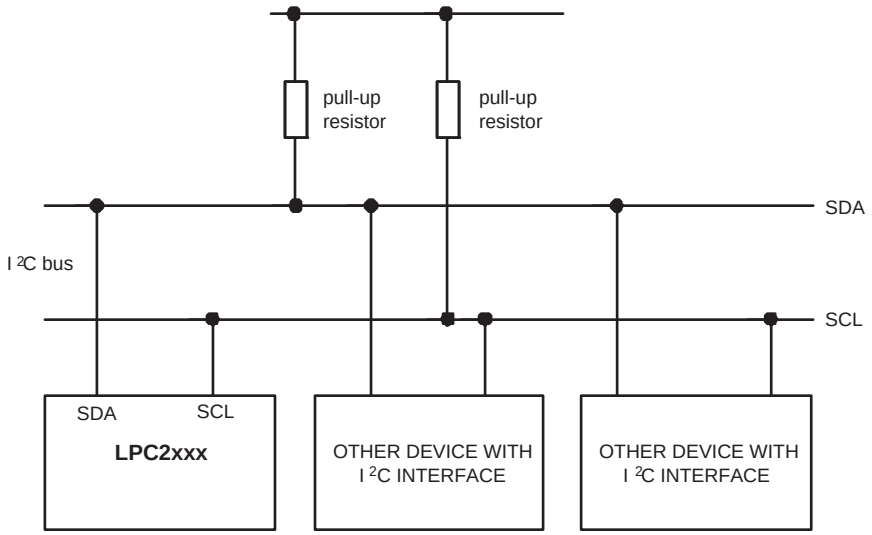


Fig 25. I2C-bus Configuration

5. Pin description

Table 113. I2C pin description

Pin	Type	Description
SDA	Input/Output	I2C serial data
SCL	Input/Output	I2C Serial clock

Remark: The SDA and SCL outputs are open-drain outputs for I2C-bus compliance.

6. I2C operating modes

In a given application, the I2C block may operate as a master, a slave, or both. In the slave mode, the I2C hardware looks for 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 I2C 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 11–114. I2EN must be set to 1 to enable the I2C function. If the AA bit is 0, the I2C interface will not acknowledge any address when another device is master of the bus, so it can not enter slave mode. The STA, STO and SI bits must be 0. The SI Bit is cleared by writing 1 to the SIC bit in the I2CONCLR register.

Table 114. I2CCONSET 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 I2C interface will enter master transmitter mode when software sets the STA bit. The I2C 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 11–129 to Table 11–132.

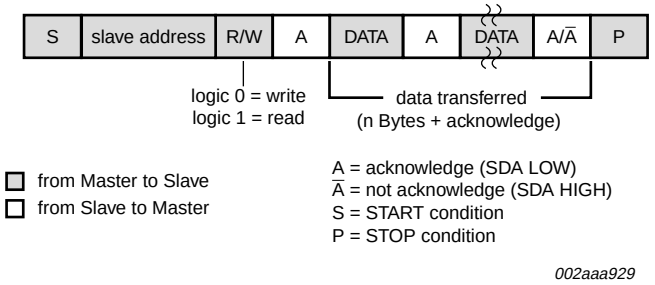


Fig 26. Format in the Master Transmitter mode

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 I2C 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 11–130.

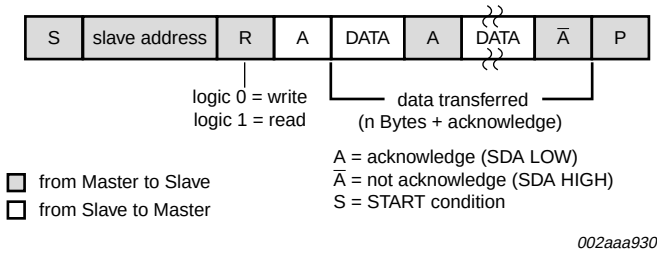


Fig 27. Format of Master Receiver mode

After a repeated START condition, I2C may switch to the master transmitter mode.

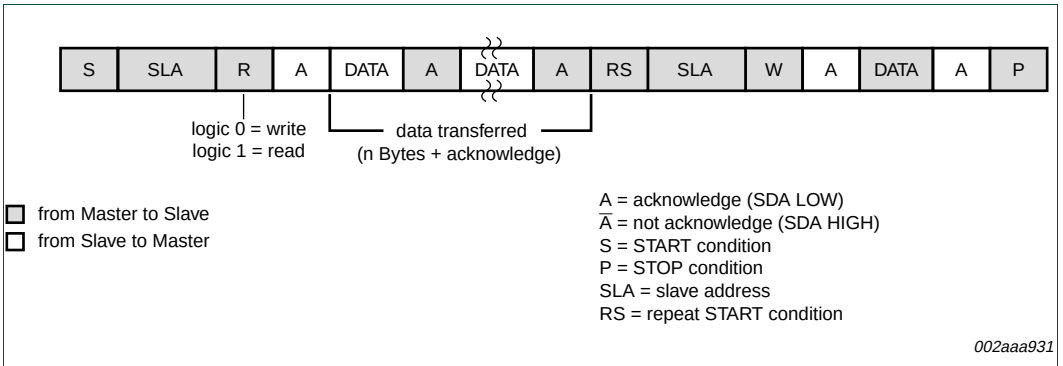


Fig 28. A Master Receiver switches to Master Transmitter after sending Repeated START

6.3 Slave Receiver mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, user write the Slave Address register (I2ADR) and write the I2C Control Set register (I2CONSET) as shown in Table 11–115.

Table 115. I2CONSET used to configure Slave mode

Bit	7	6	5	4	3	2	1	0
Symbol	-	I2EN	STA	STO	SI	AA	-	-
Value	-	1	0	0	0	1	-	-

I2EN must be set to 1 to enable the I2C function. AA bit must be set to 1 to acknowledge its own slave address or the general call address. The STA, STO and SI bits are set to 0.

After I2ADR and I2CONSET are initialized, the I2C interface waits until it is addressed by its own address or general address followed by the data direction bit. If the direction bit is 0 (W), it enters slave receiver mode. If the direction bit is 1 (R), it enters slave transmitter mode. After the address and direction bit have been received, the SI bit is set and a valid status code can be read from the Status register (I2STAT). Refer to Table 11–131 for the status codes and actions.

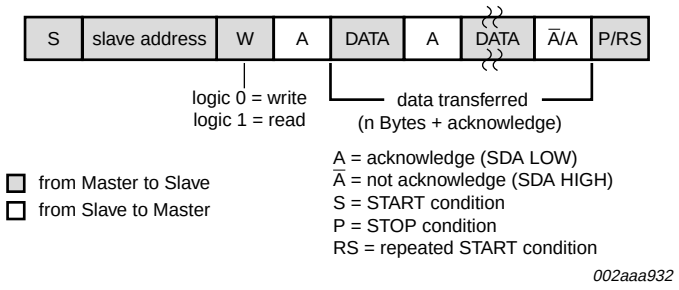


Fig 29. Format of Slave Receiver mode

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, I2C may operate as a master and as a slave. In the slave mode, the I2C 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 I2C interface switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

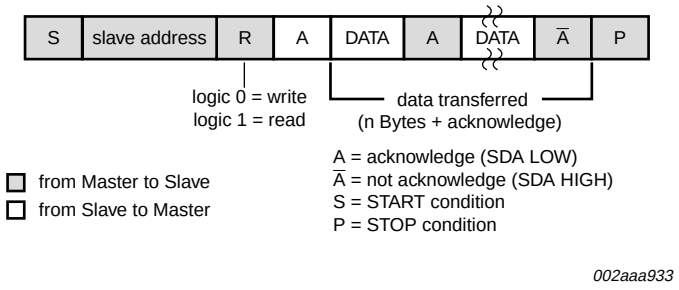


Fig 30. Format of Slave Transmitter mode

7. I2C Implementation and operation

Figure 11–31 shows how the on-chip I2C-bus interface is implemented, and the following text describes the individual blocks.

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 I2C is a special pad designed to conform to the I2C specification.

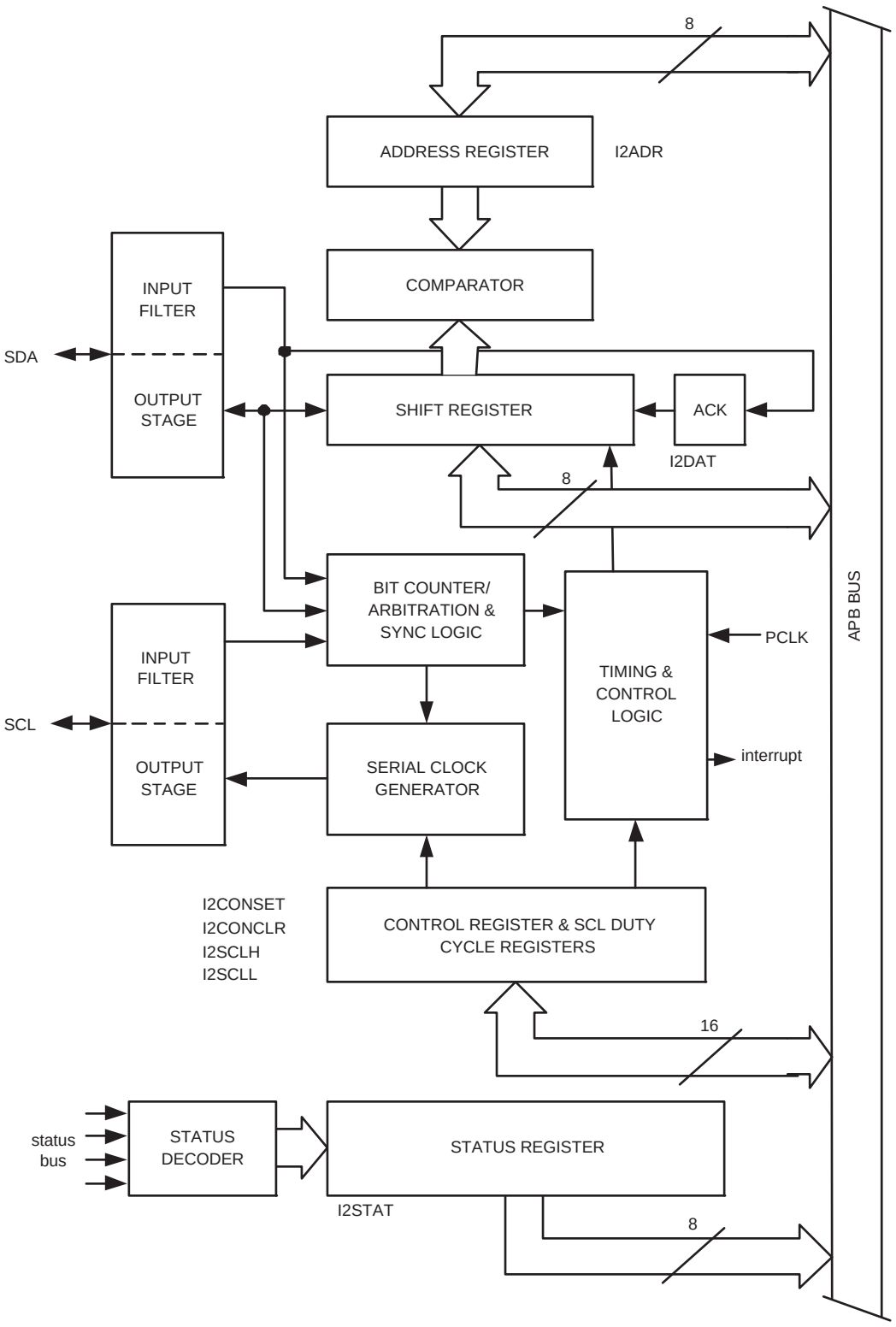


Fig 31. I²C serial interface block diagram

7.2 Address Register, I2ADDR

This register may be loaded with the 7-bit slave address (7 most significant bits) to which the I2C block will respond when programmed as a slave transmitter or receiver. The LSB (GC) is used to enable general call address (0x00) recognition.

7.3 Comparator

The comparator compares the received 7-bit slave address with its own slave address (7 most significant bits in I2ADR). 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.

7.4 Shift register, I2DAT

This 8-bit register contains a byte of serial data to be transmitted or a byte which has just been received. 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. While data is being shifted out, data on the bus is simultaneously being shifted in; I2DAT 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 I2DAT.

7.5 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 I2C-bus. If another device on the bus overrules a logic 1 and pulls the SDA line low, arbitration is lost, and the I2C block immediately changes from master transmitter to slave receiver. The I2C 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 I2C 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 I2C block generates no further clock pulses. Figure 11–32 shows the arbitration procedure.

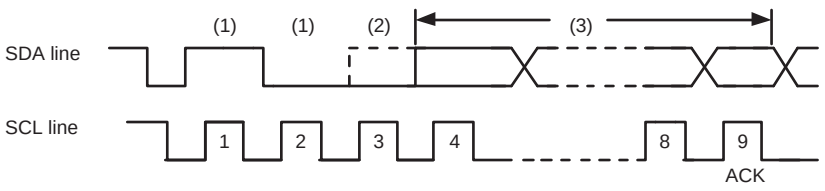


Fig 32. Arbitration procedure

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 11–33 shows the synchronization procedure.

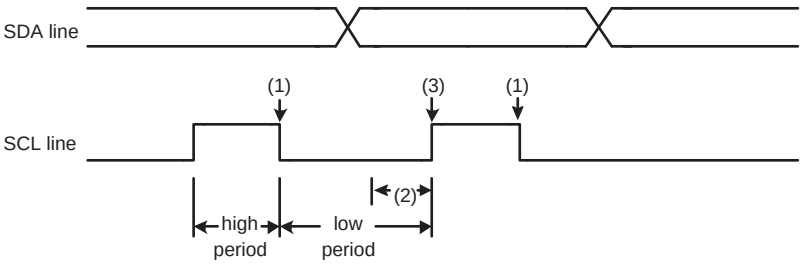


Fig 33. Serial clock synchronization

A slave may stretch the space duration to slow down the bus master. The space duration may also be stretched for handshaking purposes. This can be done after each bit or after a complete byte transfer. the I²C block will stretch the SCL space duration after a byte has been transmitted or received and the acknowledge bit has been transferred. The serial interrupt flag (SI) is set, and the stretching continues until the serial interrupt flag is cleared.

7.6 Serial clock generator

This programmable clock pulse generator provides the SCL clock pulses when the I²C block is in the master transmitter or master receiver mode. It is switched off when the I²C block is in a slave mode. The I²C output clock frequency and duty cycle is programmable via the I²C Clock Control Registers. See the description of the 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.

7.7 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 I²C-bus status.

7.8 Control register, I2CONSET and I2CONCLR

The I²C control register contains bits used to control the following I²C block functions: start and restart of a serial transfer, termination of a serial transfer, bit rate, address recognition, and acknowledgment.

The contents of the I²C control register may be read as I2CONSET. Writing to I2CONSET will set bits in the I²C control register that correspond to ones in the value written. Conversely, writing to I2CONCLR will clear bits in the I²C control register that correspond to ones in the value written.

7.9 Status decoder and Status register

The status decoder takes all of the internal status bits and compresses them into a 5-bit code. This code is unique for each I²C-bus status. The 5-bit code may be used to generate vector addresses for fast processing of the various service routines. Each service routine processes a particular bus status. There are 26 possible bus states if all

four modes of the I2C block are used. The 5-bit status code is latched into the five most significant bits of the status register when the serial interrupt flag is set (by hardware) and remains stable until the interrupt flag is cleared by software. The three least significant bits of the status register are always zero. If the status code is used as a vector to service routines, then the routines are displaced by eight address locations. Eight bytes of code is sufficient for most of the service routines (see the software example in this section).

8. Register description

Each I2C interface contains 7 registers as shown in Table 11–116 below.

Table 116. I2C register map

Name	Description	Access	Reset value[1]	Address
I2CONSET	**I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I2C control register is set. Writing a zero has no effect on the corresponding bit in the I2C control register.	R/W	0x00	0xE001 C000
I2STAT	**I2C Status Register.** During I2C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8	0xE001 C004
I2DAT	**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.	R/W	0x00	0xE001 C008
I2ADR	**I2C Slave Address Register.** Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address.	R/W	0x00	0xE001 C00C
I2SCLH	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I2C clock.	R/W	0x04	0xE001 C010
I2SCLL	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I2C clock. I2SCLL and I2SCLH together determine the clock frequency generated by an I2C master and certain times used in slave mode.	R/W	0x04	0xE001 C014
I2CONCLR	**I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I2C control register is cleared. Writing a zero has no effect on the corresponding bit in the I2C control register.	WO	NA	0xE001 C018

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

8.1 I2C Control Set register (I2CONSET - 0xE001 C000)

The I2CONSET registers control setting of bits in the I2CON register that controls operation of the I2C interface. Writing a one to a bit of this register causes the corresponding bit in the I2C control register to be set. Writing a zero has no effect.

Table 117. I2C Control Set register (I2CONSET - address 0xE001 C000) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AA	Assert acknowledge flag. See the text below.	
3	SI	I2C interrupt flag.	0
4	STO	STOP flag. See the text below.	0
5	STA	START flag. See the text below.	0
6	I2EN	I2C interface enable. See the text below.	0
7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I2EN I2C Interface Enable. When I2EN is 1, the I2C interface is enabled. I2EN can be cleared by writing 1 to the I2ENC bit in the I2CONCLR register. When I2EN is 0, the I2C interface is disabled.

When I2EN is “0”, the SDA and SCL input signals are ignored, the I2C block is in the “not addressed” slave state, and the STO bit is forced to “0”.

I2EN should not be used to temporarily release the I2C-bus since, when I2EN is reset, the I2C-bus status is lost. The AA flag should be used instead.

STA is the START flag. Setting this bit causes the I2C interface to enter master mode and transmit a START condition or transmit a repeated START condition if it is already in master mode.

When STA is 1 and the I2C interface is not already in master mode, it enters master mode, checks the bus and generates a START condition if the bus is free. If the bus is not free, it waits for a STOP condition (which will free the bus) and generates a START condition after a delay of a half clock period of the internal clock generator. If the I2C interface is already in master mode and data has been transmitted or received, it transmits a repeated START condition. STA may be set at any time, including when the I2C interface is in an addressed slave mode.

STA can be cleared by writing 1 to the STAC bit in the 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 I2C-bus if it the interface is in master mode, and transmits a START condition thereafter. If the I2C interface is in slave mode, an internal STOP condition is generated, but is not transmitted on the bus.

STO is the STOP flag. Setting this bit causes the I2C interface to transmit a STOP condition in master mode, or recover from an error condition in slave mode. When STO is 1 in master mode, a STOP condition is transmitted on the I2C-bus. When the bus detects the STOP condition, STO is cleared automatically.

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

SI is the I2C Interrupt Flag. This bit is set when the I2C state changes. However, entering state F8 does not set SI since there is nothing for an interrupt service routine to do in that case.

While SI is set, the low period of the serial clock on the SCL line is stretched, and the serial transfer is suspended. When SCL is high, it is unaffected by the state of the SI flag. SI must be reset by software, by writing a 1 to the SIC bit in 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 I2C is in the master receiver mode.
4. A data byte has been received while the I2C 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 I2C is in the master receiver mode.
2. A data byte has been received while the I2C is in the addressed slave receiver mode.

8.2 I2C Control Clear register (I2CONCLR - 0xE001 C018)

The I2CONCLR registers control clearing of bits in the I2CON register that controls operation of the I2C interface. Writing a one to a bit of this register causes the corresponding bit in the I2C control register to be cleared. Writing a zero has no effect.

Table 118. I2C Control Set register (I2CONCLR - address 0xE001 C018) bit description

Bit	Symbol	Description	Reset value
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AAC	Assert acknowledge Clear bit.	
3	SIC	I2C interrupt Clear bit.	0
4	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
5	STAC	START flag Clear bit.	0
6	I2ENC	I2C interface Disable bit.	0
7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

AAC is the Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the I2CONSET register. Writing 0 has no effect.

SIC is the I2C Interrupt Clear bit. Writing a 1 to this bit clears the SI bit in the 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 I2C Interface Disable bit. Writing a 1 to this bit clears the I2EN bit in the I2CONSET register. Writing 0 has no effect.

8.3 I2C Status register (I2STAT - 0xE001 C004)

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

Table 119. I2C Status register (I2STAT - address 0xE001) bit description

Bit	Symbol	Description	Reset value
2:0	-	These bits are unused and are always 0.	0
7:3	Status	These bits give the actual status information about the I2C interface.	0x1F

The three least significant bits are always 0. Taken as a byte, the status register contents represent a status code. There are 26 possible status codes. When the status code is 0xF8, there is no relevant information available and the SI bit is not set. All other 25 status codes correspond to defined I2C states. When any of these states entered, the SI bit will be set. For a complete list of status codes, refer to tables from Table 11–129 to Table 11–132.

8.4 I2C Data register (I2DAT - 0xE001 C008)

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 120. I2C Data register (I2DAT - address 0xE001 C008) 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

8.5 I2C Slave Address register (I2ADR - 0xE001 C00C)

These registers are readable and writable, and is only used when an I2C interface is set to slave mode. In master mode, this register has no effect. The LSB of I2ADR is the general call bit. When this bit is set, the general call address (0x00) is recognized.

Table 121. I2C Slave Address register (I2ADR - address 0xE001 C00C) bit description

Bit	Symbol	Description	Reset value
0	GC	General Call enable bit.	0
7:1	Address	The I2C device address for slave mode.	0x00

8.6 I2C SCL High duty cycle register (I2SCLH - 0xE001 C010)

Table 122. I2C SCL High Duty Cycle register (I2SCLH - address 0xE001 C010) bit description

Bit	Symbol	Description	Reset value
15:0	SCLH	Count for SCL HIGH time period selection.	0x0004

8.7 I²C SCL Low duty cycle register (I2SCLL - 0xE001 C014)

Table 123. I²C SCL Low Duty Cycle register (I2SCLL - address 0xE001 C014) bit description

Bit	Symbol	Description	Reset value
15:0	SCLL	Count for SCL LOW time period selection.	0x0004

8.8 Selecting the appropriate I²C 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 frequency is determined by the following formula (PCLK is the frequency of the peripheral bus APB):

(7)

$$I^2C_{bitfrequency} = \frac{PCLK}{I2CSCLH + I2CSCLL}$$

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 I²C-bus specification defines the SCL low time and high time at different values for a 400 kHz I²C rate. The value of the register must ensure that the data rate is in the I²C data rate range of 0 through 400 kHz. Each register value must be greater than or equal to 4. Table 11–124 gives some examples of I²C-bus rates based on PCLK frequency and I2SCLL and I2SCLH values.

Table 124. Example I²C clock rates

I2SCLL + I2SCLH	I²C 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

9. Details of I²C operating modes

The four operating modes are:

- Master Transmitter
- Master Receiver
- Slave Receiver

- Slave Transmitter

Data transfers in each mode of operation are shown in Figures 34 to 38. Table 11–125 lists abbreviations used in these figures when describing the I²C operating modes.

Table 125. Abbreviations used to describe an I²C operation

Abbreviation	Explanation
S	Start Condition
SLA	7-bit slave address
R	Read bit (high level at SDA)
W	Write bit (low level at SDA)
A	Acknowledge bit (low level at SDA)
$\overline{A}$	Not acknowledge bit (high level at SDA)
Data	8-bit data byte
P	Stop condition

In Figures 34 to 38, 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 11–129 to Table 11–133.

9.1 Master Transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 11–34). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

Table 126. 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 I²C rate must also be configured in the I2SCLL and I2SCLH registers. I2EN must be set to logic 1 to enable the I²C block. If the AA bit is reset, the I²C block will not acknowledge its own slave address or the general call address in the event of another device becoming master of the bus. In other words, if AA is reset, the I²C interface cannot enter a slave mode. STA, STO, and SI must be reset.

The master transmitter mode may now be entered by setting the STA bit. The I²C logic will now test the I²C-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 11–129. After a repeated start condition (state 0x10). The I2C 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 11–35). 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 11–130. After a repeated start condition (state 0x10), the I2C 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 11–36). To initiate the slave receiver mode, I2ADR and I2CON must be loaded as follows:

Table 127. I2CADR 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 I2C block will respond when addressed by a master. If the LSB (GC) is set, the I2C block will respond to the general call address (0x00); otherwise it ignores the general call address.

Table 128. I2CONSET 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 I2C-bus rate settings do not affect the I2C block in the slave mode. I2EN must be set to logic 1 to enable the I2C block. The AA bit must be set to enable the I2C 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 I2C block waits until it is addressed by its own slave address followed by the data direction bit which must be “0” (W) for the I2C 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 104. The slave receiver mode may also be entered if arbitration is lost while the I2C block is in the master mode (see status 0x68 and 0x78).

If the AA bit is reset during a transfer, the I2C block will return a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the I2C block does not respond to its own slave address or a general call address. However, the I2C-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 I2C block from the I2C-bus.

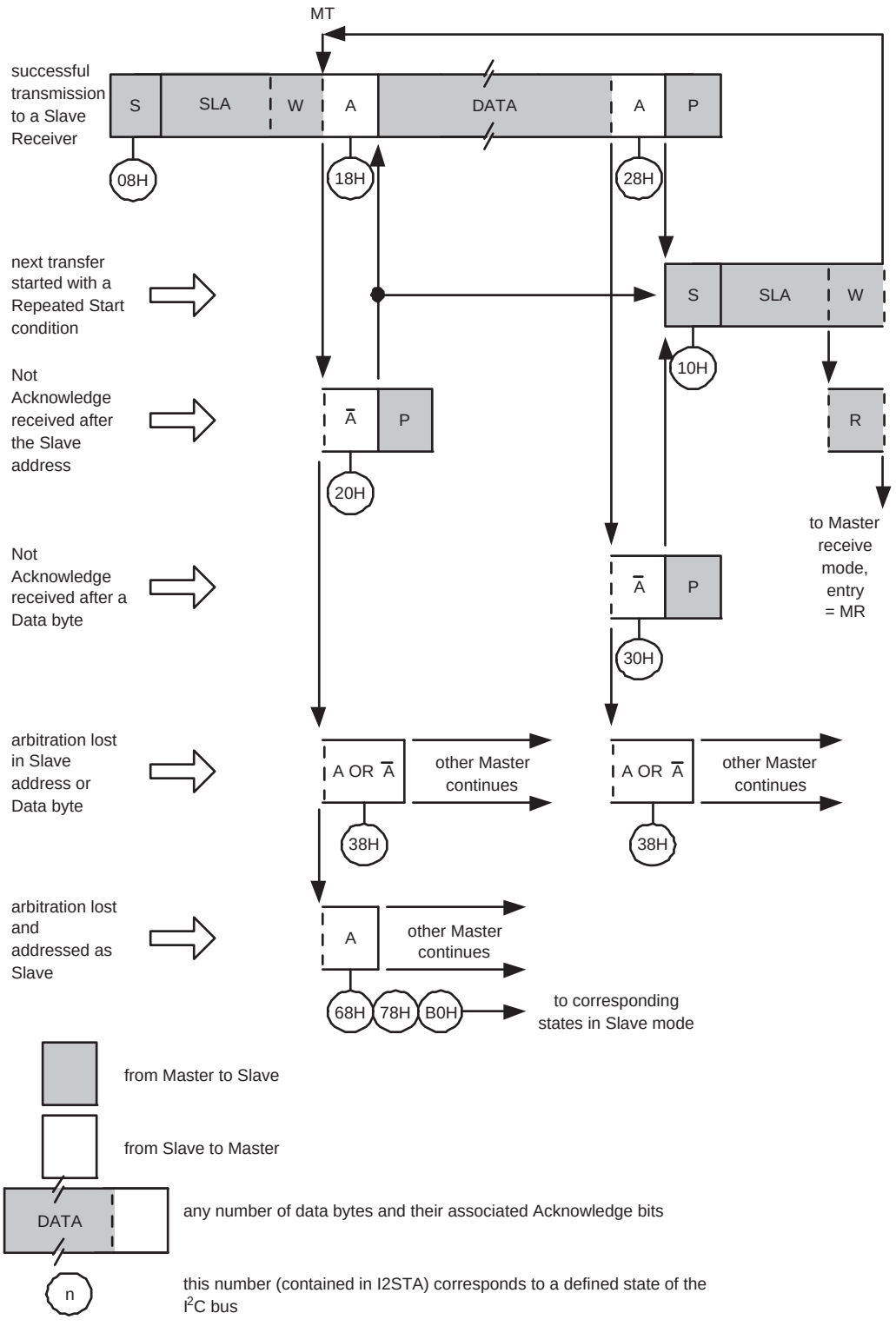


Fig 34. Format and States in the Master Transmitter mode

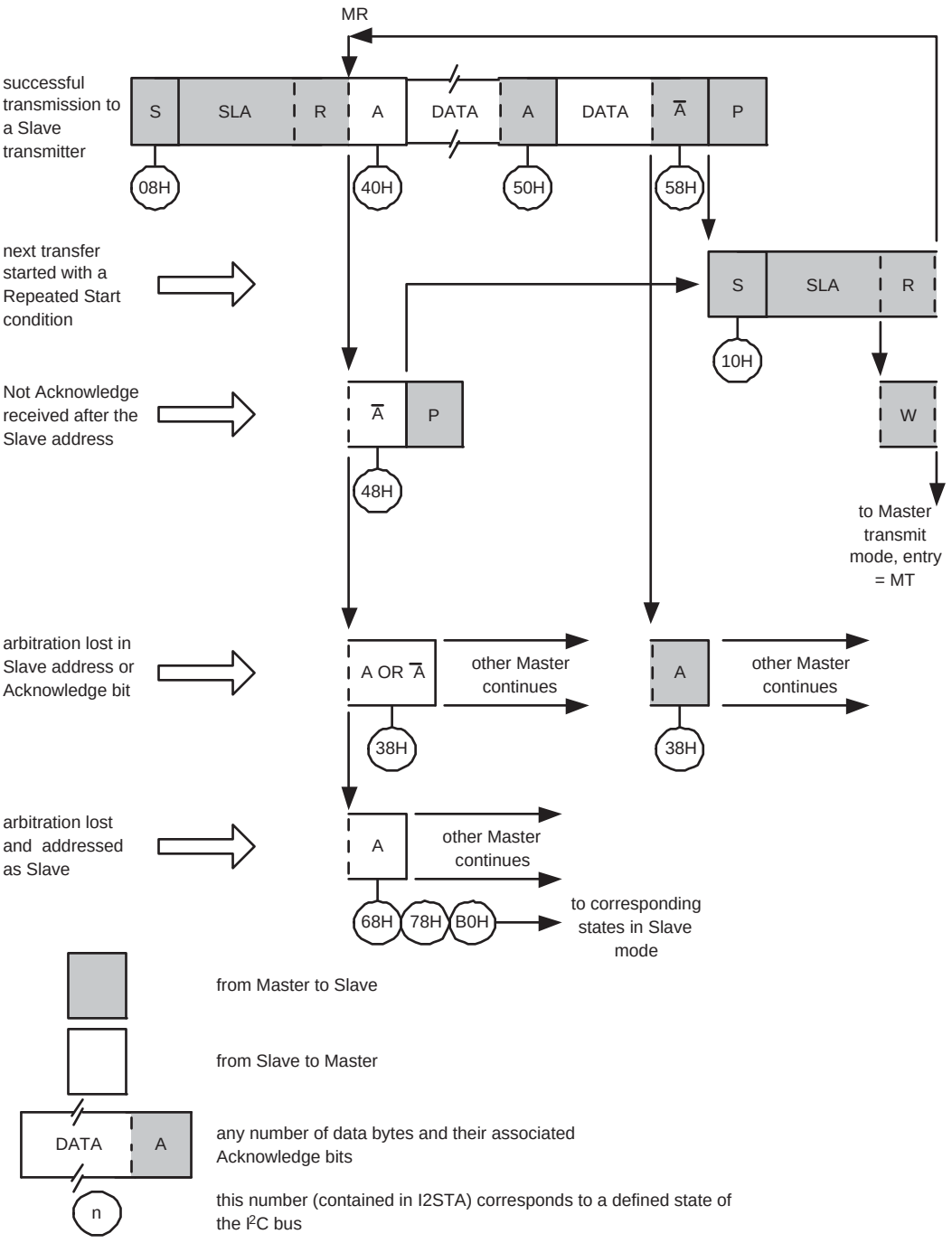


Fig 35. Format and States in the Master Receiver mode

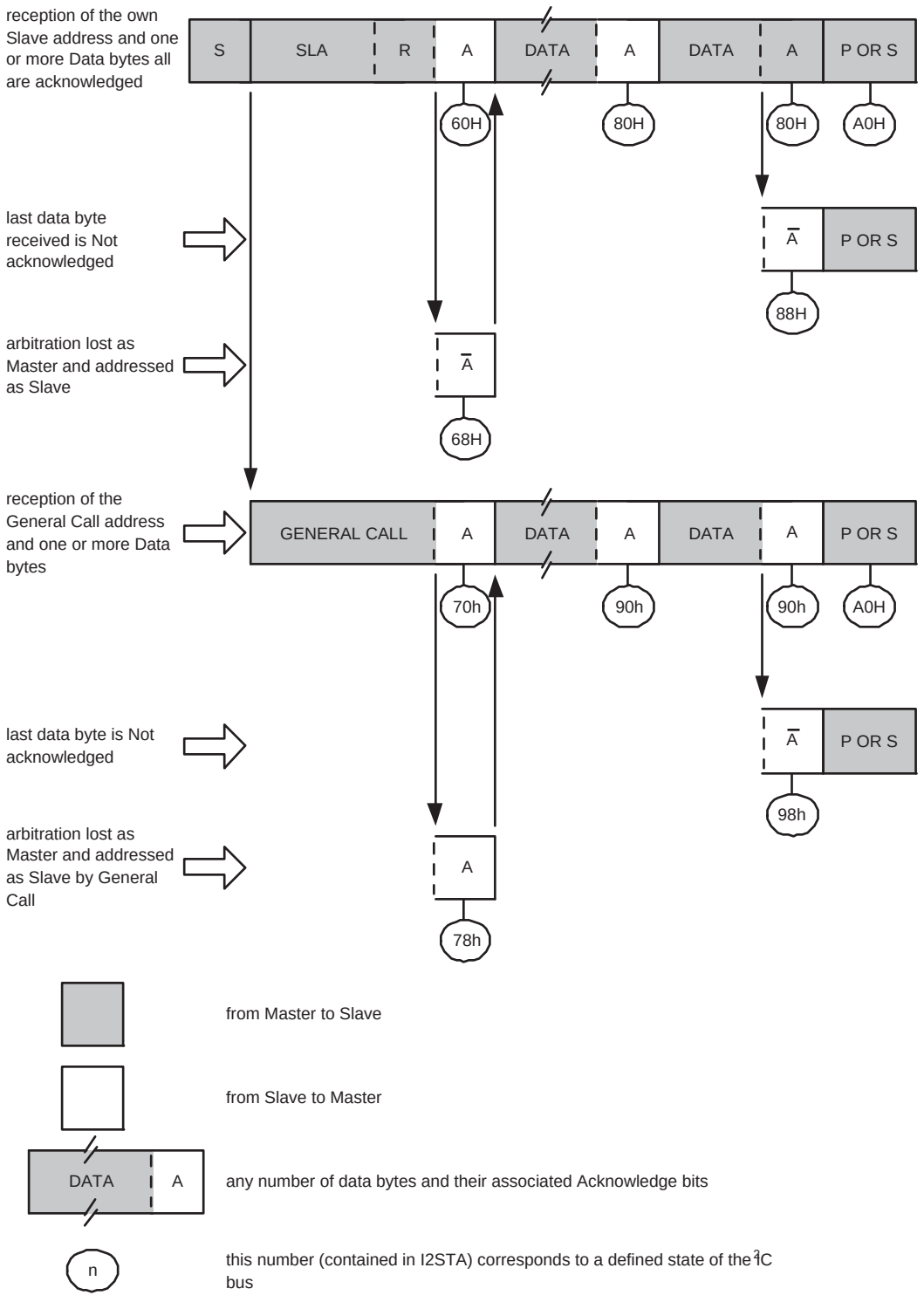


Fig 36. Format and States in the Slave Receiver mode

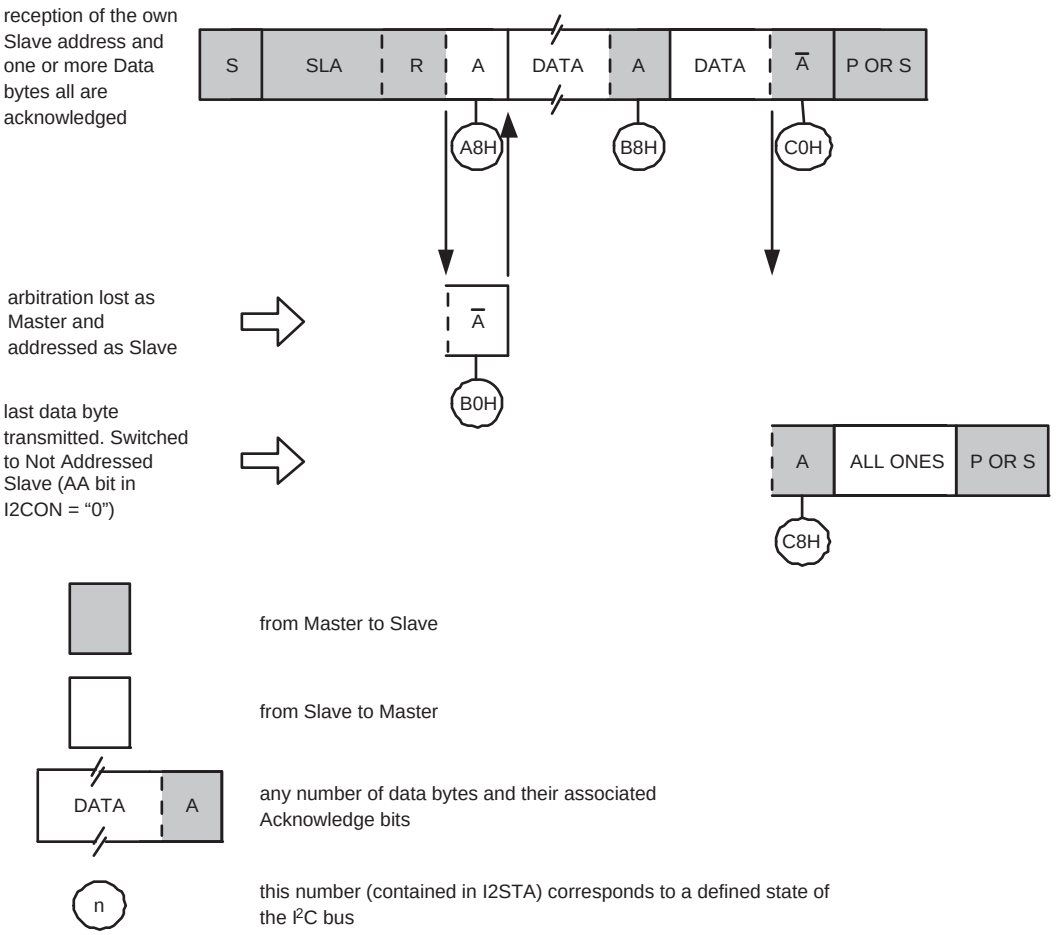


Fig 37. Format and States in the Slave Transmitter mode

9.4 Slave Transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 11–37). Data transfer is initialized as in the slave receiver mode. When I2ADR and I2CON have been initialized, the I2C block waits until it is addressed by its own slave address followed by the data direction bit which must be "1" (R) for the I2C 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 11–132. The slave transmitter mode may also be entered if arbitration is lost while the I2C block is in the master mode (see state 0xB0).

If the AA bit is reset during a transfer, the I2C block will transmit the last byte of the transfer and enter state 0xC0 or 0xC8. The I2C 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 I2C block does not respond to its own slave address or a general call address. However, the I2C-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 I2C block from the I2C-bus.

Table 129. Master Transmitter mode

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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 I²C 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	I²C-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 130. Master Receiver mode

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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 I²C block will be switched to MST/TRX mode.
0x38	Arbitration lost in NOT ACK bit.	No I2DAT action or	0	0	0	X	I²C-bus will be released; the I²C 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 131. Slave Receiver mode

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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 131. Slave Receiver mode

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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 132. Slave Transmitter mode

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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	01	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^2C hardware state (see Table 11–133). These are discussed below.

9.6 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^2C block is not involved in a serial transfer.

9.7 I2STAT = 0x00

This status code indicates that a bus error has occurred during an I^2C serial transfer. A bus error is caused when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. A bus error may also be caused when external interference disturbs the internal I^2C block signals. When a bus error occurs, SI is set. To recover from a bus error, the STO flag must be set and SI must be cleared. This causes the I^2C block to enter the “not addressed” slave mode (a defined state) and to clear the STO flag (no other bits in I2CON are affected). The SDA and SCL lines are released (a STOP condition is not transmitted).

Table 133. Miscellaneous States

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C 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 I²C 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 I²C block is switched to the not addressed SLV mode. STO is reset.

9.8 Some special cases

The I²C hardware has facilities to handle the following special cases that may occur during a serial transfer:

9.9 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 11–38). Until this occurs, arbitration is not lost by either master since they were both transmitting the same data.

If the I²C hardware detects a repeated START condition on the I²C-bus before generating a repeated START condition itself, it will release the bus, and no interrupt request is generated. If another master frees the bus by generating a STOP condition, the I²C block will transmit a normal START condition (state 0x08), and a retry of the total serial data transfer can commence.

9.10 Data transfer after loss of arbitration

Arbitration may be lost in the master transmitter and master receiver modes (see Figure 11–32). Loss of arbitration is indicated by the following states in I2STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 11–34 and Figure 11–35).

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.

9.11 Forced access to the I²C-bus

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

If an uncontrolled source generates a superfluous START or masks a STOP condition, then the I²C-bus stays busy indefinitely. If the STA flag is set and bus access is not obtained within a reasonable amount of time, then a forced access to the I²C-bus is possible. This is achieved by setting the STO flag while the STA flag is still set. No STOP condition is transmitted. The I²C hardware behaves as if a STOP condition was received and is able to transmit a START condition. The STO flag is cleared by hardware (see Figure 34).

9.12 I²C-bus obstructed by a low level on SCL or SDA

An I²C-bus hang-up occurs if SDA or SCL is pulled LOW by an uncontrolled source. If the SCL line is obstructed (pulled LOW) by a device on the bus, no further serial transfer is possible, and the I²C hardware cannot resolve this type of problem. When this occurs, the problem must be resolved by the device that is pulling the SCL bus line LOW.

If the SDA line is obstructed by another device on the bus (e.g., a slave device out of bit synchronization), the problem can be solved by transmitting additional clock pulses on the SCL line (see Figure 11–40). The I²C hardware transmits additional clock pulses when the STA flag is set, but no START condition can be generated because the SDA line is pulled LOW while the I²C-bus is considered free. The I²C hardware attempts to generate a START condition after every two additional clock pulses on the SCL line. When the SDA line is eventually released, a normal START condition is transmitted, state 0x08 is entered, and the serial transfer continues.

If a forced bus access occurs or a repeated START condition is transmitted while SDA is obstructed (pulled LOW), the I²C hardware performs the same action as described above. In each case, state 0x08 is entered after a successful START condition is transmitted and normal serial transfer continues. Note that the CPU is not involved in solving these bus hang-up problems.

9.13 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 I²C hardware only reacts to a bus error when it is involved in a serial transfer either as a master or an addressed slave. When a bus error is detected, the I²C block immediately switches to the not addressed slave mode, releases the SDA and SCL lines, sets the interrupt flag, and loads the status register with 0x00. This status code may be used to vector to a state service routine which either attempts the aborted serial transfer again or simply recovers from the error condition as shown in Table 11–133.

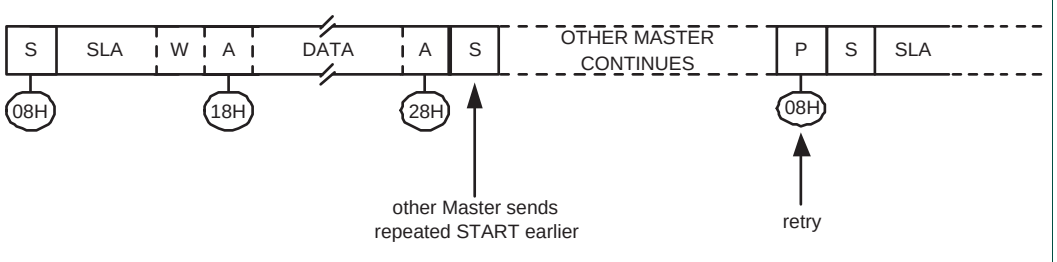


Fig 38. Simultaneous repeated START conditions from two masters

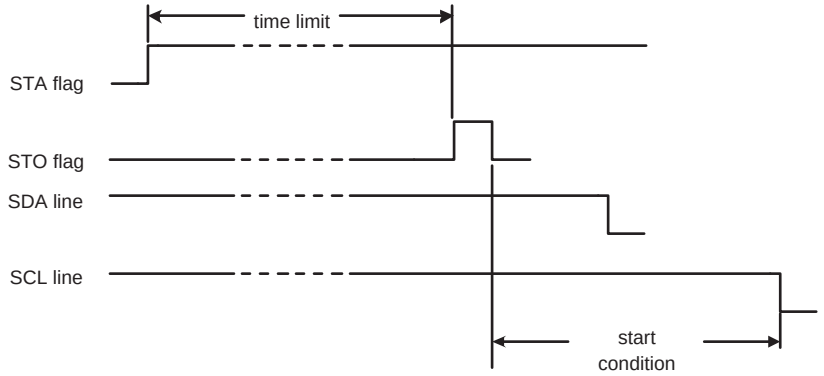


Fig 39. Forced access to a busy I2C-bus

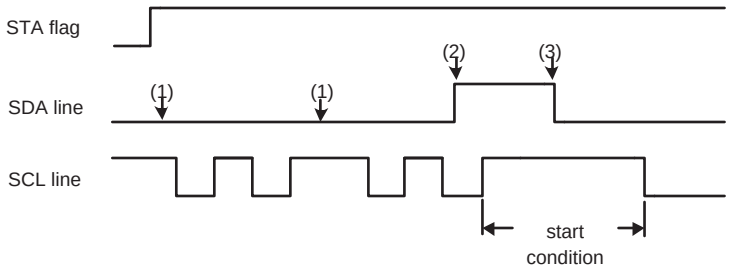


Fig 40. Recovering from a bus obstruction caused by a low level on SDA

9.14 I2C State service routines

This section provides examples of operations that must be performed by various I2C state service routines. This includes:

- Initialization of the I2C block after a Reset.
- I2C Interrupt Service
- The 26 state service routines providing support for all four I2C operating modes.

9.15 Initialization

In the initialization example, the I2C 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 I2C 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 I2C hardware now begins checking the I2C-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.

9.16 I2C interrupt service

When the I2C interrupt is entered, I2STAT contains a status code which identifies one of the 26 state services to be executed.

9.17 The State service routines

Each state routine is part of the I2C interrupt routine and handles one of the 26 states.

9.18 Adapting State services to an application

The state service examples show the typical actions that must be performed in response to the 26 I2C state codes. If one or more of the four I2C 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 I2C operations, in order to trap an inoperative bus or a lost service routine.

10. Software example

10.1 Initialization routine

Example to initialize I2C Interface as a Slave and/or Master.

1. Load I2ADR with own Slave Address, enable general call recognition if needed.
2. Enable I2C 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 I2C interrupt routine

Determine the I2C state and which state routine will be used to handle it.

1. Read the I2C status from I2STA.
2. Use the status value to branch to one of 26 possible state routines.

10.5 Non mode specific States

10.6 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

10.7 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.

10.8 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

10.9 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

10.10 Master Transmitter States

10.11 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

10.12 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

10.13 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

10.14 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

10.15 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

10.16 Master Receive States

10.17 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

10.18 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

10.19 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

10.20 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

10.21 Slave Receiver States

10.22 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

10.23 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.

10.24 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

10.25 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

10.26 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

10.27 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

10.28 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

10.29 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

10.30 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

10.31 Slave Transmitter States

10.32 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

10.33 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

10.34 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

10.35 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.

10.36 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

1. How to read this chapter

The following features and registers are available **in LPC2104/01, LPC2105/01, and LPC2106/01 only**:

- Transfer width selectable 8 bit to 16 bit (see Table 12–137, bits 11:8).

  Remark: The transfer width is fixed (8 bit) on LPC2104/05/06 and LPC2104/05/06/00 parts.
- SSEL pin can be used for a different function when the SPI interface is used in Master mode only (see Table 12–135).
- SPI can be configured as SSP interface (see Section 13–2).

2. Basic configuration

The SPI peripheral is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCSPI.

   Remark: On reset, SPI is enabled (PCSPI = 1). On LPC2104/05/06/01 only, the SPI shares its pin with the SSP interface. The SSP interface is disabled on reset. To use the SPI, the SSP interface must be disabled in the PCONP register (Table 3–27), PCSSP = 0.
2. Clock: In Master mode, the SPI clock must be scaled down (see Section 12–6.4).
3. Pins: Select SPI pins in registers PINSEL0/1 (see Section 7–2). See Table 12–135 for behavior of the SSEL pin.
4. Interrupts: To enable SPI interrupts, see Section 12–6.5. Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

   Remark: On LPC2104/05/06/01, the SPI shares an interrupt line with the SSP interface in the VIC.

3. Features

- Compliant with Serial Peripheral Interface (SPI) specification
- Synchronous, serial, and full duplex communication
- Combined SPI master and slave
- Maximum data bit rate of one eighth of the input clock rate
- 8 bit to 16 bit per transfer

4. Description

4.1 SPI overview

SPI is a full duplex serial interface. It can handle multiple masters and slaves being connected to a given bus. Only a single master and a single slave can communicate on the interface during a given data transfer.

4.2 SPI data transfers

Figure 12–41 is a timing diagram that illustrates the four different data transfer formats that are available with the SPI. This timing diagram illustrates a single 8 bit data transfer. The first thing you should notice in this timing diagram is that it is divided into three horizontal parts. The first part describes the SCK and SSEL signals. The second part describes the MOSI and MISO signals when the CPHA variable is 0. The third part describes the MOSI and MISO signals when the CPHA variable is 1.

In the first part of the timing diagram, note two points. First, the SPI is illustrated with CPOL set to both 0 and 1. The second point to note is the activation and de-activation of the SSEL signal. When CPHA = 0, the SSEL signal will always go inactive between data transfers. This is not guaranteed when CPHA = 1 (the signal can remain active).

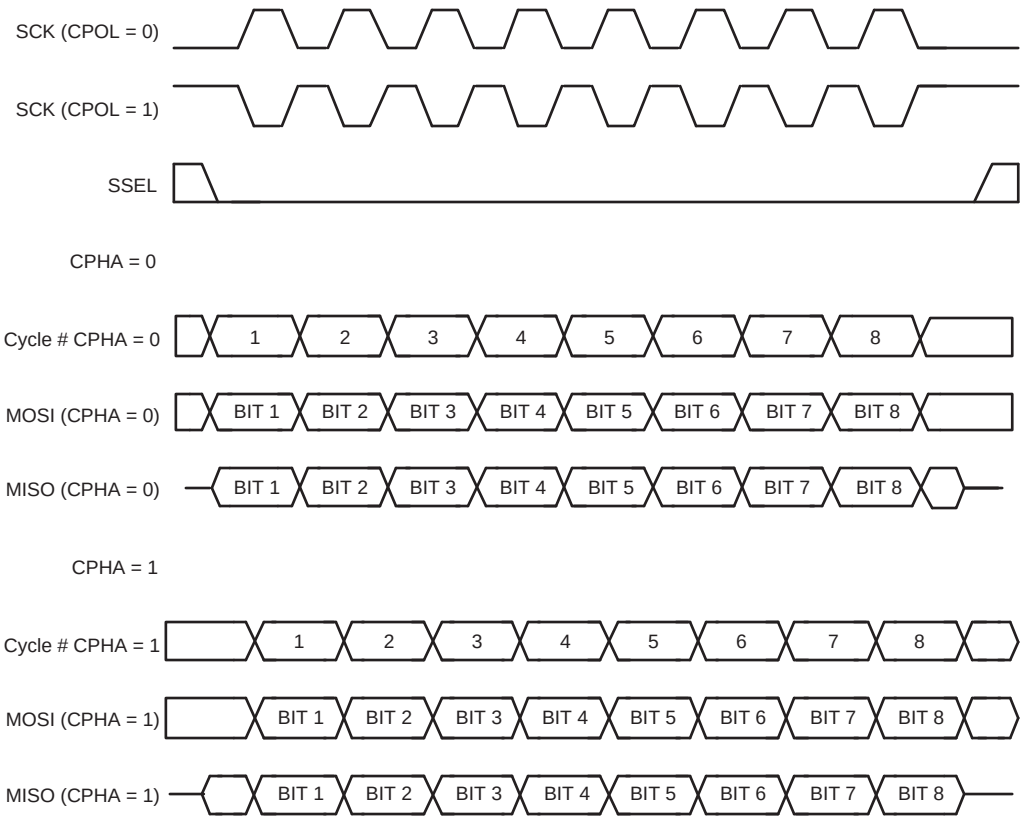


Fig 41. SPI data transfer format (CPHA = 0 and CPHA = 1)

The data and clock phase relationships are summarized in Table 12–134. This table summarizes the following for each setting of CPOL and CPHA.

- When the first data bit is driven
- When all other data bits are driven
- When data is sampled

Table 134. SPI data to clock phase relationship

CPOL and CPHA settings	First data driven	Other data driven	Data sampled
CPOL = 0, CPHA = 0	Prior to first SCK rising edge	SCK falling edge	SCK rising edge
CPOL = 0, CPHA = 1	First SCK rising edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 0	Prior to first SCK falling edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 1	First SCK falling edge	SCK falling edge	SCK rising edge

The definition of when an 8 bit transfer starts and stops is dependent on whether a device is a master or a slave, and the setting of the CPHA variable.

When a device is a master, the start of a transfer is indicated by the master having a byte of data that is ready to be transmitted. At this point, the master can activate the clock, and begin the transfer. The transfer ends when the last clock cycle of the transfer is complete.

When a device is a slave, and CPHA is set to 0, the transfer starts when the SSEL signal goes active, and ends when SSEL goes inactive. When a device is a slave, and CPHA is set to 1, the transfer starts on the first clock edge when the slave is selected, and ends on the last clock edge where data is sampled.

4.3 SPI peripheral details

4.3.1 General information

There are four registers that control the SPI peripheral. They are described in detail in Section 12–6 “Register description” on page 163.

The SPI control register contains a number of programmable bits used to control the function of the SPI block. The settings for this register must be set up prior to a given data transfer taking place.

The SPI status register contains read only bits that are used to monitor the status of the SPI interface, including normal functions, and exception conditions. The primary purpose of this register is to detect completion of a data transfer. This is indicated by the SPIF bit. The remaining bits in the register are exception condition indicators. These exceptions will be described later in this section.

The SPI data register is used to provide the transmit and receive data bytes. An internal shift register in the SPI block logic is used for the actual transmission and reception of the serial data. Data is written to the SPI data register for the transmit case. There is no buffer between the data register and the internal shift register. A write to the data register goes directly into the internal shift register. Therefore, data should only be written to this register when a transmit is not currently in progress. Read data is buffered. When a transfer is complete, the receive data is transferred to a single byte data buffer, where it is later read. A read of the SPI data register returns the value of the read data buffer.

The SPI clock counter register controls the clock rate when the SPI block is in master mode. This needs to be set prior to a transfer taking place, when the SPI block is a master. This register has no function when the SPI block is a slave.

The I/Os for this implementation of SPI are standard CMOS I/Os. The open drain SPI option is not implemented in this design. When a device is set up to be a slave, its I/Os are only active when it is selected by the SSEL signal being active.

4.3.2 Master operation

The following sequence describes how to process a data transfer with the SPI block when it is set up as the master. This process assumes that any prior data transfer has already completed.

1. Set the SPI clock counter register to the desired clock rate.
2. Set the SPI control register to the desired settings.
3. Write the data to transmitted to the SPI data register. This write starts the SPI data transfer.
4. Wait for the SPIF bit in the SPI status register to be set to 1. The SPIF bit will be set after the last cycle of the SPI data transfer.
5. Read the SPI status register.
6. Read the received data from the SPI data register (optional).
7. Go to step 3 if more data is required to transmit.

Note: A read or write of the SPI data register is required in order to clear the SPIF status bit. Therefore, if the optional read of the SPI data register does not take place, a write to this register is required in order to clear the SPIF status bit.

4.3.3 Slave operation

The following sequence describes how to process a data transfer with the SPI block when it is set up as slave. This process assumes that any prior data transfer has already completed. It is required that the system clock driving the SPI logic be at least 8X faster than the SPI.

1. Set the SPI control register to the desired settings.
2. Write the data to transmitted to the SPI data register (optional). Note that this can only be done when a slave SPI transfer is not in progress.
3. Wait for the SPIF bit in the SPI status register to be set to 1. The SPIF bit will be set after the last sampling clock edge of the SPI data transfer.
4. Read the SPI status register.
5. Read the received data from the SPI data register (optional).
6. Go to step 2 if more data is required to transmit.

Note: A read or write of the SPI data register is required in order to clear the SPIF status bit. Therefore, at least one of the optional reads or writes of the SPI data register must take place, in order to clear the SPIF status bit.

4.3.4 Exception conditions

4.3.4.1 Read overrun

A read overrun occurs when the SPI block internal read buffer contains data that has not been read by the processor, and a new transfer has completed. The read buffer containing valid data is indicated by the SPIF bit in the status register being active. When a transfer completes, the SPI block needs to move the received data to the read buffer. If the SPIF bit is active (the read buffer is full), the new receive data will be lost, and the read overrun (ROVR) bit in the status register will be activated.

4.3.4.2 Write collision

As stated previously, there is no write buffer between the SPI block bus interface, and the internal shift register. As a result, data must not be written to the SPI data register when a SPI data transfer is currently in progress. The time frame where data cannot be written to the SPI data register is from when the transfer starts, until after the status register has been read when the SPIF status is active. If the SPI data register is written in this time frame, the write data will be lost, and the write collision (WCOL) bit in the status register will be activated.

4.3.4.3 Mode fault

The SSEL signal must always be inactive when the SPI block is a master. If the SSEL signal goes active, when the SPI block is a master, this indicates another master has selected the device to be a slave. This condition is known as a mode fault. When a mode fault is detected, the mode fault (MODF) bit in the status register will be activated, the SPI signal drivers will be de-activated, and the SPI mode will be changed to be a slave.

4.3.4.4 Slave abort

A slave transfer is considered to be aborted, if the SSEL signal goes inactive before the transfer is complete. In the event of a slave abort, the transmit and receive data for the transfer that was in progress are lost, and the slave abort (ABRT) bit in the status register will be activated.

5. Pin description

Table 135. SPI pin description

Pin name	Type	Pin description
SCK	Input/ Output	**Serial Clock.** The SPI is a clock signal used to synchronize the transfer of data across the SPI interface. The SPI is always driven by the master and received by the slave. The clock is programmable to be active high or active low. The SPI is only active during a data transfer. Any other time, it is either in its inactive state, or tri-stated.
SSEL	Input	**Slave Select.** The SPI slave select signal is an active low signal that indicates which slave is currently selected to participate in a data transfer. Each slave has its own unique slave select signal input. The SSEL must be low before data transactions begin and normally stays low for the duration of the transaction. If the SSEL signal goes high any time during a data transfer, the transfer is considered to be aborted. In this event, the slave returns to idle, and any data that was received is thrown away. There are no other indications of this exception. This signal is not directly driven by the master. It could be driven by a simple general purpose I/O under software control. **Remark: LPC2104/05/06 and LPC2104/05/06/00 configured to operate as a SPI master MUST select SSEL functionality on P0.7 and have HIGH level on this pin in order to act as a master.** For all LPC2104/05/06/01 parts, the SSEL pin can be used for a different function when the SPI interface is only used in Master mode. For example, the pin hosting the SSEL function can be configured as an output digital GPIO pin and can be used to select one of the SPI slaves.
MISO	Input/ Output	**Master In Slave Out.** The MISO signal is a unidirectional signal used to transfer serial data from the slave to the master. When a device is a slave, serial data is output on this signal. When a device is a master, serial data is input on this signal. When a slave device is not selected, the slave drives the signal high impedance.
MOSI	Input/ Output	**Master Out Slave In.** The MOSI signal is a unidirectional signal used to transfer serial data from the master to the slave. When a device is a master, serial data is output on this signal. When a device is a slave, serial data is input on this signal.

6. Register description

The SPI contains 5 registers as shown in Table 12–136. All registers are byte, half word and word accessible.

Table 136. SPI register map

Name	Description	Access	Reset value[1]	Address
SPCR	SPI Control Register. This register controls the operation of the SPI.	R/W	0x0000	0xE002 0000
SPSR	SPI Status Register. This register shows the status of the SPI.	RO	0x00	0xE002 0004
SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI by writing to this register. Data received by the SPI can be read from this register.	R/W	0x0000	0xE002 0008
SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK.	R/W	0x00	0xE002 000C
SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	R/W	0x00	0xE002 001C

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

6.1 SPI Control Register (SPCR - 0xE002 0000)

The SPCR register controls the operation of the SPI as per the configuration bits setting.

Table 137. SPI Control Register (SPCR - address 0xE002 0000) bit description

Bit	Symbol	Value	Description	Reset value
1:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2[1]	BitEnable	0	The SPI controller sends and receives 8 bits of data per transfer.	0
		1	The SPI controller sends and receives the number of bits selected by bits 11:8.	
3	CPHA	0	Clock phase control determines the relationship between the data and the clock on SPI transfers, and controls when a slave transfer is defined as starting and ending. Data is sampled on the first clock edge of SCK. A transfer starts and ends with activation and deactivation of the SSEL signal.	0
		1	Data is sampled on the second clock edge of the SCK. A transfer starts with the first clock edge, and ends with the last sampling edge when the SSEL signal is active.	
4	CPOL	0	Clock polarity control. SCK is active high.	0
		1	SCK is active low.	
5	MSTR	0	Master mode select. The SPI operates in Slave mode.	0
		1	The SPI operates in Master mode.	
6	LSBF	0	LSB First controls which direction each byte is shifted when transferred. SPI data is transferred MSB (bit 7) first.	0
		1	SPI data is transferred LSB (bit 0) first.	
7	SPIE	0	Serial peripheral interrupt enable. SPI interrupts are inhibited.	0
		1	A hardware interrupt is generated each time the SPIF or MODF bits are activated.	

Table 137. SPI Control Register (SPCR - address 0xE002 0000) bit description

Bit	Symbol	Value	Description	Reset value
11:8[1]	BITS		When bit 2 of this register is 1, this field controls the number of bits per transfer:	0000
		1000	8 bits per transfer	
		1001	9 bits per transfer	
		1010	10 bits per transfer	
		1011	11 bits per transfer	
		1100	12 bits per transfer	
		1101	13 bits per transfer	
		1110	14 bits per transfer	
		1111	15 bits per transfer	
		0000	16 bits per transfer	
15:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] Available in LPC2104/05/06/01 only. These bits are reserved for LPC2104/05/06 and LPC2104/05/06/00.

6.2 SPI Status Register (SPSR - 0xE002 0004)

The SPSR register controls the operation of the SPI as per the configuration bits setting.

Table 138. SPI Status Register (SPSR - address 0xE002 0004) bit description

Bit	Symbol	Description	Reset value
2:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	ABRT	Slave abort. When 1, this bit indicates that a slave abort has occurred. This bit is cleared by reading this register.	0
4	MODF	Mode fault. when 1, this bit indicates that a Mode fault error has occurred. This bit is cleared by reading this register, then writing the SPI control register.	0
5	ROVR	Read overrun. When 1, this bit indicates that a read overrun has occurred. This bit is cleared by reading this register.	0
6	WCOL	Write collision. When 1, this bit indicates that a write collision has occurred. This bit is cleared by reading this register, then accessing the SPI data register.	0
7	SPIF	SPI transfer complete flag. When 1, this bit indicates when a SPI data transfer is complete. When a master, this bit is set at the end of the last cycle of the transfer. When a slave, this bit is set on the last data sampling edge of the SCK. This bit is cleared by first reading this register, then accessing the SPI data register. **Note:** This is not the SPI interrupt flag. This flag is found in the SPINT register.	0

6.3 SPI Data Register (SPDR - 0xE002 0008)

This bi-directional data register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI by writing to this register. Data received by the SPI can be read from this register. When a master, a write to this register will start a SPI data transfer. Writes to this register will be blocked from when a data transfer starts to when the SPIF status bit is set, and the status register has not been read.

Table 139. SPI Data Register (SPDR - address 0xE002 0008) bit description

Bit	Symbol	Description	Reset value
15:0	Data	SPI Bi-directional data port.	0

6.4 SPI Clock Counter Register (SPCCR - 0xE002 000C)

This register controls the frequency of a master's SCK. The register indicates the number of SPI peripheral clock cycles that make up an SPI clock.

In Master mode, this register must be an even number greater than or equal to 8. Violations of this can result in unpredictable behavior. The SPI SCK rate may be calculated as: PCLK / SnSPCCR value. The PCLK rate is CCLK /APB divider rate as determined by the APBDIV register contents (see Section 3–12.2).

In Slave mode, the SPI clock rate provided by the master must not exceed 1/8 of the peripheral clock. The content of the S0SPCCR register is not relevant.

Table 140. SPI Clock Counter Register (SPCCR - address 0xE002 000C) bit description

Bit	Symbol	Description	Reset value
7:0	Counter	SPI Clock counter setting.	0x00

6.5 SPI Interrupt Register (SPINT - 0xE002 001C)

This register contains the interrupt flag for the SPI interface.

Table 141. SPI Interrupt Register (SPINT - address 0xE002 001C) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:1	SPI Interrupt	SPI interrupt flag. Set by the SPI interface to generate an interrupt. Cleared by writing a 1 to this bit. **Note:** This bit will be set once when SPIE = 1 and at least one of SPIF and MODF bits changes from 0 to 1. However, only when the SPI Interrupt bit is set and SPI Interrupt is enabled in the VIC, SPI based interrupt can be processed by interrupt handling software.	0

7. Architecture

The block diagram of the SPI interface is shown in the Figure 12–42.

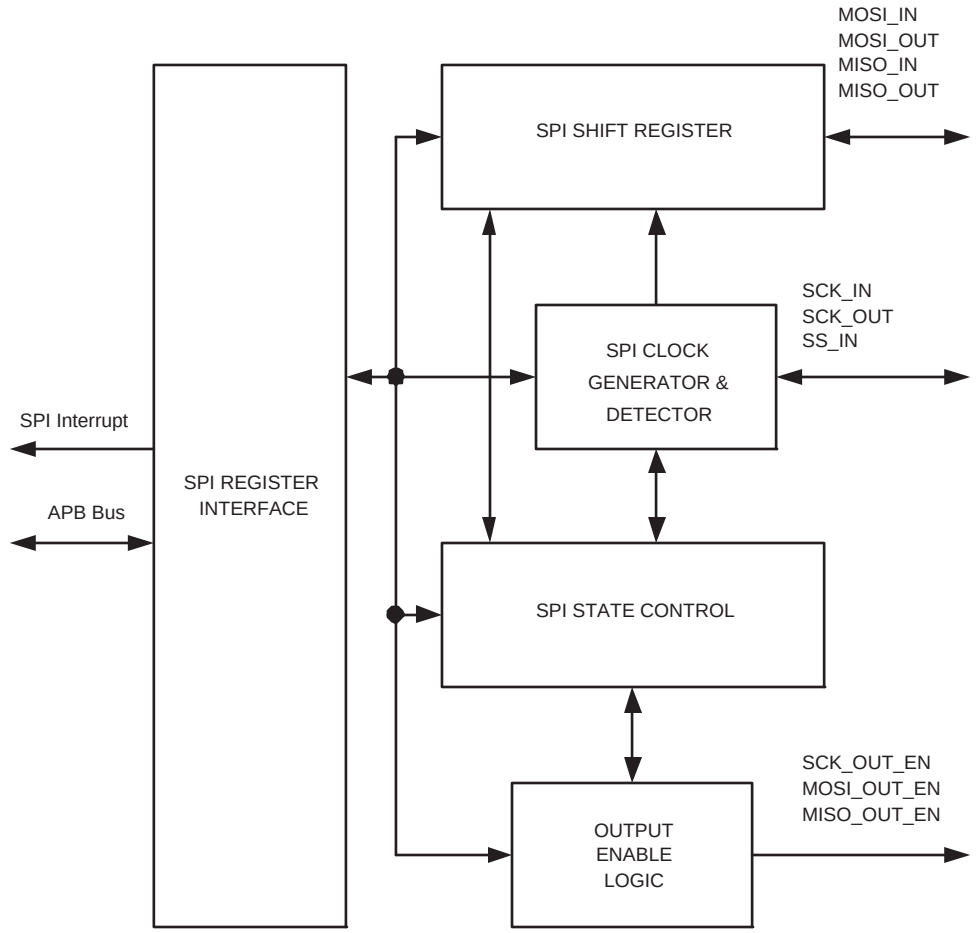


Fig 42. SPI block diagram

1. How to read this chapter

The SSP interface is available on **LPC2104/01, LPC2105/01, and LPC2106/01 only**.

2. Basic configuration

The SSP peripheral is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCSSP.

   Remark: On reset, SSP is disabled (PCSSP = 0). The SSP interface shares its pins with the SPI. To use the SSP interface, the SPI must first be disabled in the PCONP register (Table 3–27), PCSPI = 0. Also see Section 13–5.
2. Clock: In Master mode, the SSP clock must be scaled (see Table 13–148).
3. Pins: Select SSP pins (same as SPI pins) in registers PINSEL0/1 (see Section 7–2).
4. Interrupts: To enable SSP interrupts, see Table 13–149. Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

   Remark: For LPC2104/05/06/01, the SSP shares an interrupt line with the SPI in the VIC.

3. Features

- Compatible with Motorola SPI, 4-wire TI SSI, and National Semiconductor Microwire buses
- Synchronous serial communication
- Master or slave operation
- 8-frame FIFOs for both transmit and receive
- 4 to 16 bit frame

4. Description

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

5. SSP usage notes

Because the SSP and SPI peripherals share the same physical pins it is not possible to have both of these two peripherals active at the same time. Bit 8 (PCSPI) and bit 21 (PCSSP) in the PCONP register (see Section 3–10.3) control the activity of the SPI and SSP module respectively. The corresponding peripheral is enabled when its control bit is

1, and it is disabled when the control bit is 0. After power-on reset, SPI is enabled, maintaining the backward compatibility with other NXP LPC2000 microcontrollers. Any attempt to write 1 to PSPI0 and PSSP bits at the same time will result in PCSPI = 1 and PCSSP = 0.

To switch on the fly from SPI to SSP and back, first disable the active peripheral's interrupt(s), both in the peripheral's and VIC's registers. Next, clear all pending interrupt flags (if any set). Only then, the currently enabled peripheral can be turned off in the PCONP register. After this, the other serial interface can be enabled.

It is important to disable the currently used peripheral by clearing its bit in the PCONP register only at the very end of the peripheral's shut-down procedure. Otherwise, having 0 in a bit in PCONP will disable all clocks from coming into the peripheral controlled by that bit. Then, reading from the peripheral's registers will not yield valid data, and write and/or modify access will be banned, i.e. no content can be changed. Consequently, if any of the interrupt triggering flags are left active in the peripheral's register(s) when the peripheral is disabled via the PCONP, the invoked ISR may not be able to successfully service pending interrupt, and the same interrupt may keep overloading the microcontroller even though its peripheral is disabled.

6. Pin description

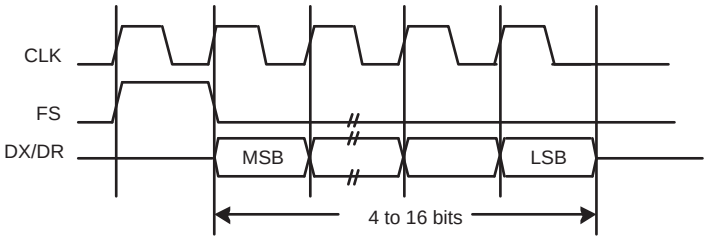
Table 142. SSP pin descriptions

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

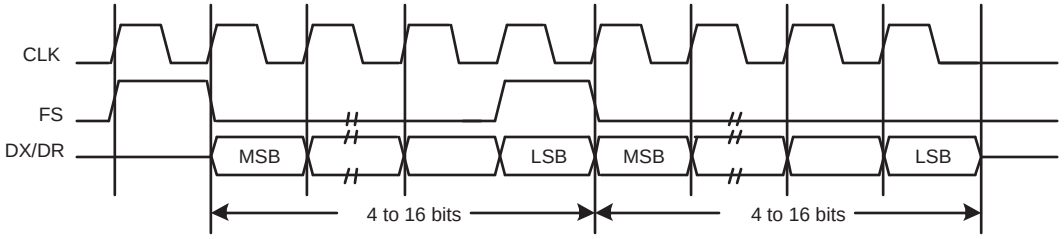
7. Bus description

7.1 Texas Instruments synchronous serial frame format

Figure 13–43 shows the 4-wire Texas Instruments synchronous serial frame format supported by the SSP module.



a. Single frame transfer



b. Continuous/back-to-back frames transfer

Fig 43. Texas Instruments synchronous serial frame format: a) single frame transfer and b) continuous/back-to-back two frames.

For device configured as a master in this mode, CLK and FS are forced LOW, and the transmit data line DX is tristated whenever the SSP is idle. Once the bottom entry of the transmit FIFO contains data, FS is pulsed HIGH for one CLK period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of CLK, the MSB of the 4 to 16-bit data frame is shifted out on the DX pin. Likewise, the MSB of the received data is shifted onto the DR pin by the off-chip serial slave device.

Both the SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each CLK. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of CLK after the LSB has been latched.

7.2 SPI frame format

The SPI interface is a four-wire interface where the SSEL signal behaves as a slave select. The main feature of the SPI format is that the inactive state and phase of the SCK signal are programmable through the CPOL and CPHA bits within the SSPCR0 control register.

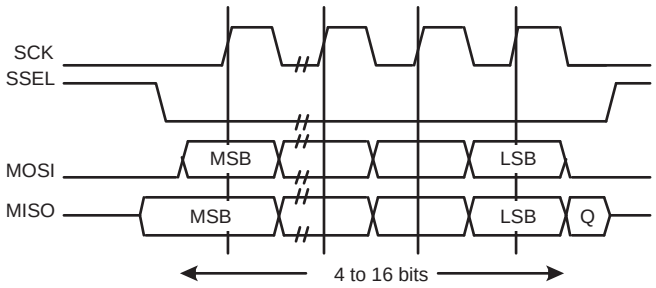
7.2.1 Clock Polarity (CPOL) and Phase (CPHA) Control

When the CPOL clock polarity control bit is LOW, it produces a steady state low value on the SCK pin. If the CPOL clock polarity control bit is HIGH, a steady state high value is placed on the CLK pin when data is not being transferred.

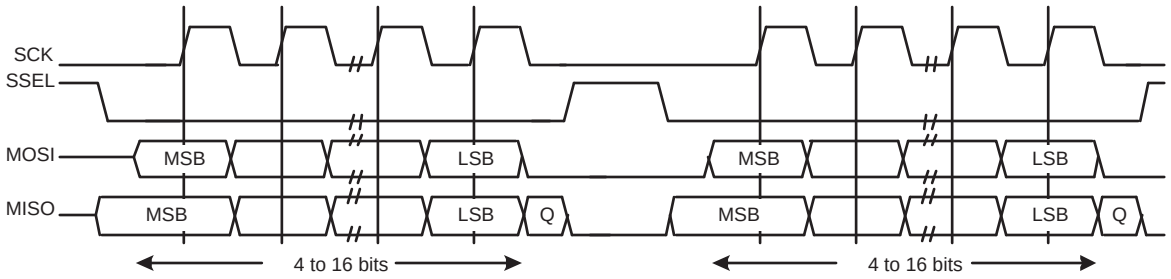
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is LOW, data is captured on the first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

7.2.2 SPI Format with CPOL = 0,CPHA = 0

Single and continuous transmission signal sequences for SPI format with CPOL = 0, CPHA = 0 are shown in Figure 13–44.



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



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

Fig 44. Motorola SPI frame format with CPOL=0 and CPHA=0 (a) single transfer and b) continuous transfer)

In this configuration, during idle periods:

- The CLK signal is forced LOW
- SSEL is forced HIGH
- The transmit MOSI/MISO pad is in high impedance

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

One half SCK period later, valid master data is transferred to the MOSI pin. Now that both the master and slave data have been set, the SCK master clock pin goes HIGH after one further half SCK period.

The data is now captured on the rising and propagated on the falling edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

7.2.3 SPI format with CPOL = 0,CPHA = 1

The transfer signal sequence for SPI format with CPOL = 0, CPHA = 1 is shown in Figure 13–45, which covers both single and continuous transfers.

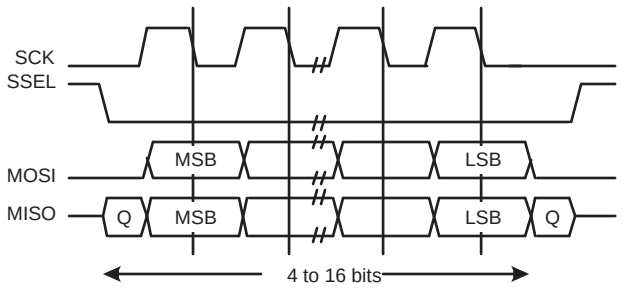


Fig 45. SPI frame format with CPOL=0 and CPHA=1

In this configuration, during idle periods:

- The CLK signal is forced LOW
- SSEL is forced HIGH
- The transmit MOSI/MISO pad is in high impedance

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master's MOSI pin is enabled. After a further one half SCK period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the SCK is enabled with a rising edge transition.

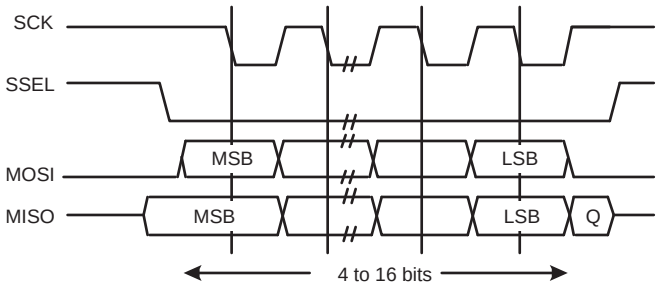
Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

In the case of a single word transfer, after all bits have been transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

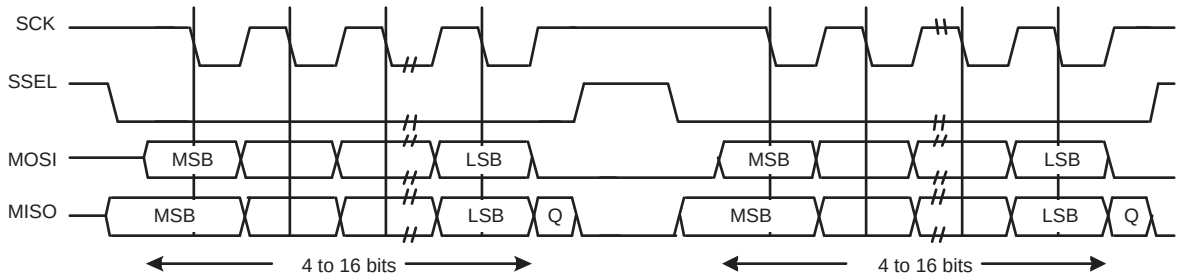
For continuous back-to-back transfers, the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

7.2.4 SPI format with CPOL = 1,CPHA = 0

Single and continuous transmission signal sequences for SPI format with CPOL=1, CPHA=0 are shown in Figure 13–46.



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



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

Fig 46. SPI frame format with CPOL = 1 and CPHA = 0 (a) single and b) continuous transfer)

In this configuration, during idle periods:

- The CLK signal is forced HIGH
- SSEL is forced HIGH
- The transmit MOSI/MISO pad is in high impedance

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW, which causes slave data to be immediately transferred onto the MISO line of the master. Master's MOSI pin is enabled.

One half period later, valid master data is transferred to the MOSI line. Now that both the master and slave data have been set, the SCK master clock pin becomes LOW after one further half SCK period. This means that data is captured on the falling edges and be propagated on the rising edges of the SCK signal.

In the case of a single word transmission, after all bits of the data word are transferred, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSEL signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the CPHA bit is logic zero. Therefore the master device must raise the SSEL pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSEL pin is returned to its idle state one SCK period after the last bit has been captured.

7.2.5 SPI format with CPOL = 1,CPHA = 1

The transfer signal sequence for SPI format with CPOL = 1, CPHA = 1 is shown in Figure 13–47, which covers both single and continuous transfers.

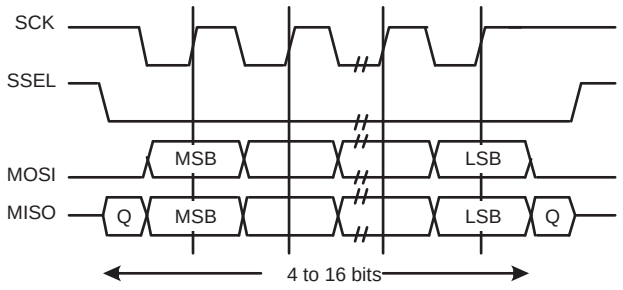


Fig 47. SPI frame format with CPOL = 1 and CPHA = 1

In this configuration, during idle periods:

- The CLK signal is forced HIGH
- SSEL is forced HIGH
- The transmit MOSI/MISO pad is in high impedance

If the SSP is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSEL master signal being driven LOW. Master's MOSI is enabled. After a further one half SCK period, both master and slave data are enabled onto their respective transmission lines. At the same time, the SCK is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the SCK signal.

After all bits have been transferred, in the case of a single word transmission, the SSEL line is returned to its idle HIGH state one SCK period after the last bit has been captured. For continuous back-to-back transmissions, the SSEL pins remains in its active LOW state, until the final bit of the last word has been captured, and then returns to its idle state as described above. In general, for continuous back-to-back transfers the SSEL pin is held LOW between successive data words and termination is the same as that of the single word transfer.

7.3 Semiconductor Microwire frame format

Figure 13–48 shows the Microwire frame format for a single frame. Figure 44 shows the same format when back-to-back frames are transmitted.

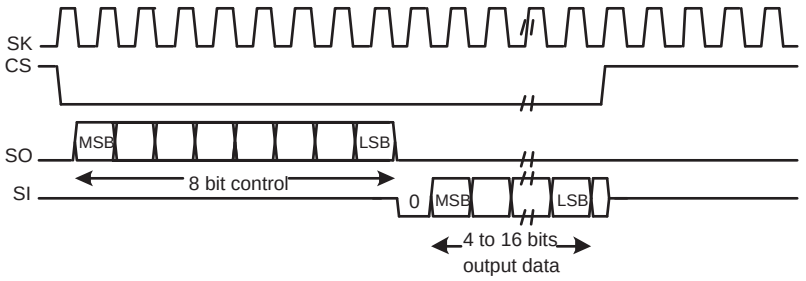


Fig 48. Microwire frame format (single transfer)

Microwire format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSP to the off-chip slave device. During this transmission, no incoming data is received by the SSP. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- The SK signal is forced LOW
- CS is forced HIGH
- The transmit data line SO is arbitrarily forced LOW

A transmission is triggered by writing a control byte to the transmit FIFO.The falling edge of CS causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the SO pin. CS remains LOW for the duration of the frame transmission. The SI pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SK. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSP. Each bit is driven onto SI line on the falling edge of SK. The SSP in turn latches each bit on the rising edge of SK. At the end of the frame, for single transfers, the CS signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

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

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

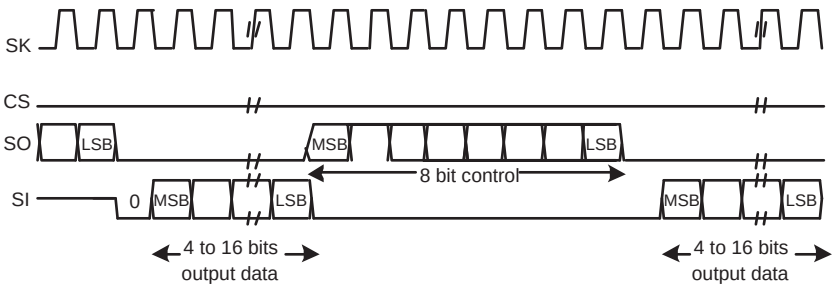


Fig 49. Microwire frame format (continuos transfers)

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

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

Figure 13–50 illustrates these setup and hold time requirements. With respect to the SK rising edge on which the first bit of receive data is to be sampled by the SSP slave, CS must have a setup of at least two times the period of SK on which the SSP operates. With respect to the SK rising edge previous to this edge, CS must have a hold of at least one SK period.

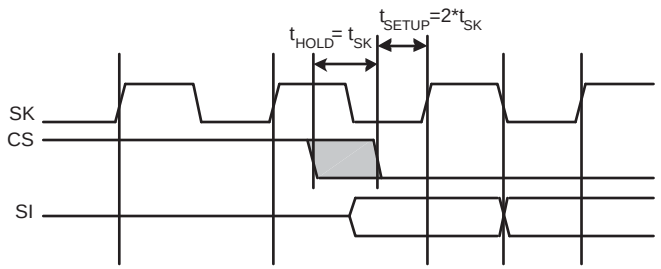


Fig 50. Microwire setup and hold details

8. Register description

The SSP contains 9 registers as shown in Table 13–143. All registers are byte, half word and word accessible.

Table 143. SSP Registers

Name	Description	Access	Reset value[1]	Address
SSPCR0	Control Register 0. Selects the serial clock rate, bus type, and data size.	R/W	0x0000	0xE005 C000
SSPCR1	Control Register 1. Selects master/slave and other modes.	R/W	0x00	0xE005 C004
SSPDR	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	R/W	0x0000	0xE005 C008
SSPSR	Status Register	RO	0x03	0xE005 C00C
SSPCPSR	Clock Prescale Register	R/W	0x00	0xE005 C010
SSPIMSC	Interrupt Mask Set and Clear Register	R/W	0x00	0xE005 C014
SSPRIS	Raw Interrupt Status Register	R/W	0x08	0xE005 C018
SSPMIS	Masked Interrupt Status Register	RO	0x00	0xE005 C01C
SSPICR	SSPICR Interrupt Clear Register	WO	NA	0xE005 C020

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

8.1 SSP Control Register 0 (SSPCR0 - 0xE005 C000)

This register controls the basic operation of the SSP controller.

Table 144: SSP Control Register 0 (SSPCR0 - address 0xE005 C000) bit description

Bit	Symbol	Value	Description	Reset value
3:0	DSS		Data Size Select. This field controls the number of bits transferred in each frame. Values 0000-0010 are not supported and should not be used.	0000
		0011	4 bit transfer	
		0100	5 bit transfer	
		0101	6 bit transfer	
		0110	7 bit transfer	
		0111	8 bit transfer	
		1000	9 bit transfer	
		1001	10 bit transfer	
		1010	11 bit transfer	
		1011	12 bit transfer	
		1100	13 bit transfer	
		1101	14 bit transfer	
		1110	15 bit transfer	
		1111	16 bit transfer	
5:4	FRF		Frame Format.	00
		00	SPI	
		01	SSI	
		10	Microwire	
		11	This combination is not supported and should not be used.	

Table 144: SSP Control Register 0 (SSPCR0 - address 0xE005 C000) bit description

Bit	Symbol	Value	Description	Reset value
6	CPOL		Clock Out Polarity. This bit is only used in SPI mode.	0
		0	SSP controller maintains the bus clock low between frames.	
		1	SSP controller maintains the bus clock high between frames.	
7	CPHA		Clock Out Phase. This bit is only used in SPI mode.	0
		0	SSP controller captures serial data on the first clock transition of the frame, that is, the transition **away from** the inter-frame state of the clock line.	
		1	SSP controller captures serial data on the second clock transition of the frame, that is, the transition **back to** the inter-frame state of the clock line.	
15:8	SCR		Serial Clock Rate. The number of prescaler-output clocks per bit on the bus, minus one. Given that CPSDVR is the prescale divider, and the VPB clock PCLK clocks the prescaler, the bit frequency is PCLK / (CPSDVSR * [SCR+1]).	0x00

8.2 SSP Control Register 1 (SSPCR1 - 0xE005 C004)

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

Table 145: SSP Control Register 1 (SSPCR1 - address 0xE005 C004) bit description

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

8.3 SSP Data Register (SSPDR - 0xE005 C008)

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

Table 146: SSP Data Register (SSPDR - address 0xE005 C008) bit description

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

8.4 SSP Status Register (SSPSR - 0xE005 C00C)

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

Table 147: SSP Status Register (SSPSR - address 0xE005 C00C) bit description

Bit	Symbol	Description	Reset value
0	TFE	Transmit FIFO Empty. This bit is 1 is the Transmit FIFO is empty, 0 if not.	1
1	TNF	Transmit FIFO Not Full. This bit is 0 if the Tx FIFO is full, 1 if not.	1
2	RNE	Receive FIFO Not Empty. This bit is 0 if the Receive FIFO is empty, 1 if not.	0
3	RFF	Receive FIFO Full. This bit is 1 if the Receive FIFO is full, 0 if not.	0
4	BSY	Busy. This bit is 0 if the SSP controller is idle, or 1 if it is currently sending/receiving a frame and/or the Tx FIFO is not empty.	0
7:5	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8.5 SSP Clock Prescale Register (SSPCPSR - 0xE005 C010)

This register controls the factor by which the Prescaler divides the APB clock PCLK to yield the prescaler clock that is, in turn, divided by the SCR factor in SSPCR0, to determine the bit clock.

Table 148: SSP Clock Prescale Register (SSPCPSR - address 0xE005 C010) bit description

Bit	Symbol	Description	Reset value
7:0	CPSDVSR	This even value between 2 and 254, by which PCLK is divided to yield the prescaler output clock. Bit 0 always reads as 0.	0

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

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

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

8.6 SSP Interrupt Mask Set/Clear Register (SSPIMSC - 0xE005 C014)

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

Table 149: SSP Interrupt Mask Set/Clear Register (SSPIMSC - address 0xE005 CF014) bit description

Bit	Symbol	Description	Reset value
0	RORIM	Software should set this bit to enable interrupt when a Receive Overrun occurs, that is, when the Rx FIFO is full and another frame is completely received. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs.	0
1	RTIM	Software should set this bit to enable interrupt when a Receive Timeout condition occurs. A Receive Timeout occurs when the Rx FIFO is not empty, and no new data has been received, nor has data been read from the FIFO, for 32 bit times.	0
2	RXIM	Software should set this bit to enable interrupt when the Rx FIFO is at least half full.	0
3	TXIM	Software should set this bit to enable interrupt when the Tx FIFO is at least half empty.	0
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8.7 SSP Raw Interrupt Status Register (SSPRIS - 0xE005 C018)

This read-only register contains a 1 for each interrupt condition that is asserted, regardless of whether or not the interrupt is enabled in the SSPIMSC.

Table 150: SSP Raw Interrupt Status Register (SSPRIS - address 0xE005 C018) bit description

Bit	Symbol	Description	Reset value
0	RORRIS	This bit is 1 if another frame was completely received while the RxFIFO was full. The ARM spec implies that the preceding frame data is overwritten by the new frame data when this occurs.	0
1	RTRIS	This bit is 1 if when there is a Receive Timeout condition. **Note:** A Receive Timeout can be negated if further data is received.	0
2	RXRIS	This bit is 1 if the Rx FIFO is at least half full.	0
3	TXRIS	This bit is 1 if the Tx FIFO is at least half empty.	1
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8.8 SSP Masked Interrupt Register (SSPMIS - 0xE005 C01C)

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

Table 151: SSP Masked Interrupt Status Register (SSPMIS -address 0xE005 C01C) bit description

Bit	Symbol	Description	Reset value
0	RORMIS	This bit is 1 if another frame was completely received while the RxFIFO was full, and this interrupt is enabled.	0
1	RTMIS	This bit is 1 when there is a Receive Timeout condition and this interrupt is enabled. **Note:** A Receive Timeout can be negated if further data is received.	0
2	RXMIS	This bit is 1 if the Rx FIFO is at least half full, and this interrupt is enabled.	0
3	TXMIS	This bit is 1 if the Tx FIFO is at least half empty, and this interrupt is enabled.	0
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

8.9 SSP Interrupt Clear Register (SSPICR - 0xE005 C020)

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

Table 152: SSP interrupt Clear Register (SSPICR - address 0xE005 C020) bit description

Bit	Symbol	Description	Reset value
0	RORIC	Writing a 1 to this bit clears the “frame was received when RxFIFO was full” interrupt.	Undefined
1	RTIC	Writing a 1 to this bit clears the Receive Timeout interrupt.	Undefined
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

1. How to read this chapter

The following features and registers are available in **LPC2104/01, LPC2105/01, and LPC2106/01 only:**

- External event counting
- T0CTCR and T1CTCR registers (Table 14–157)

2. Basic configuration

The Timer0/1 peripherals are configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bits PCTIM0/1.

   Remark: On reset, Timer0/1 are enabled (PCTIM0/1 = 1).
2. Pins: Select Timer0/1 pins and pin modes in registers PINSELn and PINMODEn (see Section 7–2).
3. Interrupts: See register T0/1MCR (Table 14–158) and T0/1CCR (Table 14–159) for match and capture events. Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

3. Features

- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.
- Counter or Timer operation
- External Event Counting capabilities.
- Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to four external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.

4. Applications

- Interval Timer for counting internal events.
- Pulse Width Demodulator via Capture inputs.
- Free running timer.
- External Event/Clock counter.

5. Description

The Timer/Counter is designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock, and can optionally generate interrupts or perform other actions at specified timer values, based on four match registers. It also includes four capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

6. Pin description

Table 14–153 gives a brief summary of each of the Timer/Counter related pins.

Table 153. Timer/Counter pin description

Pin	Type	Description
CAP0.2..0 CAP1.3..0	Input	Capture Signals- A transition on a capture pin can be configured to load one of the Capture Registers with the value in the Timer Counter and optionally generate an interrupt. Capture functionality can be selected from a number of pins. Here is the list of all CAPTURE signals, together with pins on where they can be selected: • CAP0.0: P0.2 • CAP0.1: P0.4 • CAP0.2: P0.6 • CAP1.0: P0.10 • CAP1.1: P0.11 • CAP1.2: P0.17 • CAP1.3: P0.18 Timer/Counter block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 14–7.3.
MAT0.2..0 MAT1.3..0	Output	External Match Output 0/1- When a match register 0/1 (MR3:0) equals the timer counter (TC) this output can either toggle, go low, go high, or do nothing. The External Match Register (EMR) controls the functionality of this output. Match Output functionality can be selected on a number of pins in parallel. Here is the list of all MATCH signals, together with pins on where they can be selected: • MAT0.0: P0.3 • MAT0.1: P0.5 • MAT0.2: P0.16 • MAT1.0: P0.12 • MAT1.1: P0.13 • MAT1.2: P0.19 • MAT1.3: P0.20

7. Register description

Each Timer/Counter contains the registers shown in Table 14–154. More detailed descriptions follow.

Table 154. TIMER/COUNTER0 and TIMER/COUNTER1 register map

Generic Name	Description	Access	Reset value[1]	TIMER/ COUNTER0 Address & Name	TIMER/ COUNTER1 Address & Name
IR	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0	0xE000 4000 T0IR	0xE000 8000 T1IR
TCR	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	R/W	0	0xE000 4004 T0TCR	0xE000 8004 T1TCR
TC	Timer Counter. The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	R/W	0	0xE000 4008 T0TC	0xE000 8008 T1TC
PR	Prescale Register. The Prescale Counter (below) is equal to this value, the next clock increments the TC and clears the PC.	R/W	0	0xE000 400C T0PR	0xE000 800C T1PR
PC	Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	R/W	0	0xE000 4010 T0PC	0xE000 8010 T1PC
MCR	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	R/W	0	0xE0004014 T0MCR	0xE000 8014 T1MCR
MR0	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	R/W	0	0xE000 4018 T0MR0	0xE000 8018 T1MR0
MR1	Match Register 1. See MR0 description.	R/W	0	0xE000 401C T0MR1	0xE000 801C T1MR1
MR2	Match Register 2. See MR0 description.	R/W	0	0xE000 4020 T0MR2	0xE000 8020 T1MR2
MR3	Match Register 3. See MR0 description.	R/W	0	0xE000 4024 T0MR3	0xE000 8024 T1MR3
CCR	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.	R/W	0	0xE000 4028 T0CCR	0xE000 8028 T1CCR
CR0	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CAPn.0(CAP0.0 or CAP1.0 respectively) input.	RO	0	0xE000 402C T0CR0	0xE000 802C T1CR0
CR1	Capture Register 1. See CR0 description.	RO	0	0xE000 4030 T0CR1	0xE000 8030 T1CR1
CR2	Capture Register 2. See CR0 description.	RO	0	0xE000 4034 T0CR2	0xE000 8034 T1CR2

Table 154. TIMER/COUNTER0 and TIMER/COUNTER1 register map

Generic Name	Description	Access	Reset value[1]	TIMER/ COUNTER0 Address & Name	TIMER/ COUNTER1 Address & Name
CR3	Capture Register 3. See CR0 description.	RO	0	0xE000 4038 T0CR3	0xE000 8038 T1CR3
EMR	External Match Register. The EMR controls the external match pins MATn.0-3 (MAT0.0-3 and MAT1.0-3 respectively).	R/W	0	0xE000 403C T0EMR	0xE000 803C T1EMR
CTCR	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	R/W	0	0xE000 4070 T0CTCR	0xE000 8070 T1CTCR

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

7.1 Interrupt Register (IR, TIMER0: T0IR - 0xE000 4000 and TIMER1: T1IR - 0xE000 8000)

The Interrupt Register consists of four bits for the match interrupts and four bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 155: Interrupt Register (IR, TIMER0: T0IR - address 0xE000 4000 and TIMER1: T1IR - address 0xE000 8000) bit description

Bit	Symbol	Description	Reset value
0	MR0 Interrupt	Interrupt flag for match channel 0.	0
1	MR1 Interrupt	Interrupt flag for match channel 1.	0
2	MR2 Interrupt	Interrupt flag for match channel 2.	0
3	MR3 Interrupt	Interrupt flag for match channel 3.	0
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.	0
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.	0
6	CR2 Interrupt	Interrupt flag for capture channel 2 event.	0
7[1]	CR3 Interrupt	Interrupt flag for capture channel 3 event.	0

[1] Available for Timer1 only. This bit is reserved for Timer0.

7.2 Timer Control Register (TCR, TIMER0: T0TCR - 0xE000 4004 and TIMER1: T1TCR - 0xE000 8004)

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

Table 156: Timer Control Register (TCR, TIMER0: T0TCR - address 0xE000 4004 and TIMER1: T1TCR - address 0xE000 8004) bit description

Bit	Symbol	Description	Reset value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until TCR[1] is returned to zero.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7.3 Count Control Register (CTCR, TIMER0: T0CTCR - 0xE000 4070 and TIMER1: T1TCR - 0xE000 8070)

The Count Control Register (CTCR) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting (see Section 14–1).

When Counter Mode is chosen as a mode of operation, the CAP input (selected by the CTCR bits 3:2) is sampled on every rising edge of the PCLK clock. After comparing two consecutive samples of this CAP input, one of the following four events is recognized: rising edge, falling edge, either of edges or no changes in the level of the selected CAP input. Only if the identified event corresponds to the one selected by bits 1:0 in the CTCR register, the Timer Counter register will be incremented.

Effective processing of the externally supplied clock to the counter has some limitations. Since two successive rising edges of the PCLK clock are used to identify only one edge on the CAP selected input, the frequency of the CAP input can not exceed one half of the PCLK clock. Consequently, duration of the high/low levels on the same CAP input in this case can not be shorter than 1/PCLK.

Table 157: Count Control Register (CTCR, TIMER0: T0CTCR - address 0xE000 4070 and TIMER1: T1TCR - address 0xE000 8070) bit description

Bit	Symbol	Value	Description	Reset value
1:0	Counter/ Timer Mode		This field selects which rising PCLK edges can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC).	00
		00	Timer Mode: every rising PCLK edge	
		01	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.	
		10	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.	
		11	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.	

Table 157: Count Control Register (CTCR, TIMER0: T0CTCR - address 0xE000 4070 and TIMER1: T1TCR - address 0xE000 8070) bit description

Bit	Symbol	Value	Description	Reset value
3:2	Count Input Select		When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking:	00
		00	CAPn.0 (CAP0.0 for TIMER0 and CAP1.0 for TIMER1)	
		01	CAPn.1 (CAP0.1 for TIMER0 and CAP1.1 for TIMER1)	
		10	CAPn.2 (CAP0.2 for TIMER0 and CAP1.2 for TIMER1)	
		11	CAPn.3 (CAP1.3 for TIMER1)	
			Note: If Counter mode is selected for a particular CAPn input in the TnCTCR, the 3 bits for that input in the Capture Control Register (TnCCR) must be programmed as 000. However, capture and/or interrupt can be selected for the other 3 CAPn inputs in the same timer.	
7:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7.4 Timer Counter (TC, TIMER0: T0TC - 0xE000 4008 and TIMER1: T1TC - 0xE000 8008)

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

7.5 Prescale Register (PR, TIMER0: T0PR - 0xE000 400C and TIMER1: T1PR - 0xE000 800C)

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

7.6 Prescale Counter Register (PC, TIMER0: T0PC - 0xE000 4010 and TIMER1: T1PC - 0xE000 8010)

The 32-bit Prescale Counter controls division of PCLK by some constant value before it is applied to the Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale Register, the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, etc.

7.7 Match Registers (MR0 - MR3)

The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

7.8 Match Control Register (MCR, TIMER0: T0MCR - 0xE000 4014 and TIMER1: T1MCR - 0xE000 8014)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter. The function of each of the bits is shown in Table 14–158.

Table 158: Match Control Register (MCR, TIMER0: T0MCR - address 0xE000 4014 and TIMER1: T1MCR - address 0xE000 8014) bit description

Bit	Symbol	Value	Description	Reset value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		0	This interrupt is disabled	
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.	0
		0	Feature disabled.	
5	MR1S	1	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.	0
		0	Feature disabled.	
6	MR2I	1	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.	0
		0	This interrupt is disabled	
7	MR2R	1	Reset on MR2: the TC will be reset if MR2 matches it.	0
		0	Feature disabled.	
8	MR2S	1	Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.	0
		0	Feature disabled.	
9	MR3I	1	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0
		0	This interrupt is disabled	
10	MR3R	1	Reset on MR3: the TC will be reset if MR3 matches it.	0
		0	Feature disabled.	
11	MR3S	1	Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.	0
		0	Feature disabled.	
15:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7.9 Capture Registers (CR0 - CR3)

Each Capture register is associated with a device pin and may be loaded with the Timer Counter value when a specified event occurs on that pin. The settings in the Capture Control Register register determine whether the capture function is enabled, and whether a capture event happens on the rising edge of the associated pin, the falling edge, or on both edges.

7.10 Capture Control Register (CCR, TIMER0: T0CCR - 0xE000 4028 and TIMER1: T1CCR - 0xE000 8028)

The Capture Control Register is used to control whether one of the four Capture Registers is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event. Setting both the rising and falling bits at the same time is a valid configuration, resulting in a capture event for both edges. In the description below, "n" represents the Timer number, 0 or 1.

Table 159: Capture Control Register (CCR, TIMER0: T0CCR - address 0xE000 4028 and TIMER1: T1CCR - address 0xE000 8028) bit description

Bit	Symbol	Value	Description	Reset value
0	CAP0RE	1	Capture on CAPn.0 rising edge: a sequence of 0 then 1 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
1	CAP0FE	1	Capture on CAPn.0 falling edge: a sequence of 1 then 0 on CAPn.0 will cause CR0 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
2	CAP0I	1	Interrupt on CAPn.0 event: a CR0 load due to a CAPn.0 event will generate an interrupt.	0
		0	This feature is disabled.	
3	CAP1RE	1	Capture on CAPn.1 rising edge: a sequence of 0 then 1 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
4	CAP1FE	1	Capture on CAPn.1 falling edge: a sequence of 1 then 0 on CAPn.1 will cause CR1 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
5	CAP1I	1	Interrupt on CAPn.1 event: a CR1 load due to a CAPn.1 event will generate an interrupt.	0
		0	This feature is disabled.	
6	CAP2RE	1	Capture on CAPn.2 rising edge: A sequence of 0 then 1 on CAPn.2 will cause CR2 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
7	CAP2FE	1	Capture on CAPn.2 falling edge: a sequence of 1 then 0 on CAPn.2 will cause CR2 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
8	CAP2I	1	Interrupt on CAPn.2 event: a CR2 load due to a CAPn.2 event will generate an interrupt.	0
		0	This feature is disabled.	

Table 159: Capture Control Register (CCR, TIMER0: T0CCR - address 0xE000 4028 and TIMER1: T1CCR - address 0xE000 8028) bit description

Bit	Symbol	Value	Description	Reset value
9[1]	CAP3RE	1	Capture on CAPn.3 rising edge: a sequence of 0 then 1 on CAPn.3 will cause CR3 to be loaded with the contents of TC.	0
		0	This feature is disabled.	
10[1]	CAP3FE	1	Capture on CAPn.3 falling edge: a sequence of 1 then 0 on CAPn.3 will cause CR3 to be loaded with the contents of TC	0
		0	This feature is disabled.	
11[1]	CAP3I	1	Interrupt on CAPn.3 event: a CR3 load due to a CAPn.3 event will generate an interrupt.	0
		0	This feature is disabled.	
15:12	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] Available for Timer1 only. This bit is reserved for Timer0.

7.11 External Match Register (EMR, TIMER0: T0EMR - 0xE000 403C; and TIMER1: T1EMR - 0xE000 803C)

The External Match Register provides both control and status of the external match pins MAT(0-3). Bits EM3:0 can be written only when Timer is disabled (bit 1 in Timer Control Register is 0). Only under this condition an initial output level on MAT pin(s) can be set. Once the Timer is enabled, EM3:0 can be changed only by Timer's activities specified by the EMC bits.

Table 160: External Match Register (EMR, TIMER0: T0EMR - address 0xE000 403C and TIMER1: T1EMR - address0xE000 803C) bit description

Bit	Symbol	Description	Reset value
0	EM0	External Match 0. This bit reflects the state of output MAT0.0/MAT1.0, whether or not this output is connected to its pin. When a match occurs between the TC and MR0, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[5:4] control the functionality of this output.	0
1	EM1	External Match 1. This bit reflects the state of output MAT0.1/MAT1.1, whether or not this output is connected to its pin. When a match occurs between the TC and MR1, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[7:6] control the functionality of this output.	0
2	EM2	External Match 2. This bit reflects the state of output MAT0.2/MAT1.2, whether or not this output is connected to its pin. When a match occurs between the TC and MR2, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[9:8] control the functionality of this output.	0
3[1]	EM3	External Match 3. This bit reflects the state of output MAT1.3, whether or not this output is connected to its pin. When a match occurs between the TC and MR3, this output of the timer can either toggle, go low, go high, or do nothing. Bits EMR[11:10] control the functionality of this output.	0
5:4	EMC0	External Match Control 0. Determines the functionality of External Match 0. Table 14–161 shows the encoding of these bits.	00
7:6	EMC1	External Match Control 1. Determines the functionality of External Match 1. Table 14–161 shows the encoding of these bits.	00

Table 160: External Match Register (EMR, TIMER0: T0EMR - address 0xE000 403C and TIMER1: T1EMR - address0xE000 803C) bit description

Bit	Symbol	Description	Reset value
9:8	EMC2	External Match Control 2. Determines the functionality of External Match 2. Table 14–161 shows the encoding of these bits.	00
11:10	EMC3[1]	External Match Control 3. Determines the functionality of External Match 3. Table 14–161 shows the encoding of these bits.	00
15:12	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] Available for Timer1 only. This bit is reserved for Timer0.

Table 161. External match control

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

8. Example timer operation

Figure 14–51 shows a timer configured to reset the count and generate an interrupt on match. The prescaler is set to 2 and the match register set to 6. At the end of the timer cycle where the match occurs, the timer count is reset. This gives a full length cycle to the match value. The interrupt indicating that a match occurred is generated in the next clock after the timer reached the match value.

Figure 14–52 shows a timer configured to stop and generate an interrupt on match. The prescaler is again set to 2 and the match register set to 6. In the next clock after the timer reaches the match value, the timer enable bit in TCR is cleared, and the interrupt indicating that a match occurred is generated.

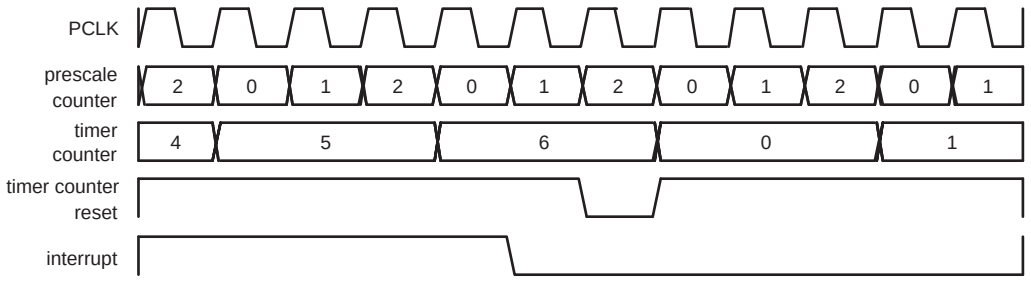


Fig 51. A timer cycle in which PR=2, MRx=6, and both interrupt and reset on match are enabled

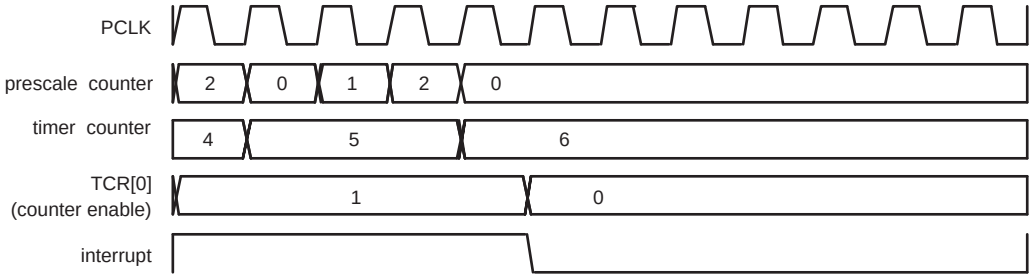


Fig 52. A timer cycle in which PR=2, MRx=6, and both interrupt and stop on match are enabled

9. Architecture

The block diagram for TIMER/COUNTER0 and TIMER/COUNTER1 is shown in Figure 14–53.

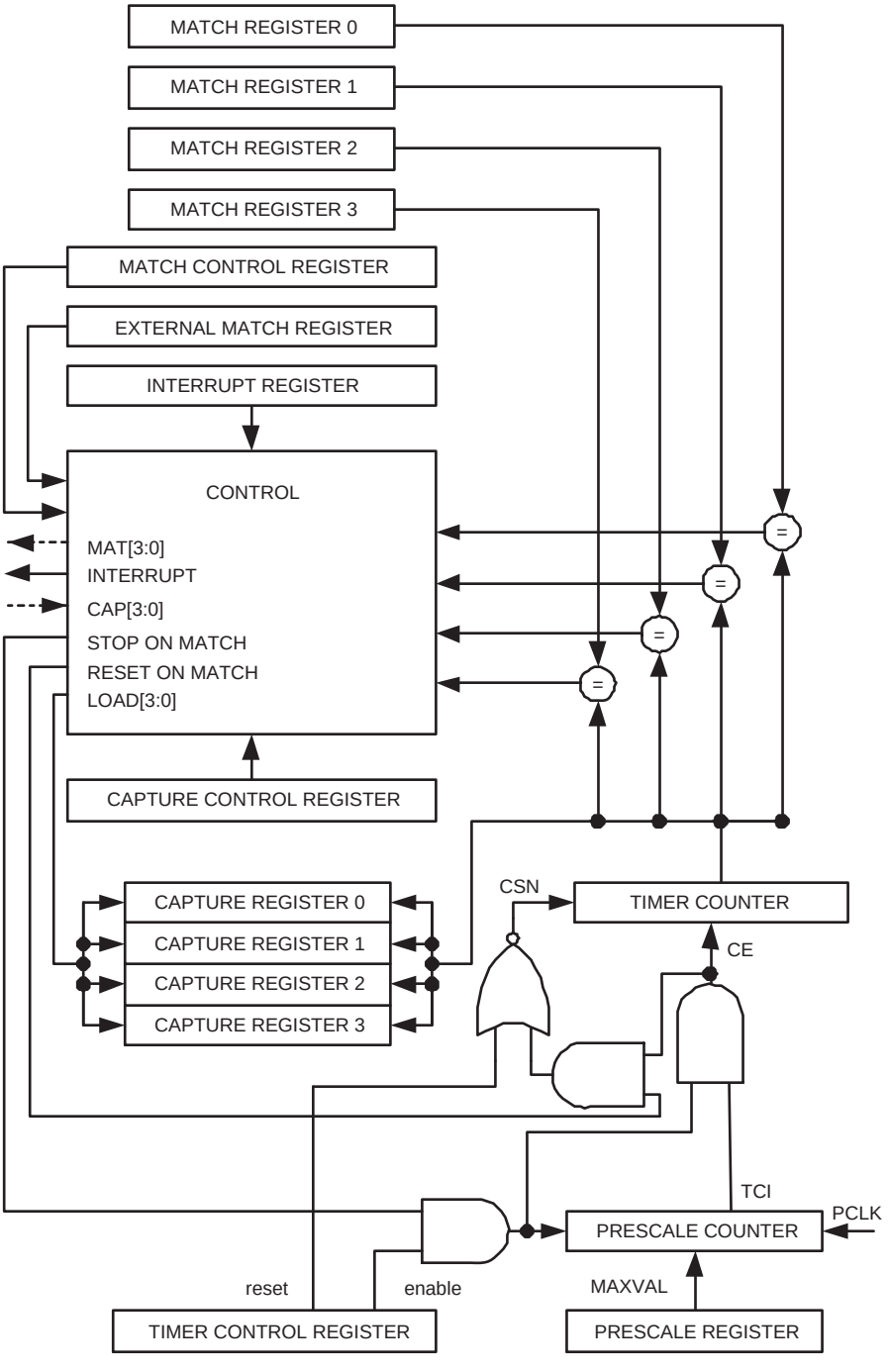


Fig 53. Timer block diagram

1. Basic configuration

The PWM is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCPWM0.

   Remark: On reset, the PWM is enabled (PCPWM0 = 1).
2. Pins: Select PWM pins in registers PINSEL0/1 (see Section 7–2).
3. Interrupts: See register PWMMCR (Table 15–167) for match events. Interrupts are enabled in the VIC using the VICIntEnable register (Table 5–43).

2. Features

- Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types. The match registers also allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- An external output for each match register with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Toggle on match.
  - Do nothing on match.
- Supports single edge controlled and/or double edge controlled PWM outputs. Single edge controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low. Double edge controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.
- Pulse period and width can be any number of timer counts. This allows complete flexibility in the trade-off between resolution and repetition rate. All PWM outputs will occur at the same repetition rate.
- Double edge controlled PWM outputs can be programmed to be either positive going or negative going pulses.
- Match register updates are synchronized with pulse outputs to prevent generation of erroneous pulses. Software must "release" new match values before they can become effective.
- May be used as a standard timer if the PWM mode is not enabled.
- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.

3. Description

The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC2104/05/06. The Timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers. It also includes four capture inputs to save the timer value when an input signal transitions, and optionally generate an interrupt when those events occur. The PWM function is in addition to these features, and is based on match register events.

The ability to separately control rising and falling edge locations allows the PWM to be used for more applications. For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and positions.

Two match registers can be used to provide a single edge controlled PWM output. One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match. The other match register controls the PWM edge position. Additional single edge controlled PWM outputs require only one match register each, since the repetition rate is the same for all PWM outputs. Multiple single edge controlled PWM outputs will all have a rising edge at the beginning of each PWM cycle, when an PWMMR0 match occurs.

Three match registers can be used to provide a PWM output with both edges controlled. Again, the PWMMR0 match register controls the PWM cycle rate. The other match registers control the two PWM edge positions. Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs.

With double edge controlled PWM outputs, specific match registers control the rising and falling edge of the output. This allows both positive going PWM pulses (when the rising edge occurs prior to the falling edge), and negative going PWM pulses (when the falling edge occurs prior to the rising edge).

Figure 15–54 shows the block diagram of the PWM. The portions that have been added to the standard timer block are on the right hand side and at the top of the diagram.

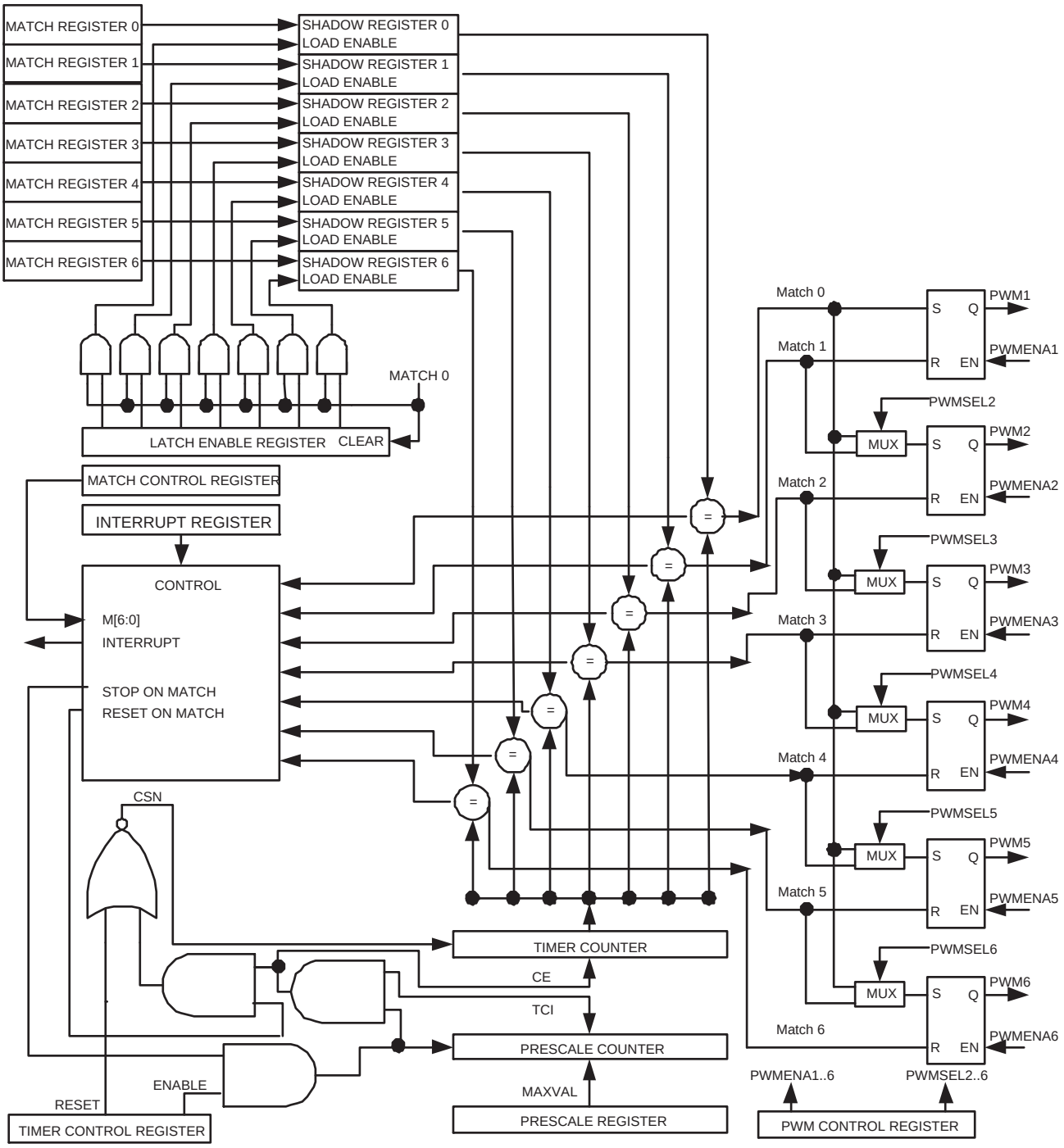
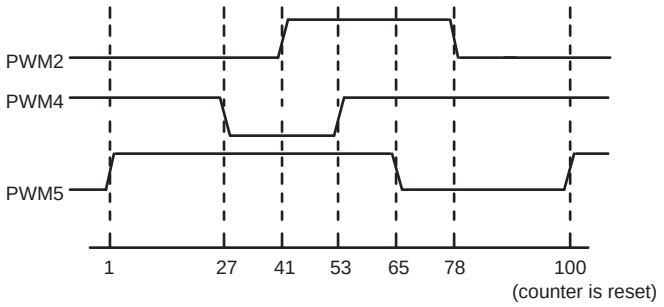


Fig 54. PWM block diagram.

A sample of how PWM values relate to waveform outputs is shown in Figure 15–55. PWM output logic is shown in Figure 15–54 that allows selection of either single or double edge controlled PWM outputs via the multiplexers controlled by the PWMSELn bits. The match register selections for various PWM outputs is shown in Table 15–162. This

implementation supports up to N-1 single edge PWM outputs or (N-1)/2 double edge PWM outputs, where N is the number of match registers that are implemented. PWM types can be mixed if desired.



The waveforms below show a single PWM cycle and demonstrate PWM outputs under the following conditions:

The timer is configured for PWM mode (counter resets to one).

Match 0 is configured to reset the timer/counter when a match event occurs.

All PWM related Match registers are configured for toggle on match.

Control bits PWMSEL2 and PWMSEL4 are set.

The Match register values are as follows:

MR0 = 100 (PWM rate)

MR1 = 41, MR2 = 78 (PWM2 output)

MR3 = 53, MR$ = 27 (PWM4 output)

MR5 = 65 (PWM5 output)

Fig 55. Sample PWM waveforms

Table 162. Set and reset inputs for PWM Flip-Flops

PWM Channel	Single edge PWM (PWMSELn = 0)		Double edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0[1]	Match 1[1]
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2[2]	Match 3[2]
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4[2]	Match 5[2]
6	Match 0	Match 6	Match 5	Match 6

[1] Identical to single edge mode in this case since Match 0 is the neighboring match register. Essentially, PWM1 cannot be a double edged output.

[2] It is generally not advantageous to use PWM channels 3 and 5 for double edge PWM outputs because it would reduce the number of double edge PWM outputs that are possible. Using PWM 2, PWM4, and PWM6 for double edge PWM outputs provides the most pairings.

3.1 Rules for Single Edge Controlled PWM Outputs

1. All single edge controlled PWM outputs go high at the beginning of a PWM cycle unless their match value is equal to 0.

2. Each PWM output will go low when its match value is reached. If no match occurs (i.e. the match value is greater than the PWM rate), the PWM output remains continuously high.

3.2 Rules for Double Edge Controlled PWM Outputs

Five rules are used to determine the next value of a PWM output when a new cycle is about to begin:

1. The match values for the **next** PWM cycle are used at the end of a PWM cycle (a time point which is coincident with the beginning of the next PWM cycle), except as noted in rule 3.
2. A match value equal to 0 or the current PWM rate (the same as the Match channel 0 value) have the same effect, except as noted in rule 3. For example, a request for a falling edge at the beginning of the PWM cycle has the same effect as a request for a falling edge at the end of a PWM cycle.
3. When match values are changing, if one of the "old" match values is equal to the PWM rate, it is used again once if the neither of the new match values are equal to 0 or the PWM rate, and there was no old match value equal to 0.
4. If both a set and a clear of a PWM output are requested at the same time, clear takes precedence. This can occur when the set and clear match values are the same as in, or when the set or clear value equals 0 and the other value equals the PWM rate.
5. If a match value is out of range (i.e. greater than the PWM rate value), no match event occurs and that match channel has no effect on the output. This means that the PWM output will remain always in one state, allowing always low, always high, or "no change" outputs.

4. Pin description

Table 15–163 gives a brief summary of each of PWM related pins.

Table 163. Pin summary

Pin	Type	Description
PWM1	Output	Output from PWM channel 1.
PWM2	Output	Output from PWM channel 2.
PWM3	Output	Output from PWM channel 3.
PWM4	Output	Output from PWM channel 4.
PWM5	Output	Output from PWM channel 5.
PWM6	Output	Output from PWM channel 6.

5. Register description

The PWM function adds new registers and registers bits as shown in Table 15–164 below.

Table 164. Pulse Width Modulator Register Map

Name	Description	Access	Reset value[1]	Address
PWMIR	PWM Interrupt Register. The PWMIR can be written to clear interrupts. The PWMIR can be read to identify which of the possible interrupt sources are pending.	R/W	0	0xE001 4000
PWMTCR	PWM Timer Control Register. The PWMTCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the PWMTCR.	R/W	0	0xE001 4004
PWMTC	PWM Timer Counter. The 32-bit TC is incremented every PWMPR+1 cycles of PCLK. The PWMTC is controlled through the PWMTCR.	R/W	0	0xE001 4008
PWMPR	PWM Prescale Register. The PWMTC is incremented every PWMPR+1 cycles of PCLK.	R/W	0	0xE001 400C
PWMPC	PWM Prescale Counter. The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PWMPR is reached, the PWMTC is incremented. The PWMTC is observable and controllable through the bus interface.	R/W	0	0xE001 4010
PWMMCR	PWM Match Control Register. The PWMMCR is used to control if an interrupt is generated and if the PWMTC is reset when a Match occurs.	R/W	0	0xE001 4014
PWMMR0	PWM Match Register 0. PWMMR0 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR0 and the PWMTC sets all PWM outputs that are in single-edge mode, and sets PWM1 if it is in double-edge mode.	R/W	0	0xE001 4018
PWMMR1	PWM Match Register 1. PWMMR1 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR1 and the PWMTC clears PWM1 in either single-edge mode or double-edge mode, and sets PWM2 if it is in double-edge mode.	R/W	0	0xE001 401C
PWMMR2	PWM Match Register 2. PWMMR2 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR2 and the PWMTC clears PWM2 in either single-edge mode or double-edge mode, and sets PWM3 if it is in double-edge mode.	R/W	0	0xE001 4020

Table 164. Pulse Width Modulator Register Map

Name	Description	Access	Reset value[1]	Address
PWMMR3	PWM Match Register 3. PWMMR3 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR3 and the PWMTC clears PWM3 in either single-edge mode or double-edge mode, and sets PWM4 if it is in double-edge mode.	R/W	0	0xE001 4024
PWMMR4	PWM Match Register 4. PWMMR4 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR4 and the PWMTC clears PWM4 in either single-edge mode or double-edge mode, and sets PWM5 if it is in double-edge mode.	R/W	0	0xE001 4040
PWMMR5	PWM Match Register 5. PWMMR5 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR5 and the PWMTC clears PWM5 in either single-edge mode or double-edge mode, and sets PWM6 if it is in double-edge mode.	R/W	0	0xE001 4044
PWMMR6	PWM Match Register 6. PWMMR6 can be enabled through PWMMCR to reset the PWMTC, stop both the PWMTC and PWMPC, and/or generate an interrupt when it matches the PWMTC. In addition, a match between PWMMR6 and the PWMTC clears PWM6 in either single-edge mode or double-edge mode.	R/W	0	0xE001 4048
PWMPCR	PWM Control Register. Enables PWM outputs and selects PWM channel types as either single-edge or double-edge controlled.	R/W	0	0xE001 404C
PWMLER	PWM Latch Enable Register. Enables use of new PWM match values.	R/W	0	0xE001 4050

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

5.1 PWM Interrupt Register (PWMIR - 0xE001 4000)

The PWM Interrupt Register consists bits described in (Table 15–165). If an interrupt is generated then the corresponding bit in the PWMIR will be high. Otherwise, the bit will be low. Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Table 165: PWM Interrupt Register (PWMIR - address 0xE001 4000) bit description

Bit	Symbol	Description	Reset value
0	PWMMR0 Interrupt	Interrupt flag for PWM match channel 0.	0
1	PWMMR1 Interrupt	Interrupt flag for PWM match channel 1.	0
2	PWMMR2 Interrupt	Interrupt flag for PWM match channel 2.	0

Table 165: PWM Interrupt Register (PWMIR - address 0xE001 4000) bit description

Bit	Symbol	Description	Reset value
3	PWMMR3 Interrupt	Interrupt flag for PWM match channel 3.	0
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0000
8	PWMMR4 Interrupt	Interrupt flag for PWM match channel 4.	0
9	PWMMR5 Interrupt	Interrupt flag for PWM match channel 5.	0
10	PWMMR6 Interrupt	Interrupt flag for PWM match channel 6.	0
15:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.2 PWM Timer Control Register (PWMTCR - 0xE001 4004)

The PWM Timer Control Register (PWMTCR) is used to control the operation of the PWM Timer Counter. The function of each of the bits is shown in Table 15–166.

Table 166: PWM Timer Control Register (PWMTCR - address 0xE001 4004) bit description

Bit	Symbol	Value	Description	Reset Value
0	Counter Enable	1	The PWM Timer Counter and PWM Prescale Counter are enabled for counting.	0
		0	The counters are disabled.	
1	Counter Reset	1	The PWM Timer Counter and the PWM Prescale Counter are synchronously reset on the next positive edge of PCLK. The counters remain reset until this bit is returned to zero.	0
		0	Clear reset.	
2	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
3	PWM Enable	1	PWM mode is enabled (counter resets to 1). PWM mode causes the shadow registers to operate in connection with the Match registers. A program write to a Match register will not have an effect on the Match result until the corresponding bit in PWMLER has been set, followed by the occurrence of a PWM Match 0 event. Note that the PWM Match register that determines the PWM rate (PWM Match Register 0 - MR0) must be set up prior to the PWM being enabled. Otherwise a Match event will not occur to cause shadow register contents to become effective.	0
		0	Timer mode is enabled (counter resets to 0).	
7:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.3 PWM Timer Counter (PWMTC - 0xE001 4008)

The 32-bit PWM Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the PWMTC will count up through the value 0xFFFF FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

5.4 PWM Prescale Register (PWMPR - 0xE001 400C)

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

5.5 PWM Prescale Counter Register (PWMPC - 0xE001 4010)

The 32-bit PWM Prescale Counter controls division of PCLK by some constant value before it is applied to the PWM Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The PWM Prescale Counter is incremented on every PCLK. When it reaches the value stored in the PWM Prescale Register, the PWM Timer Counter is incremented and the PWM Prescale Counter is reset on the next PCLK. This causes the PWM TC to increment on every PCLK when PWMPR = 0, every 2 PCLKs when PWMPR = 1, etc.

5.6 PWM Match Registers (PWMMR0 - PWMMR6)

The 32-bit PWM Match register values are continuously compared to the PWM Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the PWM Timer Counter, or stop the timer. Actions are controlled by the settings in the PWMMCR register.

5.7 PWM Match Control Register (PWMMCR - 0xE001 4014)

The PWM Match Control Register is used to control what operations are performed when one of the PWM Match Registers matches the PWM Timer Counter. The function of each of the bits is shown in Table 15–167.

Table 167: Match Control Register (MCR, TIMER0: T0MCR - address 0xE000 4014 and TIMER1: T1MCR - address 0xE000 8014) bit description

Bit	Symbol	Value	Description	Reset Value
0	PWMMR0I	1	Interrupt on PWMMR0: an interrupt is generated when PWMMR0 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
1	PWMMR0R	1	Reset on PWMMR0: the PWMTC will be reset if PWMMR0 matches it.	0
		0	This feature is disabled.	
2	PWMMR0S	1	Stop on PWMMR0: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches the PWMTC.	0
		0	This feature is disabled	

Table 167: Match Control Register (MCR, TIMER0: T0MCR - address 0xE000 4014 and TIMER1: T1MCR - address 0xE000 8014) bit description

Bit	Symbol	Value	Description	Reset Value
3	PWMMR1I	1	Interrupt on PWMMR1: an interrupt is generated when PWMMR1 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
4	PWMMR1R	1	Reset on PWMMR1: the PWMTC will be reset if PWMMR1 matches it.	0
		0	This feature is disabled.	
5	PWMMR1S	1	Stop on PWMMR1: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR1 matches the PWMTC.	0
		0	This feature is disabled.	
6	PWMMR2I	1	Interrupt on PWMMR2: an interrupt is generated when PWMMR2 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
7	PWMMR2R	1	Reset on PWMMR2: the PWMTC will be reset if PWMMR2 matches it.	0
		0	This feature is disabled.	
8	PWMMR2S	1	Stop on PWMMR2: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR2 matches the PWMTC.	0
		0	This feature is disabled	
9	PWMMR3I	1	Interrupt on PWMMR3: an interrupt is generated when PWMMR3 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
10	PWMMR3R	1	Reset on PWMMR3: the PWMTC will be reset if PWMMR3 matches it.	0
		0	This feature is disabled	
11	PWMMR3S	1	Stop on PWMMR3: The PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR3 matches the PWMTC.	0
		0	This feature is disabled	
12	PWMMR4I	1	Interrupt on PWMMR4: An interrupt is generated when PWMMR4 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
13	PWMMR4R	1	Reset on PWMMR4: the PWMTC will be reset if PWMMR4 matches it.	0
		0	This feature is disabled.	
14	PWMMR4S	1	Stop on PWMMR4: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR4 matches the PWMTC.	0
		0	This feature is disabled	
15	PWMMR5I	1	Interrupt on PWMMR5: An interrupt is generated when PWMMR5 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	

Table 167: Match Control Register (MCR, TIMER0: T0MCR - address 0xE000 4014 and TIMER1: T1MCR - address 0xE000 8014) bit description

Bit	Symbol	Value	Description	Reset Value
16	PWMMR5R	1	Reset on PWMMR5: the PWMTC will be reset if PWMMR5 matches it.	0
		0	This feature is disabled.	
17	PWMMR5S	1	Stop on PWMMR5: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR5 matches the PWMTC.	0
		0	This feature is disabled	
18	PWMMR6I	1	Interrupt on PWMMR6: an interrupt is generated when PWMMR6 matches the value in the PWMTC.	0
		0	This interrupt is disabled.	
19	PWMMR6R	1	Reset on PWMMR6: the PWMTC will be reset if PWMMR6 matches it.	0
		0	This feature is disabled.	
20	PWMMR6S	1	Stop on PWMMR6: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR6 matches the PWMTC.	0
		0	This feature is disabled	
31:21	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.8 PWM Control Register (PWMPCR - 0xE001 404C)

The PWM Control Register is used to enable and select the type of each PWM channel. The function of each of the bits are shown in Table 15–168.

Table 168: PWM Control Register (PWMPCR - address 0xE001 404C) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	PWMSEL2	1	Selects double edge controlled mode for the PWM2 output.	0
		0	Selects single edge controlled mode for PWM2.	
3	PWMSEL3	1	Selects double edge controlled mode for the PWM3 output.	0
		0	Selects single edge controlled mode for PWM3.	
4	PWMSEL4	1	Selects double edge controlled mode for the PWM4 output.	0
		0	Selects single edge controlled mode for PWM4.	
5	PWMSEL5	1	Selects double edge controlled mode for the PWM5 output.	0
		0	Selects single edge controlled mode for PWM5.	
6	PWMSEL6	1	Selects double edge controlled mode for the PWM6 output.	0
		0	Selects single edge controlled mode for PWM6.	
8:7	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
9	PWMENA1	1	The PWM1 output enabled.	0
		0	The PWM1 output disabled.	

Table 168: PWM Control Register (PWMPCR - address 0xE001 404C) bit description

Bit	Symbol	Value	Description	Reset Value
10	PWMENA2	1	The PWM2 output enabled.	0
		0	The PWM2 output disabled.	
11	PWMENA3	1	The PWM3 output enabled.	0
		0	The PWM3 output disabled.	
12	PWMENA4	1	The PWM4 output enabled.	0
		0	The PWM4 output disabled.	
13	PWMENA5	1	The PWM5 output enabled.	0
		0	The PWM5 output disabled.	
14	PWMENA6	1	The PWM6 output enabled.	0
		0	The PWM6 output disabled.	
15	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.9 PWM Latch Enable Register (PWMLER - 0xE001 4050)

The PWM Latch Enable Register is used to control the update of the PWM Match registers when they are used for PWM generation. When software writes to the location of a PWM Match register while the Timer is in PWM mode, the value is held in a shadow register. When a PWM Match 0 event occurs (normally also resetting the timer in PWM mode), the contents of shadow registers will be transferred to the actual Match registers if the corresponding bit in the Latch Enable Register has been set. At that point, the new values will take effect and determine the course of the next PWM cycle. Once the transfer of new values has taken place, all bits of the LER are automatically cleared. Until the corresponding bit in the PWMLER is set and a PWM Match 0 event occurs, any value written to the PWM Match registers has no effect on PWM operation.

For example, if PWM2 is configured for double edge operation and is currently running, a typical sequence of events for changing the timing would be:

- Write a new value to the PWM Match1 register.
- Write a new value to the PWM Match2 register.
- Write to the PWMLER, setting bits 1 and 2 at the same time.
- The altered values will become effective at the next reset of the timer (when a PWM Match 0 event occurs).

The order of writing the two PWM Match registers is not important, since neither value will be used until after the write to PWMLER. This insures that both values go into effect at the same time, if that is required. A single value may be altered in the same way if needed.

The function of each of the bits in the PWMLER is shown in Table 15–169.

Table 169: PWM Latch Enable Register (PWMLER - address 0xE001 4050) bit description

Bit	Symbol	Description	Reset value
0	Enable PWM Match 0 Latch	Writing a one to this bit allows the last value written to the PWM Match 0 register to be become effective when the timer is next reset by a PWM Match event. Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
1	Enable PWM Match 1 Latch	Writing a one to this bit allows the last value written to the PWM Match 1 register to be become effective when the timer is next reset by a PWM Match event. Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
2	Enable PWM Match 2 Latch	Writing a one to this bit allows the last value written to the PWM Match 2 register to be become effective when the timer is next reset by a PWM Match event. See Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
3	Enable PWM Match 3 Latch	Writing a one to this bit allows the last value written to the PWM Match 3 register to be become effective when the timer is next reset by a PWM Match event. See Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
4	Enable PWM Match 4 Latch	Writing a one to this bit allows the last value written to the PWM Match 4 register to be become effective when the timer is next reset by a PWM Match event. See Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
5	Enable PWM Match 5 Latch	Writing a one to this bit allows the last value written to the PWM Match 5 register to be become effective when the timer is next reset by a PWM Match event. See Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
6	Enable PWM Match 6 Latch	Writing a one to this bit allows the last value written to the PWM Match 6 register to be become effective when the timer is next reset by a PWM Match event. See Section 15–5.7 “PWM Match Control Register (PWMMCR - 0xE001 4014)”.	0
7	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

1. Basic configuration

The RTC is configured using the following registers:

1. Power: In the PCONP register (Table 3–27), set bit PCRTC = 1.

   Remark: On reset, the RTC is enabled.

2. Interrupts: See Section 16–5.1 for RTC interrupt handling. Interrupts are enabled in the VIC using the VICIntEnable register (Section 5–5.4).

2. Features

- Measures the passage of time to maintain a calendar and clock.
- Ultra low power design to support battery powered systems.
- Provides Seconds, Minutes, Hours, Day of Month, Month, Year, Day of Week, and Day of Year.
- Programmable reference clock divider allows adjustment of the RTC to match various crystal frequencies.

3. Description

The Real Time Clock (RTC) is designed to provide a set of counters to measure time during system power on and off operation. The RTC has been designed to use little power in Power-down mode, making it suitable for battery powered systems where the CPU is not running continuously (sleep mode).

4. Architecture

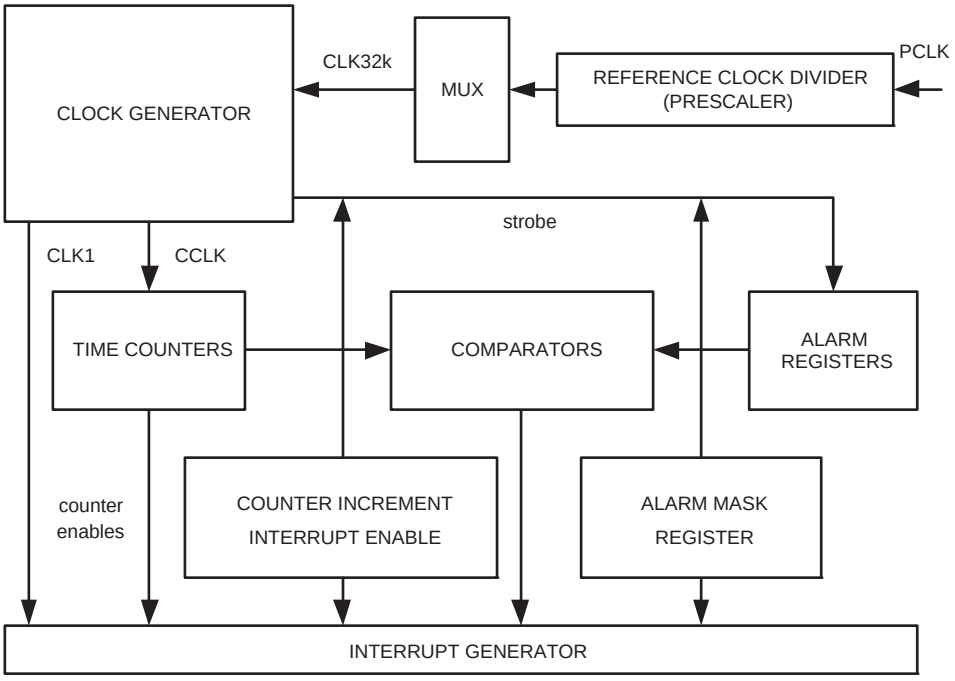


Fig 56. RTC block diagram

5. Register description

The RTC includes a number of registers. The address space is split into four sections by functionality. The first eight addresses are the Miscellaneous Register Group (Section 16–5.2). The second set of eight locations are the Time Counter Group (Section 16–5.12). The third set of eight locations contain the Alarm Register Group (Section 16–5.14). The remaining registers control the Reference Clock Divider.

The Real Time Clock includes the register shown in Table 16–170. Detailed descriptions of the registers follow.

Table 170. Real Time Clock (RTC) register map

Name	Size	Description	Access	Reset value[1]	Address
ILR	2	Interrupt Location Register	R/W	n/a	0xE002 4000
CTC	15	Clock Tick Counter	RO	n/a	0xE002 4004
CCR	4	Clock Control Register	R/W	n/a	0xE002 4008
CIIR	8	Counter Increment Interrupt Register	R/W	n/a	0xE002 400C
AMR	8	Alarm Mask Register	R/W	n/a	0xE002 4010
CTIME0	32	Consolidated Time Register 0	RO	n/a	0xE002 4014
CTIME1	32	Consolidated Time Register 1	RO	n/a	0xE002 4018
CTIME2	32	Consolidated Time Register 2	RO	n/a	0xE002 401C

Table 170. Real Time Clock (RTC) register map

Name	Size	Description	Access	Reset value[1]	Address
SEC	6	Seconds Counter	R/W	n/a	0xE002 4020
MIN	6	Minutes Register	R/W	n/a	0xE002 4024
HOUR	5	Hours Register	R/W	n/a	0xE002 4028
DOM	5	Day of Month Register	R/W	n/a	0xE002 402C
DOW	3	Day of Week Register	R/W	n/a	0xE002 4030
DOY	9	Day of Year Register	R/W	n/a	0xE002 4034
MONTH	4	Months Register	R/W	n/a	0xE002 4038
YEAR	12	Years Register	R/W	n/a	0xE002 403C
ALSEC	6	Alarm value for Seconds	R/W	n/a	0xE002 4060
ALMIN	6	Alarm value for Minutes	R/W	n/a	0xE002 4064
ALHOUR	5	Alarm value for Hours	R/W	n/a	0xE002 4068
ALDOM	5	Alarm value for Day of Month	R/W	n/a	0xE002 406C
ALDOW	3	Alarm value for Day of Week	R/W	n/a	0xE002 4070
ALDOY	9	Alarm value for Day of Year	R/W	n/a	0xE002 4074
ALMON	4	Alarm value for Months	R/W	n/a	0xE002 4078
ALYEAR	12	Alarm value for Year	R/W	n/a	0xE002 407C
PREINT	13	Prescaler value, integer portion	R/W	0	0xE002 4080
PREFRAC	15	Prescaler value, integer portion	R/W	0	0xE002 4084

[1] Registers in the RTC other than those that are part of the Prescaler are not affected by chip Reset. These registers must be initialized by software if the RTC is enabled. Reset value reflects the data stored in used bits only. It does not include reserved bits content.

5.1 RTC interrupts

Interrupt generation is controlled through the Interrupt Location Register (ILR), Counter Increment Interrupt Register (CIIR), the alarm registers, and the Alarm Mask Register (AMR). Interrupts are generated only by the transition into the interrupt state. The ILR separately enables CIIR and AMR interrupts. Each bit in CIIR corresponds to one of the time counters. If CIIR is enabled for a particular counter, then every time the counter is incremented an interrupt is generated. The alarm registers allow the user to specify a date and time for an interrupt to be generated. The AMR provides a mechanism to mask alarm compares. If all non-masked alarm registers match the value in their corresponding time counter, then an interrupt is generated.

5.2 Miscellaneous register group

Table 16–171 summarizes the registers located from 0 to 7 of A[6:2]. More detailed descriptions follow.

Table 171. Miscellaneous registers

Name	Size	Description	Access	Address
ILR	2	Interrupt Location. Reading this location indicates the source of an interrupt. Writing a one to the appropriate bit at this location clears the associated interrupt.	R/W	0xE002 4000
CTC	15	Clock Tick Counter. Value from the clock divider.	RO	0xE002 4004
CCR	4	Clock Control Register. Controls the function of the clock divider.	R/W	0xE002 4008
CIIR	8	Counter Increment Interrupt. Selects which counters will generate an interrupt when they are incremented.	R/W	0xE002 400C
AMR	8	Alarm Mask Register. Controls which of the alarm registers are masked.	R/W	0xE002 4010
CTIME0	32	Consolidated Time Register 0	RO	0xE002 4014
CTIME1	32	Consolidated Time Register 1	RO	0xE002 4018
CTIME2	32	Consolidated Time Register 2	RO	0xE002 401C

5.3 Interrupt Location Register (ILR - 0xE002 4000)

The Interrupt Location Register is a 2-bit register that specifies which blocks are generating an interrupt (see Table 16–172). Writing a one to the appropriate bit clears the corresponding interrupt. Writing a zero has no effect. This allows the programmer to read this register and write back the same value to clear only the interrupt that is detected by the read.

Table 172: Interrupt Location Register (ILR - address 0xE002 4000) bit description

Bit	Symbol	Description	Reset value
0	RTCCIF	When one, the Counter Increment Interrupt block generated an interrupt. Writing a one to this bit location clears the counter increment interrupt.	NA
1	RTCALF	When one, the alarm registers generated an interrupt. Writing a one to this bit location clears the alarm interrupt.	NA
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.4 Clock Tick Counter Register (CTCR - 0xE002 4004)

The Clock Tick Counter is read only. It can be reset to zero through the Clock Control Register (CCR). The CTC consists of the bits of the clock divider counter.

Table 173: Clock Tick Counter Register (CTCR - address 0xE002 4004) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:1	Clock Tick Counter	Prior to the Seconds counter, the CTC counts 32,768 clocks per second. Due to the RTC Prescaler, these 32,768 time increments may not all be of the same duration. Refer to the Section 16–7 “Reference clock divider (prescaler)” on page 217 for details.	NA

5.5 Clock Control Register (CCR - 0xE002 4008)

The clock register is a 5-bit register that controls the operation of the clock divide circuit. Each bit of the clock register is described in Table 16–174.

Table 174: Clock Control Register (CCR - address 0xE002 4008) bit description

Bit	Symbol	Description	Reset value
0	CLKEN	Clock Enable. When this bit is a one the time counters are enabled. When it is a zero, they are disabled so that they may be initialized.	NA
1	CTCRST	CTC Reset. When one, the elements in the Clock Tick Counter are reset. The elements remain reset until CCR[1] is changed to zero.	NA
3:2	CTTEST	Test Enable. These bits should always be zero during normal operation.	NA
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.6 Counter Increment Interrupt Register (CIIR - 0xE002 400C)

The Counter Increment Interrupt Register (CIIR) gives the ability to generate an interrupt every time a counter is incremented. This interrupt remains valid until cleared by writing a one to bit zero of the Interrupt Location Register (ILR[0]).

Table 175: Counter Increment Interrupt Register (CIIR - address 0xE002 400C) bit description

Bit	Symbol	Description	Reset value
0	IMSEC	When 1, an increment of the Second value generates an interrupt.	NA
1	IMMIN	When 1, an increment of the Minute value generates an interrupt.	NA
2	IMHOUR	When 1, an increment of the Hour value generates an interrupt.	NA
3	IMDOM	When 1, an increment of the Day of Month value generates an interrupt.	NA
4	IMDOW	When 1, an increment of the Day of Week value generates an interrupt.	NA
5	IMDOY	When 1, an increment of the Day of Year value generates an interrupt.	NA
6	IMMON	When 1, an increment of the Month value generates an interrupt.	NA
7	IMYEAR	When 1, an increment of the Year value generates an interrupt.	NA

5.7 Alarm Mask Register (AMR - 0xE002 4010)

The Alarm Mask Register (AMR) allows the user to mask any of the alarm registers. Table 16–176 shows the relationship between the bits in the AMR and the alarms. For the alarm function, every non-masked alarm register must match the corresponding time counter for an interrupt to be generated. The interrupt is generated only when the counter comparison first changes from no match to match. The interrupt is removed when a one is written to the appropriate bit of the Interrupt Location Register (ILR). If all mask bits are set, then the alarm is disabled.

Table 176: Alarm Mask Register (AMR - address 0xE002 4010) bit description

Bit	Symbol	Description	Reset value
0	AMRSEC	When 1, the Second value is not compared for the alarm.	NA
1	AMRMIN	When 1, the Minutes value is not compared for the alarm.	NA
2	AMRHOUR	When 1, the Hour value is not compared for the alarm.	NA
3	AMRDOM	When 1, the Day of Month value is not compared for the alarm.	NA
4	AMRDOW	When 1, the Day of Week value is not compared for the alarm.	NA
5	AMRDOY	When 1, the Day of Year value is not compared for the alarm.	NA
6	AMRMON	When 1, the Month value is not compared for the alarm.	NA
7	AMRYEAR	When 1, the Year value is not compared for the alarm.	NA

5.8 Consolidated time registers

The values of the Time Counters can optionally be read in a consolidated format which allows the programmer to read all time counters with only three read operations. The various registers are packed into 32-bit values as shown in Table 16–177, Table 16–178, and Table 16–179. The least significant bit of each register is read back at bit 0, 8, 16, or 24.

The Consolidated Time Registers are read only. To write new values to the Time Counters, the Time Counter addresses should be used.

5.9 Consolidated Time register 0 (CTIME0 - 0xE002 4014)

The Consolidated Time Register 0 contains the low order time values: Seconds, Minutes, Hours, and Day of Week.

Table 177: Consolidated Time register 0 (CTIME0 - address 0xE002 4014) bit description

Bit	Symbol	Description	Reset value
5:0	Seconds	Seconds value in the range of 0 to 59	NA
7:6	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
13:8	Minutes	Minutes value in the range of 0 to 59	NA
15:14	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
20:16	Hours	Hours value in the range of 0 to 23	NA
23:21	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	Day Of Week	Day of week value in the range of 0 to 6	NA
31:27	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.10 Consolidated Time register 1 (CTIME1 - 0xE002 4018)

The Consolidate Time register 1 contains the Day of Month, Month, and Year values.

Table 178: Consolidated Time register 1 (CTIME1 - address 0xE002 4018) bit description

Bit	Symbol	Description	Reset value
4:0	Day of Month	Day of month value in the range of 1 to 28, 29, 30, or 31 (depending on the month and whether it is a leap year).	NA
7:5	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
11:8	Month	Month value in the range of 1 to 12.	NA
15:12	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
27:16	Year	Year value in the range of 0 to 4095.	NA
31:28	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.11 Consolidated Time register 2 (CTIME2 - 0xE002 401C)

The Consolidate Time register 2 contains just the Day of Year value.

Table 179: Consolidated Time register 2 (CTIME2 - address 0xE002 401C) bit description

Bit	Symbol	Description	Reset value
11:0	Day of Year	Day of year value in the range of 1 to 365 (366 for leap years).	NA
31:12	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

5.12 Time counter group

The time value consists of the eight counters shown in Table 16–180 and Table 16–181. These counters can be read or written at the locations shown in Table 16–181.

Table 180. Time counter relationships and values

Counter	Size	Enabled by	Minimum value	Maximum value
Second	6	Clk1 (see Figure 16–56)	0	59
Minute	6	Second	0	59
Hour	5	Minute	0	23
Day of Month	5	Hour	1	28, 29, 30 or 31
Day of Week	3	Hour	0	6
Day of Year	9	Hour	1	365 or 366 (for leap year)
Month	4	Day of Month	1	12
Year	12	Month or day of Year	0	4095

Table 181. Time counter registers

Name	Size	Description	Access	Address
SEC	6	Seconds value in the range of 0 to 59	R/W	0xE002 4020
MIN	6	Minutes value in the range of 0 to 59	R/W	0xE002 4024
HOUR	5	Hours value in the range of 0 to 23	R/W	0xE002 4028

Table 181. Time counter registers

Name	Size	Description	Access	Address
DOM	5	Day of month value in the range of 1 to 28, 29, 30, or 31 (depending on the month and whether it is a leap year).[1]	R/W	0xE002 402C
DOW	3	Day of week value in the range of 0 to 6[1]	R/W	0xE002 4030
DOY	9	Day of year value in the range of 1 to 365 (366 for leap years)[1]	R/W	0xE002 4034
MONTH	4	Month value in the range of 1 to 12	R/W	0xE002 4038
YEAR	12	Year value in the range of 0 to 4095	R/W	0xE002 403C

[1] These values are simply incremented at the appropriate intervals and reset at the defined overflow point. They are not calculated and must be correctly initialized in order to be meaningful.

5.13 Leap year calculation

The RTC does a simple bit comparison to see if the two lowest order bits of the year counter are zero. If true, then the RTC considers that year a leap year. The RTC considers all years evenly divisible by 4 as leap years. This algorithm is accurate from the year 1901 through the year 2099, but fails for the year 2100, which is not a leap year. The only effect of leap year on the RTC is to alter the length of the month of February for the month, day of month, and year counters.

5.14 Alarm register group

The alarm registers are shown in Table 16–182. The values in these registers are compared with the time counters. If all the unmasked (See Section 16–5.7 “Alarm Mask Register (AMR - 0xE002 4010)” on page 213) alarm registers match their corresponding time counters then an interrupt is generated. The interrupt is cleared when a one is written to bit one of the Interrupt Location Register (ILR[1]).

Table 182. Alarm registers

Name	Size	Description	Access	Address
ALSEC	6	Alarm value for Seconds	R/W	0xE002 4060
ALMIN	6	Alarm value for Minutes	R/W	0xE002 4064
ALHOUR	5	Alarm value for Hours	R/W	0xE002 4068
ALDOM	5	Alarm value for Day of Month	R/W	0xE002 406C
ALDOW	3	Alarm value for Day of Week	R/W	0xE002 4070
ALDOY	9	Alarm value for Day of Year	R/W	0xE002 4074
ALMON	4	Alarm value for Months	R/W	0xE002 4078
ALYEAR	12	Alarm value for Years	R/W	0xE002 407C

6. RTC usage notes

Since the RTC operates from the APB clock (PCLK), any interruption of that clock will cause the time to drift away from the time value it would have provided otherwise. The variance could be to actual clock time if the RTC was initialized to that, or simply an error in elapsed time since the RTC was activated.

No provision is made in the LPC2104/05/06 to retain RTC status upon power loss, or to maintain time incrementation if the clock source is lost, interrupted, or altered. Loss of chip power will result in complete loss of all RTC register contents. Entry to Power Down mode will cause a lapse in the time update. Altering the RTC timebase during system operation (by reconfiguring the PLL, the APB timer, or the RTC prescaler) will result in some form of accumulated time error.

7. Reference clock divider (prescaler)

The reference clock divider (hereafter referred to as the prescaler) allows generation of a 32.768 kHz reference clock from any peripheral clock frequency greater than or equal to 65.536 kHz (2 × 32.768 kHz). This permits the RTC to always run at the proper rate regardless of the peripheral clock rate. Basically, the Prescaler divides the peripheral clock (PCLK) by a value which contains both an integer portion and a fractional portion. The result is not a continuous output at a constant frequency, some clock periods will be one PCLK longer than others. However, the overall result can always be 32,768 counts per second.

The reference clock divider consists of a 13-bit integer counter and a 15-bit fractional counter. The reasons for these counter sizes are as follows:

1. For frequencies that are expected to be supported by the LPC2104/05/06, a 13-bit integer counter is required. This can be calculated as 160 MHz divided by 32,768 minus 1 = 4881 with a remainder of 26,624. Thirteen bits are needed to hold the value 4881, but actually supports frequencies up to 268.4 MHz (32,768 × 8192).
2. The remainder value could be as large as 32,767, which requires 15 bits.

Table 183. Reference clock divider registers

Name	Size	Description	Access	Address
PREINT	13	Prescale Value, integer portion	R/W	0xE002 4080
PREFRAC	15	Prescale Value, fractional portion	R/W	0xE002 4084

7.1 Prescaler Integer register (PREINT - 0xE002 4080)

This is the integer portion of the prescale value, calculated as:

PREINT = int (PCLK / 32768) − 1. The value of PREINT must be greater than or equal to 1.

Table 184: Prescaler Integer register (PREINT - address 0xE002 4080) bit description

Bit	Symbol	Description	Reset value
12:0	Prescaler Integer	Contains the integer portion of the RTC prescaler value.	0
15:13	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7.2 Prescaler Fraction register (PREFRAC - 0xE002 4084)

This is the fractional portion of the prescale value, and may be calculated as:

PREFRAC = PCLK − ((PREINT + 1) × 32768).

Table 185: Prescaler Integer register (PREFRAC - address 0xE002 4084) bit description

Bit	Symbol	Description	Reset value
14:0	Prescaler Fraction	Contains the integer portion of the RTC prescaler value.	0
15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

7.3 Example of prescaler usage

In a simplistic case, the PCLK frequency is 65.537 kHz. So:

PREINT = int (PCLK / 32768) − 1 = 1 and
PREFRAC = PCLK - ([PREINT + 1] × 32768) = 1

With this prescaler setting, exactly 32,768 clocks per second will be provided to the RTC by counting 2 PCLKs 32,767 times, and 3 PCLKs once.

In a more realistic case, the PCLK frequency is 10 MHz. Then,

PREINT = int (PCLK / 32768) − 1 = 304 and
PREFRAC = PCLK − ([PREINT + 1] × 32768) = 5,760.

In this case, 5,760 of the prescaler output clocks will be 306 (305 + 1) PCLKs long, the rest will be 305 PCLKs long.

In a similar manner, any PCLK rate greater than 65.536 kHz (as long as it is an even number of cycles per second) may be turned into a 32 kHz reference clock for the RTC. The only caveat is that if PREFRAC does not contain a zero, then not all of the 32,768 per second clocks are of the same length. Some of the clocks are one PCLK longer than others. While the longer pulses are distributed as evenly as possible among the remaining pulses, this "jitter" could possibly be of concern in an application that wishes to observe the contents of the Clock Tick Counter (CTC) directly(Section 16–5.4 “Clock Tick Counter Register (CTCR - 0xE002 4004)” on page 212).

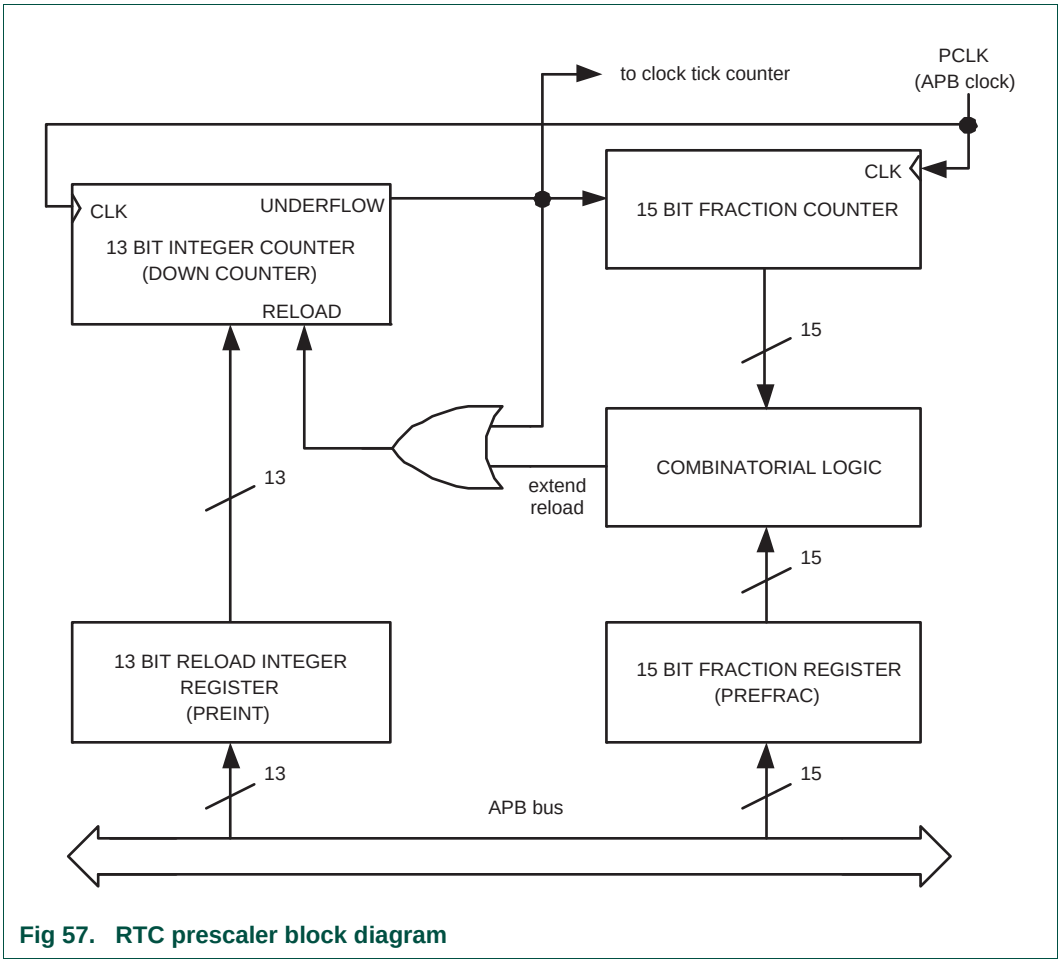


Fig 57. RTC prescaler block diagram

7.4 Prescaler operation

The Prescaler block labelled "Combination Logic" in Figure 16–57 determines when the decrement of the 13-bit PREINT counter is extended by one PCLK. In order to both insert the correct number of longer cycles, and to distribute them evenly, the combinatorial Logic associates each bit in PREFRAC with a combination in the 15-bit Fraction Counter. These associations are shown in the following Table 16–186.

For example, if PREFRAC bit 14 is a one (representing the fraction 1/2), then half of the cycles counted by the 13-bit counter need to be longer. When there is a 1 in the LSB of the Fraction Counter, the logic causes every alternate count (whenever the LSB of the Fraction Counter=1) to be extended by one PCLK, evenly distributing the pulse widths. Similarly, a one in PREFRAC bit 13 (representing the fraction 1/4) will cause every fourth cycle (whenever the two LSBs of the Fraction Counter=10) counted by the 13-bit counter to be longer.

Table 186. Prescaler cases where the Integer Counter reload value is incremented

Fraction Counter	PREFRAC Bit														
	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
--- ---- ---- ---1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
--- ---- ---- --10	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
--- ---- ---- -100	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
--- ---- ---- 1000	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
--- ---- ---1 0000	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
--- ---- --10 0000	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-
--- ---- -100 0000	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
--- ---- 1000 0000	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-
--- ---1 0000 0000	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-
--- --10 0000 0000	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-
--- -100 0000 0000	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-
--- 1000 0000 0000	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
--1 0000 0000 0000	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-
-10 0000 0000 0000	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-
100 0000 0000 0000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1

UM10275

Chapter 17: LPC2104/05/06 WatchDog Timer (WDT)

Rev. 02 — 8 April 2009 **User manual**

1. Features

- Internally resets chip if not periodically reloaded.
- Debug mode.
- Enabled by software but requires a hardware reset or a watchdog reset/interrupt to be disabled.
- Incorrect/Incomplete feed sequence causes reset/interrupt if enabled.
- Flag to indicate Watchdog reset.
- Programmable 32-bit timer with internal pre-scaler.
- Selectable time period from (T_{PCLK} x 256 x 4) to (T_{PCLK} x 2^{32} x 4) in multiples of T_{PCLK} x 4.

2. Applications

The purpose of the watchdog is to reset the microcontroller within a reasonable amount of time if it enters an erroneous state. When enabled, the watchdog will generate a system reset if the user program fails to "feed" (or reload) the watchdog within a predetermined amount of time.

For interaction of the on-chip watchdog and other peripherals, especially the reset and boot-up procedures, please read Section 3–11 of this document.

3. Description

The watchdog consists of a divide by 4 fixed pre-scaler and a 32-bit counter. The clock is fed to the timer via a pre-scaler. The timer decrements when clocked. The minimum value from which the counter decrements is 0xFF. Setting a value lower than 0xFF causes 0xFF to be loaded in the counter. Hence the minimum watchdog interval is (T_{PCLK} x 256 x 4) and the maximum watchdog interval is (T_{PCLK} x 2^{32} x 4) in multiples of (T_{PCLK} x 4). The watchdog should be used in the following manner:

- Set the watchdog timer constant reload value in WDTC register.
- Setup mode in WDMOD register.
- Start the watchdog by writing 0xAA followed by 0x55 to the WDFEED register.
- Watchdog should be fed again before the watchdog counter underflows to prevent reset/interrupt.

When the Watchdog counter underflows, the program counter will start from 0x0000 0000 as in the case of external reset. The Watchdog Time-Out Flag (WDTOF) can be examined to determine if the watchdog has caused the reset condition. The WDTOF flag must be cleared by software.

4. Register description

The watchdog contains 4 registers as shown in Table 17–187 below.

Table 187. Watchdog register map

Name	Description	Access	Reset value[1]	Address
WDMOD	Watchdog Mode register. This register contains the basic mode and status of the Watchdog Timer.	R/W	0	0xE000 0000
WDTC	Watchdog Timer Constant register. This register determines the time-out value.	R/W	0xFF	0xE000 0004
WDFEED	Watchdog Feed sequence register. Writing 0xAA followed by 0x55 to this register reloads the Watchdog timer to its preset value.	WO	NA	0xE000 0008
WDTV	Watchdog Timer Value register. This register reads out the current value of the Watchdog timer.	RO	0xFF	0xE000 000C

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

4.1 Watchdog Mode register (WDMOD - 0xE000 0000)

The WDMOD register controls the operation of the watchdog as per the combination of WDEN and RESET bits.

Table 188. Watchdog operating modes selection

WDEN	WDRESET	Mode of Operation
0	X (0 or 1)	Debug/Operate without the watchdog running.
1	0	Watchdog Interrupt Mode: debug with the Watchdog interrupt but no WDRESET enabled. When this mode is selected, a watchdog counter underflow will set the WDINT flag and the watchdog interrupt request will be generated.
1	1	Watchdog Reset Mode: operate with the watchdog interrupt and WDRESET enabled. When this mode is selected, a watchdog counter underflow will reset the microcontroller. While the watchdog interrupt is also enabled in this case (WDEN = 1) it will not be recognized since the watchdog reset will clear the WDINT flag.

Once the **WDEN** and/or **WDRESET** bits are set they can not be cleared by software. Both flags are cleared by an external reset or a watchdog timer underflow.

WDTOF The Watchdog Time-Out Flag is set when the watchdog times out. This flag is cleared by software.

WDINT The Watchdog Interrupt Flag is set when the watchdog times out. This flag is cleared when any reset occurs. Once the watchdog interrupt is serviced, it can be disabled in the VIC or the watchdog interrupt request will be generated indefinitely.

Table 189: Watchdog Mode register (WDMOD - address 0xE000 0000) bit description

Bit	Symbol	Description	Reset value
0	WDEN	WDEN Watchdog interrupt Enable bit (Set Only).	0
1	WDRESET	WDRESET Watchdog Reset Enable bit (Set Only).	0
2	WDTOF	WDTOF Watchdog Time-Out Flag.	0 (Only after external reset)
3	WDINT	WDINT Watchdog interrupt Flag (Read Only).	0
7:4	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

4.2 Watchdog Timer Constant register (WDTC - 0xE000 0004)

The WDTC register determines the time-out value. Every time a feed sequence occurs the WDTC content is reloaded in to the watchdog timer. It's a 32-bit register with 8 LSB set to 1 on reset. Writing values below 0xFF will cause 0xFF to be loaded to the WDTC. Thus the minimum time-out interval is $T_{PCLK} \times 256 \times 4$.

Table 190: Watchdog Timer Constant register (WDTC - address 0xE000 0004) bit description

Bit	Symbol	Description	Reset value
31:0	Count	Watchdog time-out interval.	0x0000 00FF

4.3 Watchdog Feed register (WDFEED - 0xE000 0008)

Writing 0xAA followed by 0x55 to this register will reload the watchdog timer to the WDTC value. This operation will also start the watchdog if it is enabled via the WDMOD register. Setting the WDEN bit in the WDMOD register is not sufficient to enable the watchdog. A valid feed sequence must first be completed before the Watchdog is capable of generating an interrupt/reset. Until then, the watchdog will ignore feed errors. After writing 0xAA to WFEED, access to any WatchDog register other than writing 0x55 to WFEED causes an immediate reset/interrupt when the WatchDog is enabled. The reset/interrupt will be generated during the second PCLK following an incorrect access to a watchdog timer register during a feed sequence.

Remark: Interrupts must be disabled during the feed sequence. An abort condition will occur if an interrupt happens during the feed sequence.

Table 191: Watchdog Feed register (WDFEED - address 0xE000 0008) bit description

Bit	Symbol	Description	Reset value
7:0	Feed	Feed value should be 0xAA followed by 0x55.	NA

4.4 Watchdog Timer Value register (WDTV - 0xE000 000C)

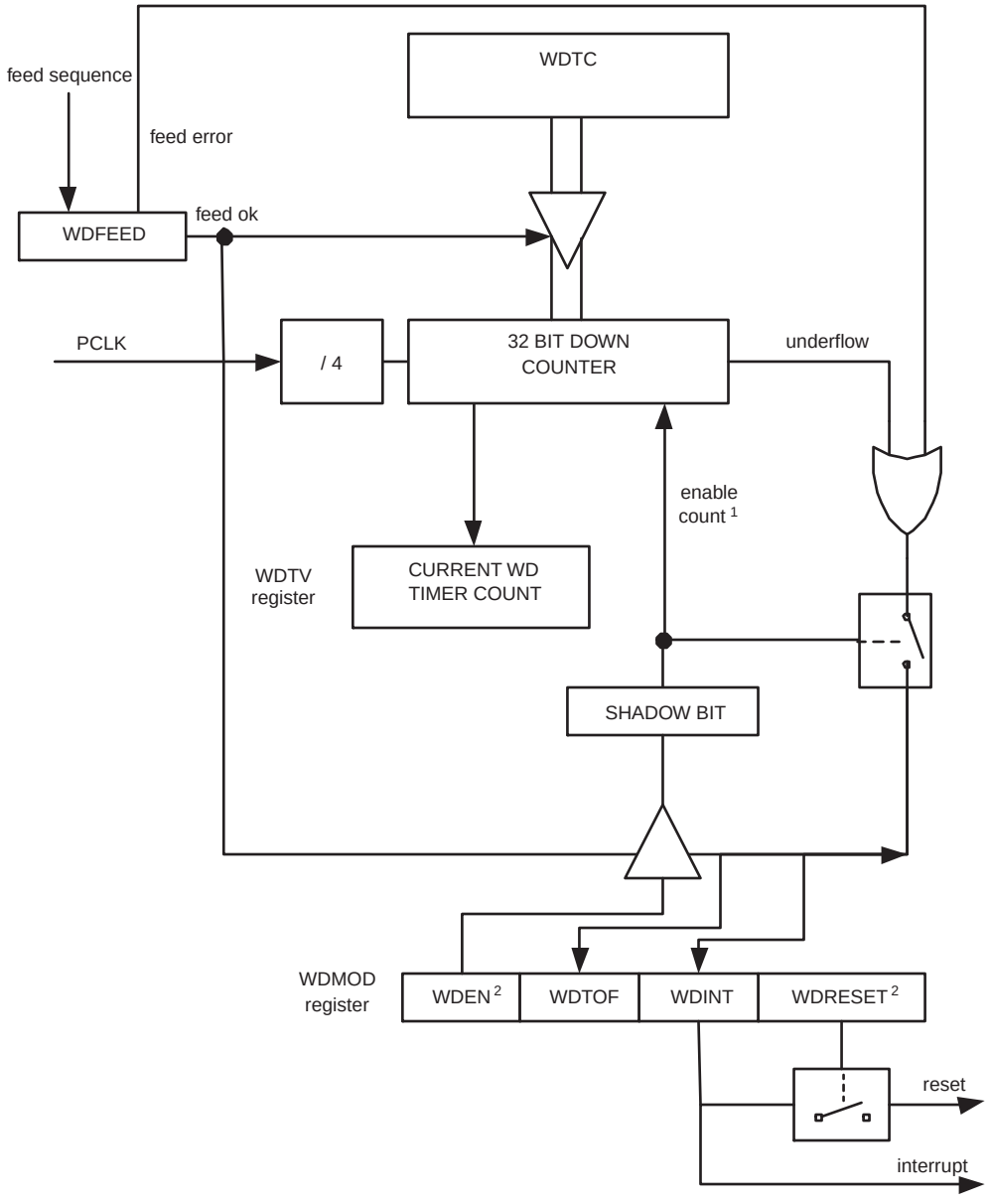
The WDTV register is used to read the current value of watchdog timer.

Table 192: Watchdog Timer Value register (WDTV - address 0xE000 000C) bit description

Bit	Symbol	Description	Reset value
31:0	Count	Counter timer value.	0x0000 00FF

5. Block diagram

The block diagram of the Watchdog is shown below in the Figure 17–58.



(1) Counter is enabled only when the WDEN bit is set and a valid feed sequence is done.

(2) WDEN and WDRESET are sticky bits. Once set they can't be cleared until the watchdog underflows or an external reset occurs.

Fig 58. Watchdog block diagram

UM10275

Chapter 18: LPC2104/05/06 Flash memory and system programming

Rev. 02 — 8 April 2009 User manual

1. How to read this chapter

The level of Ccode Read Protection (CRP) depends on the specific part:

- LPC2104/05/06: no CRP.
- LPC2104/05/06/00: no CRP.
- LPC2104/05/06/01: CRP1/2/3 levels enabled (see Table 18–194).

2. Flash boot loader

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

3. Features

- In-System Programming: In-System programming (ISP) means programming or reprogramming the on-chip flash memory using the boot loader software and a serial port. This can be done when the part resides in the end-user board.
- In Application Programming: In-Application (IAP) programming means performing erase and write operation on the on-chip flash memory, as directed by the end-user application code.

4. Applications

The flash boot loader provides both In-System and In-Application programming interfaces for programming the on-chip flash memory.

5. Description

The flash boot loader code is executed every time the part is powered on or reset. The loader can execute the ISP command handler or the user application code. A LOW level after reset at the P0.14 pin is considered as an external hardware request to start the ISP command handler. Assuming that a proper signal is present on XTAL1 pin when the rising edge on $\overline{\text{RESET}}$ pin is generated, it may take up to 3 ms before P0.14 is sampled and the decision on whether to continue with user code or ISP handler is made. If P0.14 is sampled low and the watchdog overflow flag is set, the external hardware request to start the ISP command handler is ignored. If there is no request for the ISP command handler execution (P0.14 is sampled HIGH after reset), a search is made for a valid user program. If a valid user program is found then the execution control is transferred to it. If a valid user program is not found, the auto-baud routine is invoked.

Pin P0.14, which is used as hardware request for ISP, requires special attention. Since P0.14 is in high impedance mode after reset, it is important that the user provides external hardware (a pull-up resistor or other device) to put the pin in a defined state. Otherwise unintended entry into ISP mode may occur.

5.1 Memory map after any reset

The boot block is 8 kB in size and resides in the top portion (starting from 0x0001 E000 for devices with 128 kB flash) of the on-chip flash memory. After any reset the entire boot block is also mapped to the top of the on-chip memory space. i.e. the boot block is also visible in the memory region starting from the address 0x7FFF E000. The flash boot loader is designed to run from this memory area, but both the ISP and IAP software use parts of the on-chip RAM. The RAM usage is described later in this chapter. The interrupt vectors residing in the boot block of the on-chip flash memory also become active after reset, i.e., the bottom 64 bytes of the boot block are also visible in the memory region starting from the address 0x0000 0000. The reset vector contains a jump instruction to the entry point of the flash boot loader software.

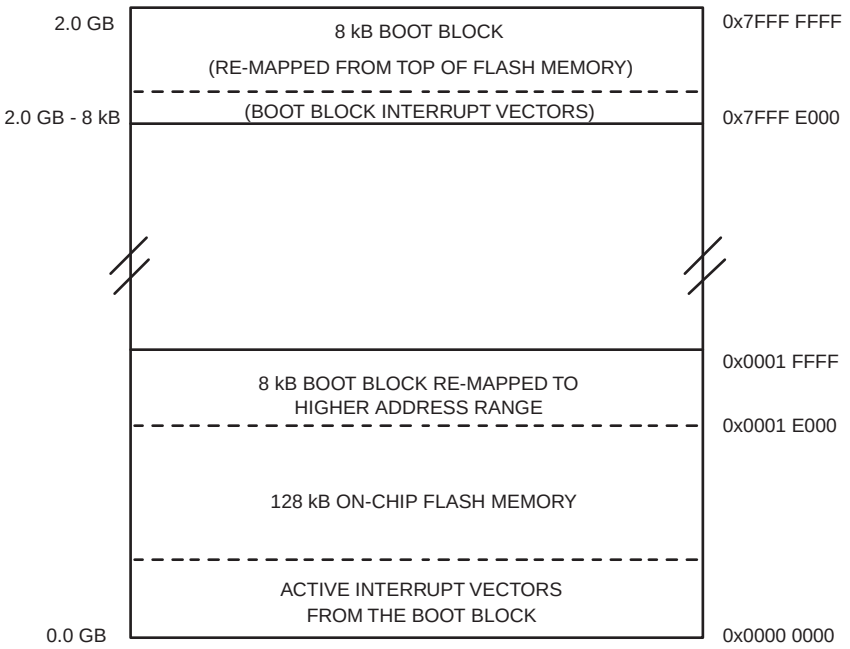


Fig 59. Map of lower memory after reset for 128 kB flash devices

5.2 Criterion for valid user code

The reserved ARM interrupt vector location (0x0000 0014) should contain the 2's complement of the check-sum of the remaining interrupt vectors. This causes the checksum of all of the vectors together to be 0. The boot loader code disables the overlaying of the interrupt vectors from the boot block, then checksums the interrupt vectors in sector 0 of the flash. If the signatures match then the execution control is transferred to the user code by loading the program counter with 0x0000 0000. Hence the user flash reset vector should contain a jump instruction to the entry point of the user application code.

If the signature is not valid, the auto-baud routine synchronizes with the host via serial port 0. The host should send a '?' (0x3F) as a synchronization character and wait for a response. The host side serial port settings should be 8 data bits, 1 stop bit and no parity. The auto-baud routine measures the bit time of the received synchronization character in terms of its own frequency and programs the baud rate generator of the serial port. It also sends an ASCII string ("Synchronized<CR><LF>") to the Host. In response to this host should send the same string ("Synchronized<CR><LF>"). The auto-baud routine looks at the received characters to verify synchronization. If synchronization is verified then "OK<CR><LF>" string is sent to the host. Host should respond by sending the crystal frequency (in kHz) at which the part is running. For example, if the part is running at 10 MHz, the response from the host should be "10000<CR><LF>". "OK<CR><LF>" string is sent to the host after receiving the crystal frequency. If synchronization is not verified then the auto-baud routine waits again for a synchronization character. For auto-baud to work correctly, the crystal frequency should be greater than or equal to 10 MHz. The on-chip PLL is not used by the boot code.

Once the crystal frequency is received the part is initialized and the ISP command handler is invoked. For safety reasons an "Unlock" command is required before executing the commands resulting in flash erase/write operations and the "Go" command. The rest of the commands can be executed without the unlock command. The Unlock command is required to be executed once per ISP session. The Unlock command is explained in Section 18–9 "ISP commands" on page 233.

5.3 Communication protocol

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

5.4 ISP command format

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

5.5 ISP response format

"Return_Code<CR><LF>Response_0<CR><LF>Response_1<CR><LF> ... Response_n<CR><LF>" "Data" (Data only for Read commands)

5.6 ISP data format

The data stream is in UU-encode format. The UU-encode algorithm converts 3 bytes of binary data in to 4 bytes of printable ASCII character set. It is more efficient than Hex format which converts 1 byte of binary data in to 2 bytes of ASCII hex. The sender should send the check-sum after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters(bytes) i.e. it can hold 45 data bytes. The receiver should compare it with the check-sum of the received bytes. If the check-sum matches then the receiver should respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match the receiver should respond with "RESEND<CR><LF>". In response the sender should retransmit the bytes.

5.7 ISP flow control

A software XON/XOFF flow control scheme is used to prevent data loss due to buffer overrun. When the data arrives rapidly, the ASCII control character DC3 (stop) is sent to stop the flow of data. Data flow is resumed by sending the ASCII control character DC1 (start). The host should also support the same flow control scheme.

5.8 ISP command abort

Commands can be aborted by sending the ASCII control character "ESC". This feature is not documented as a command under "ISP Commands" section. Once the escape code is received the ISP command handler waits for a new command.

5.9 Interrupts during ISP

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

5.10 Interrupts during IAP

The on-chip flash memory is not accessible during erase/write operations. When the user application code starts executing the interrupt vectors from the user flash area are active. The user should either disable interrupts, or ensure that user interrupt vectors are active in RAM and that the interrupt handlers reside in RAM, before making a flash erase/write IAP call. The IAP code does not use or disable interrupts.

5.11 RAM used by ISP command handler

ISP commands use on-chip RAM from 0x4000 0120 to 0x4000 01FF. The user could use this area, but the contents may be lost upon reset. Flash programming commands use the top 32 bytes of on-chip RAM. The stack is located at RAM top − 32. The maximum stack usage is 256 bytes and it grows downwards.

5.12 RAM used by IAP command handler

Flash programming commands use the top 32 bytes of on-chip RAM. The maximum stack usage in the user allocated stack space is 128 bytes and it grows downwards.

5.13 RAM used by RealMonitor

The RealMonitor uses on-chip RAM from 0x4000 0040 to 0x4000 011F. The user could use this area if RealMonitor based debug is not required. The Flash boot loader does not initialize the stack for RealMonitor.

5.14 Boot process flowchart

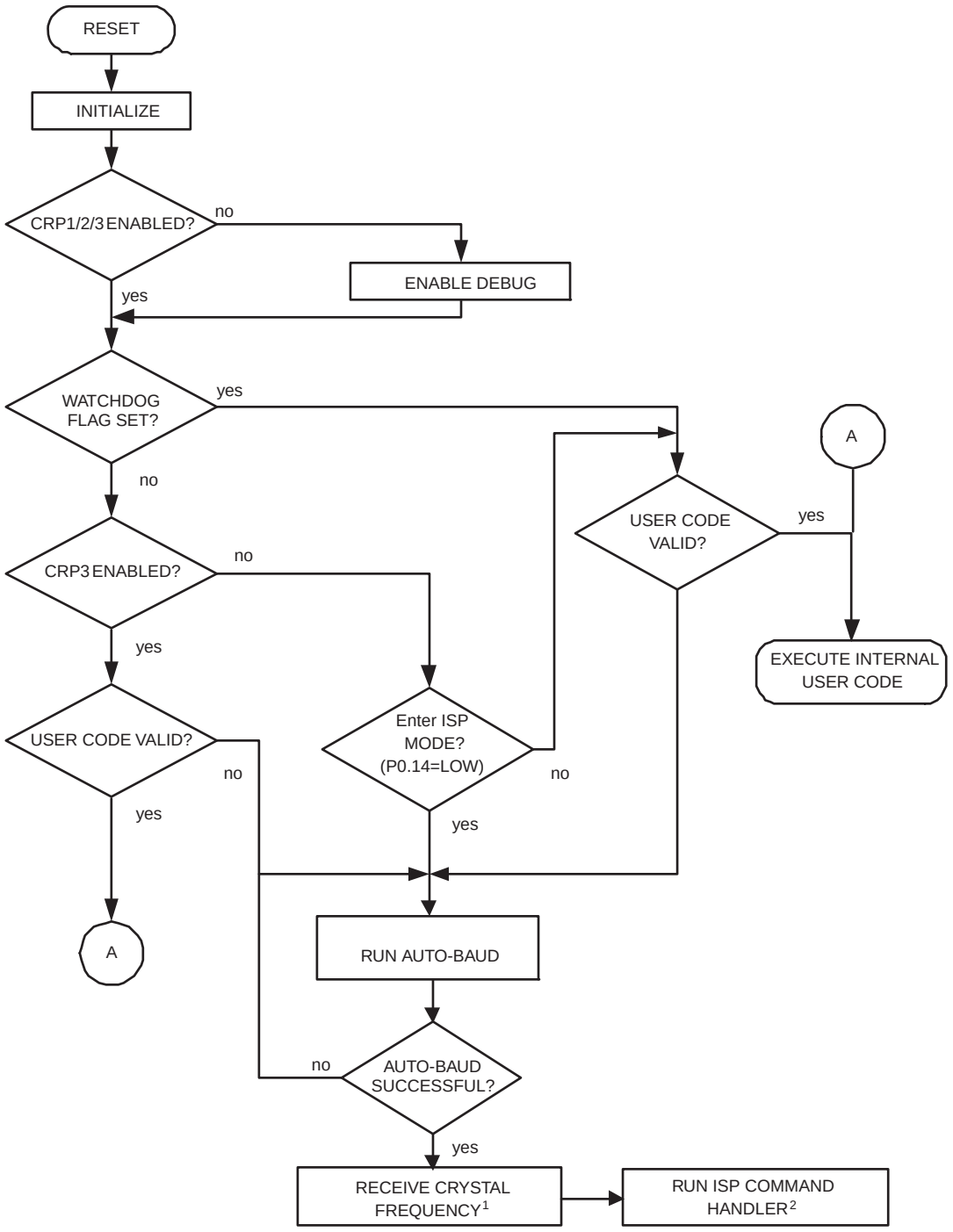


Fig 60. Boot process flow chart for LPC2104/05/06/01 with three CRP levels (CRP1/2/3) (see Section 18–1)

6. Sector numbers

Some IAP and ISP commands operate on "sectors" and specify sector numbers. The following table indicates the correspondence between sector numbers and memory addresses for LPC2104/05/06 devices containing 128 kB of flash. IAP, ISP, and RealMonitor routines are located in the boot block. The boot block is present at the top of each flash memory. Because of the boot block 120 kB of the 128 kB flash device is available for user code. ISP and IAP commands do not allow write/erase/go operation on the boot block.

Table 193. Flash sectors

Sector Number	Sector Size [kB] 128 kB flash	Address Range
0	8	0x0000 0000 - 0x0000 1FFF
1	8	0x0000 2000 - 0x0000 3FFF
2	8	0x0000 4000 - 0x0000 5FFF
3	8	0x0000 6000 - 0x0000 7FFF
4	8	0x0000 8000 - 0x0000 9FFF
5	8	0x0000 A000 - 0x0000 BFFF
6	8	0x0000 C000 - 0x0000 DFFF
7	8	0x0000 E000 - 0x0000 FFFF
8	8	0x0001 0000 - 0x0001 1FFF
9	8	0x0001 2000 - 0x0001 3FFF
10 (0x0A)	8	0x0001 4000 - 0x0001 5FFF
11 (0x0B)	8	0x0001 6000 - 0x0001 7FFF
12 (0x0C)	8	0x0001 8000 - 0x0001 9FFF
13 (0x0D)	8	0x0001 A000 - 0x0001 BFFF
14 (0x0E)	8	0x0001 C000 - 0x0001 DFFF
15 (0x0F)	8	0x0001 E000 - 0x0001 FFFF

7. Flash content protection mechanism

The LPC2104/05/06 is equipped with the Error Correction Code (ECC) capable flash memory. The purpose of an error correction module is twofold. Firstly, it decodes data words read from the memory into output data words. Secondly, it encodes data words to be written to the memory. The error correction capability consists of single bit error correction with Hamming code.

The operation of ECC is transparent to the running application. The ECC content itself is stored in a flash memory not accessible by user’s code to either read from it or write into it on its own. A byte of ECC corresponds to every consecutive 128 bits of the user accessible Flash. Consequently, Flash bytes from 0x0000 0000 to 0x0000 000F are protected by the first ECC byte, Flash bytes from 0x0000 0010 to 0x0000 001F are protected by the second ECC byte, etc.

Whenever the CPU requests a read from Flash, both 128 bits of raw data containing the specified memory location and the matching ECC byte are evaluated. If the ECC mechanism detects a single error in the fetched data, a correction will be applied before

data are provided to the CPU. When a write request into the user's Flash is made, write of user specified content is accompanied by a matching ECC value calculated and stored in the ECC memory.

When a sector of user's Flash memory is erased, corresponding ECC bytes are also erased. Once an ECC byte is written, it can not be updated unless it is erased first. Therefore, for the implemented ECC mechanism to perform properly, data must be written into the flash memory in groups of 16 bytes (or multiples of 16), aligned as described above.

8. Code Read Protection (CRP)

Code Read Protection is a mechanism that allows user to enable different levels of security in the system so that access to the on-chip Flash and use of the ISP can be restricted. When needed, CRP is invoked by programming a specific pattern in Flash location at 0x0000 01FC. IAP commands are not affected by the code read protection.

Remark: Starting with bootloader version 1.53 three levels of CRP are implemented on the LPC2104/05/06/01.

Important: any CRP change becomes effective only after reset.

Table 194. Code Read Protection options

Name	Pattern programmed in 0x000001FC	Description
CRP1	0x12345678	Access to chip via the JTAG pins is disabled. This mode allows partial Flash update using the following ISP commands and restrictions: • Write to RAM command can not access RAM below 0x40000200 • Copy RAM to Flash command can not write to Sector 0 • Erase command can erase Sector 0 only when all sectors are selected for erase • Compare command is disabled This mode is useful when CRP is required and Flash field updates are needed but all sectors can not be erased. Since compare command is disabled in case of partial updates the secondary loader should implement checksum mechanism to verify the integrity of the Flash.
CRP2	0x87654321	Access to chip via the JTAG pins is disabled. The following ISP commands are disabled: • Read Memory • Write to RAM • Go • Copy RAM to Flash • Compare When CRP2 is enabled the ISP erase command only allows erasure of all user sectors.
CRP3	0x43218765	Access to chip via the JTAG pins is disabled. ISP entry by pulling P0.14 LOW is disabled if a valid user code is present in Flash sector 0. This mode effectively disables ISP override using P0.14 pin. It is up to the user's application to provide need Flash update mechanism using IAP calls if necessary. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Table 195. Code Read Protection hardware/software interaction

CRP option	User Code Valid	P0.14 pin at reset	JTAG enabled	enter ISP mode	partial Flash update in ISP mode
No	No	X	Yes	Yes	Yes
No	Yes	High	Yes	No	NA
No	Yes	Low	Yes	Yes	Yes
CRP1	Yes	High	No	No	NA
CRP1	Yes	Low	No	Yes	Yes
CRP2	Yes	High	No	No	NA
CRP2	Yes	Low	No	Yes	No
CRP3	Yes	x	No	No	NA
CRP1	No	x	No	Yes	Yes
CRP2	No	x	No	Yes	No
CRP3	No	x	No	Yes	No

In case a CRP mode is enabled and access to the chip is allowed via the ISP, an unsupported or restricted ISP command will be terminated with return code CODE_READ_PROTECTION_ENABLED.

9. ISP commands

The following commands are accepted by the ISP command handler. Detailed status codes are supported for each command. The command handler sends the return code INVALID_COMMAND when an undefined command is received. Commands and return codes are in ASCII format.

CMD_SUCCESS is sent by ISP command handler only when received ISP command has been completely executed and the new ISP command can be given by the host. Exceptions from this rule are "Set Baud Rate", "Write to RAM", "Read Memory", and "Go" commands.

Table 196. ISP command summary

ISP Command	Usage	Described in
Unlock	U <Unlock Code>	Table 18–197
Set Baud Rate	B <Baud Rate> <stop bit>	Table 18–198
Echo	A <setting>	Table 18–200
Write to RAM	W <start address> <number of bytes>	Table 18–201
Read Memory	R <address> <number of bytes>	Table 18–202
Prepare sector(s) for write operation	P <start sector number> <end sector number>	Table 18–203
Copy RAM to Flash	C <Flash address> <RAM address> <number of bytes>	Table 18–204
Go	G <address> <Mode>	Table 18–205
Erase sector(s)	E <start sector number> <end sector number>	Table 18–206
Blank check sector(s)	I <start sector number> <end sector number>	Table 18–207
Read Part ID	J	Table 18–208
Read Boot code version	K	Table 18–210
Compare	M <address1> <address2> <number of bytes>	Table 18–211

9.1 Unlock <unlock code>

Table 197. ISP Unlock command

Command	U
Input	Unlock code: 23130_{10}
Return Code	CMD_SUCCESS | INVALID_CODE | PARAM_ERROR
Description	This command is used to unlock flash Write, Erase, and Go commands.
Example	"U 23130<CR><LF>" unlocks the flash Write/Erase & Go commands.

9.2 Set Baud Rate <baud rate> <stop bit>

Table 198. ISP Set Baud Rate command

Command	B
Input	Baud Rate: 9600 | 19200 | 38400 | 57600 | 115200 | 230400 Stop bit: 1 | 2
Return Code	CMD_SUCCESS | INVALID_BAUD_RATE | INVALID_STOP_BIT | PARAM_ERROR
Description	This command is used to change the baud rate. The new baud rate is effective after the command handler sends the CMD_SUCCESS return code.
Example	"B 57600 1<CR><LF>" sets the serial port to baud rate 57600 bps and 1 stop bit.

Table 199. Correlation between possible ISP baudrates and external crystal frequency (in MHz)

ISP Baudrate .vs. External Crystal Frequency	9600	19200	38400	57600	115200	230400
10.0000	+	+	+			
11.0592	+	+		+		
12.2880	+	+	+			
14.7456	+	+	+	+	+	+
15.3600	+					
18.4320	+	+		+		
19.6608	+	+	+			
24.5760	+	+	+			
25.0000	+	+	+			

9.3 Echo <setting>

Table 200. ISP Echo command

Command	A
Input	Setting: ON = 1 | OFF = 0
Return Code	CMD_SUCCESS | PARAM_ERROR
Description	The default setting for echo command is ON. When ON the ISP command handler sends the received serial data back to the host.
Example	"A 0<CR><LF>" turns echo off.

9.4 Write to RAM <start address> <number of bytes>

The host should send the data only after receiving the CMD_SUCCESS return code. The host should send the check-sum after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters(bytes) i.e. it can hold 45 data bytes. When the data fits in less then 20 UU-encoded lines then the check-sum should be of the actual number of bytes sent. The

ISP command handler compares it with the check-sum of the received bytes. If the check-sum matches, the ISP command handler responds with "OK<CR><LF>" to continue further transmission. If the check-sum does not match, the ISP command handler responds with "RESEND<CR><LF>". In response the host should retransmit the bytes.

Table 201. ISP Write to RAM command

Command	W
Input	**Start Address:** RAM address where data bytes are to be written. This address should be a word boundary. **Number of Bytes:** Number of bytes to be written. Count should be a multiple of 4
Return Code	CMD_SUCCESS | ADDR_ERROR (Address not on word boundary) | ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not multiple of 4) | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to download data to RAM. Data should be in UU-encoded format. This command is blocked when code read protection is enabled.
Example	"W 1073742336 4<CR><LF>" writes 4 bytes of data to address 0x4000 0200.

9.5 Read memory <address> <no. of bytes>

The data stream is followed by the command success return code. The check-sum is sent after transmitting 20 UU-encoded lines. The checksum is generated by adding raw data (before UU-encoding) bytes and is reset after transmitting 20 UU-encoded lines. The length of any UU-encoded line should not exceed 61 characters(bytes) i.e. it can hold 45 data bytes. When the data fits in less then 20 UU-encoded lines then the check-sum is of actual number of bytes sent. The host should compare it with the checksum of the received bytes. If the check-sum matches then the host should respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match then the host should respond with "RESEND<CR><LF>". In response the ISP command handler sends the data again.

Table 202. ISP Read memory command

Command	R
Input	**Start Address:** Address from where data bytes are to be read. This address should be a word boundary. **Number of Bytes:** Number of bytes to be read. Count should be a multiple of 4.
Return Code	CMD_SUCCESS followed by <actual data (UU-encoded)> | ADDR_ERROR (Address not on word boundary) | ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not a multiple of 4) | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to read data from RAM or Flash memory. This command is blocked when code read protection is enabled.
Example	"R 1073741824 4<CR><LF>" reads 4 bytes of data from address 0x4000 0000.

9.6 Prepare sector(s) for write operation <start sector number> <end sector number>

This command makes flash write/erase operation a two step process.

Table 203. ISP Prepare sector(s) for write operation command

Command	P
Input	**Start Sector Number** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR | PARAM_ERROR
Description	This command must be executed before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot block can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.
Example	"P 0 0<CR><LF>" prepares the flash sector 0.

9.7 Copy RAM to Flash <Flash address> <RAM address> <no of bytes>

Table 204. ISP Copy command

Command	C
Input	**Flash Address(DST):** Destination Flash address where data bytes are to be written. The destination address should be a 256 byte boundary. **RAM Address(SRC):** Source RAM address from where data bytes are to be read. **Number of Bytes:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096.
Return Code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not on word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR WRITE_OPERATION | BUSY | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to program the flash memory. The "Prepare Sector(s) for Write Operation" command should precede this command. The affected sectors are automatically protected again once the copy command is successfully executed. The boot block cannot be written by this command. This command is blocked when code read protection is enabled.
Example	"C 0 1073774592 512<CR><LF>" copies 512 bytes from the RAM address 0x4000 8000 to the flash address 0.

9.8 Go <address> <mode>

Table 205. ISP Go command

Command	G
Input	**Address:** Flash or RAM address from which the code execution is to be started. This address should be on a word boundary. **Mode:** T (Execute program in Thumb Mode) | A (Execute program in ARM mode).
Return Code	CMD_SUCCESS | ADDR_ERROR | ADDR_NOT_MAPPED | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to execute a program residing in RAM or Flash memory. It may not be possible to return to the ISP command handler once this command is successfully executed. This command is blocked when code read protection is enabled.
Example	"G 0 A<CR><LF>" branches to address 0x0000 0000 in ARM mode.

9.9 Erase sector(s) <start sector number> <end sector number>

Table 206. ISP Erase sector command

Command	E
Input	**Start Sector Number** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | CMD_LOCKED | PARAM_ERROR | CODE_READ_PROTECTION_ENABLED
Description	This command is used to erase one or more sector(s) of on-chip Flash memory. The boot block can not be erased using this command. This command only allows erasure of all user sectors when the code read protection is enabled.
Example	"E 2 3<CR><LF>" erases the flash sectors 2 and 3.

9.10 Blank check sector(s) <sector number> <end sector number>

Table 207. ISP Blank check sector command

Command	I
Input	**Start Sector Number:** **End Sector Number:** Should be greater than or equal to start sector number.
Return Code	CMD_SUCCESS | SECTOR_NOT_BLANK (followed by <Offset of the first non blank word location> <Contents of non blank word location>) | INVALID_SECTOR | PARAM_ERROR |
Description	This command is used to blank check one or more sectors of on-chip Flash memory. **Blank check on sector 0 always fails as first 64 bytes are re-mapped to flash boot block.**
Example	"I 2 3<CR><LF>" blank checks the flash sectors 2 and 3.

9.11 Read Part Identification number

Table 208. ISP Read Part Identification number command

Command	J
Input	None.
Return Code	CMD_SUCCESS followed by part identification number in ASCII (see Table 18–209).
Description	This command is used to read the part identification number.

Table 209. LPC2104/05/06 Part identification numbers

Device	ASCII/dec coding	Hex coding
LPC2104	4293984018	0xFFF0 FF12
LPC2105	4293984034	0xFFF0 FF22
LPC2106	4293984050	0xFFF0 FF32

9.12 Read Boot code version number

Table 210. ISP Read Boot code version number command

Command	K
Input	None
Return Code	CMD_SUCCESS followed by 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)>.
Description	This command is used to read the boot code version number.

9.13 Compare <address1> <address2> <no of bytes>

Table 211. ISP Compare command

Command	M
Input	**Address1 (DST):** Starting Flash or RAM address of data bytes to be compared. This address should be a word boundary. **Address2 (SRC):** Starting Flash or RAM address of data bytes to be compared. This address should be a word boundary. **Number of Bytes:** Number of bytes to be compared; should be a multiple of 4.
Return Code	CMD_SUCCESS | (Source and destination data are equal) COMPARE_ERROR | (Followed by the offset of first mismatch) COUNT_ERROR (Byte count is not a multiple of 4) | ADDR_ERROR | ADDR_NOT_MAPPED | PARAM_ERROR |
Description	This command is used to compare the memory contents at two locations. **Compare result may not be correct when source or destination address contains any of the first 64 bytes starting from address zero. First 64 bytes are re-mapped to flash boot sector**
Example	"M 8192 1073741824 4<CR><LF>" compares 4 bytes from the RAM address 0x4000 0000 to the 4 bytes from the flash address 0x2000.

9.14 ISP Return codes

Table 212. ISP Return codes Summary

Return Code	Mnemonic	Description
0	CMD_SUCCESS	Command is executed successfully. Sent by ISP handler only when command given by the host has been completely and successfully executed.
1	INVALID_COMMAND	Invalid command.
2	SRC_ADDR_ERROR	Source address is not on word boundary.
3	DST_ADDR_ERROR	Destination address is not on a correct boundary.
4	SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken in to consideration where applicable.
5	DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable.
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid or end sector number is greater than start sector number.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data not equal.
11	BUSY	Flash programming hardware interface is busy.

Table 212. ISP Return codes Summary

Return Code	Mnemonic	Description
12	PARAM_ERROR	Insufficient number of parameters or invalid parameter.
13	ADDR_ERROR	Address is not on word boundary.
14	ADDR_NOT_MAPPED	Address is not mapped in the memory map. Count value is taken in to consideration where applicable.
15	CMD_LOCKED	Command is locked.
16	INVALID_CODE	Unlock code is invalid.
17	INVALID_BAUD_RATE	Invalid baud rate setting.
18	INVALID_STOP_BIT	Invalid stop bit setting.
19	CODE_READ_PROTECTION_ ENABLED	Code read protection enabled.

10. IAP commands

For in application programming the IAP routine should be called with a word pointer in register r0 pointing to memory (RAM) containing command code and parameters. Result of the IAP command is returned in the result table pointed to by register r1. The user can reuse the command table for result by passing the same pointer in registers r0 and r1. The parameter table should be big enough to hold all the results in case if number of results are more than number of parameters. Parameter passing is illustrated in the Figure 18–61. The number of parameters and results vary according to the IAP command. The maximum number of parameters is 5, passed to the "Copy RAM to FLASH" command. The maximum number of results is 2, returned by the "Blankcheck sector(s)" command. The command handler sends the status code INVALID_COMMAND when an undefined command is received. The IAP routine resides at 0x7FFF FFF0 location and it is thumb code.

The IAP function could be called in the following way using C.

Define the IAP location entry point. Since the 0th bit of the IAP location is set there will be a change to Thumb instruction set when the program counter branches to this address.

```
#define IAP_LOCATION 0x7ffffff1
```

Define data structure or pointers to pass IAP command table and result table to the IAP function:

```
unsigned long command[5];
unsigned long result[3];
```

or

```
unsigned long * command;
unsigned long * result;
command=(unsigned long *) 0x……
result= (unsigned long *) 0x……
```

Define pointer to function type, which takes two parameters and returns void. Note the IAP returns the result with the base address of the table residing in R1.

```
typedef void (*IAP)(unsigned int [],unsigned int[]);
IAP iap_entry;
```

Setting function pointer:

```
iap_entry=(IAP) IAP_LOCATION;
```

Whenever you wish to call IAP you could use the following statement.

```
iap_entry (command, result);
```

The IAP call could be simplified further by using the symbol definition file feature supported by ARM Linker in ADS (ARM Developer Suite). You could also call the IAP routine using assembly code.

The following symbol definitions can be used to link IAP routine and user application:

```
#<SYMDEFS># ARM Linker, ADS1.2 [Build 826]: Last Updated: Wed May 08 16:12:23 2002
0x7fffff90 T rm_init_entry
0x7fffffa0 A rm_undef_handler
0x7fffffb0 A rm_prefetchabort_handler
0x7fffffc0 A rm_dataabort_handler
0x7fffffd0 A rm_irqhandler
0x7fffffe0 A rm_irqhandler2
0x7ffffff0 T iap_entry
```

As per the ARM specification (The ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05) up to 4 parameters can be passed in the r0, r1, r2 and r3 registers respectively. Additional parameters are passed on the stack. Up to 4 parameters can be returned in the r0, r1, r2 and r3 registers respectively. Additional parameters are returned indirectly via memory. Some of the IAP calls require more than 4 parameters. If the ARM suggested scheme is used for the parameter passing/returning then it might create problems due to difference in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such risk.

The flash memory is not accessible during a write or erase operation. IAP commands, which results in a flash write/erase operation, use 32 bytes of space in the top portion of the on-chip RAM for execution. The user program should not be use this space if IAP flash programming is permitted in the application.

Table 213. IAP command summary

IAP Command	Command Code	Described in
Prepare sector(s) for write operation	50_{10}	Table 18–214
Copy RAM to Flash	51_{10}	Table 18–215
Erase sector(s)	52_{10}	Table 18–216
Blank check sector(s)	53_{10}	Table 18–217
Read Part ID	54_{10}	Table 18–218
Read Boot code version	55_{10}	Table 18–219
Compare	56_{10}	Table 18–220

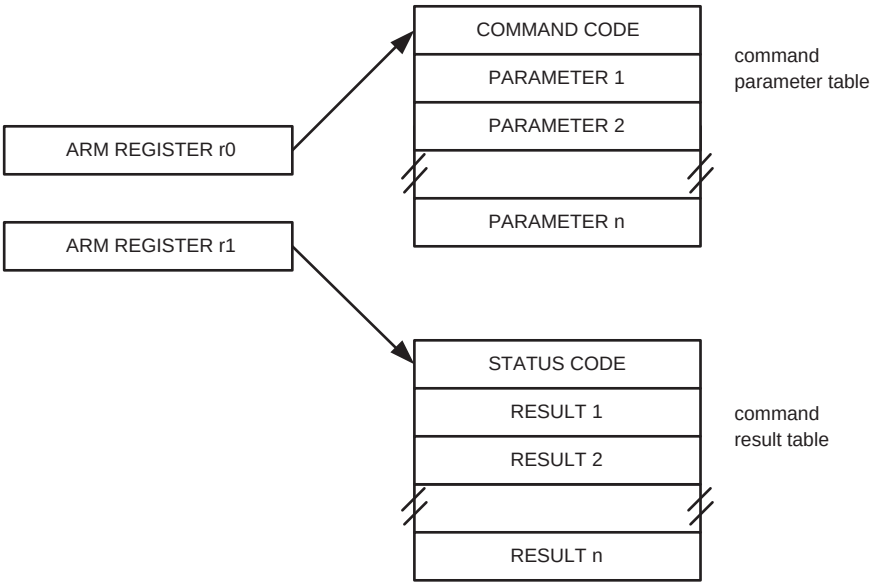


Fig 61. IAP parameter passing

10.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two step process.

Table 214. IAP Prepare sector(s) for write operation command

Command	Prepare sector(s) for write operation
Input	**Command code: 50** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR
Result	None
Description	This command must be executed before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot sector can not be prepared by this command. To prepare a single sector use the same "Start" and "End" sector numbers.

10.2 Copy RAM to Flash

Table 215. IAP Copy RAM to Flash command

Command	Copy RAM to Flash
Input	**Command code: 51** **Param0(DST):** Destination Flash address where data bytes are to be written. This address should be a 256 byte boundary. **Param1(SRC):** Source RAM address from which data bytes are to be read. This address should be a word boundary. **Param2:** Number of bytes to be written. Should be 256 | 512 | 1024 | 4096. **Param3:** System Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS | SRC_ADDR_ERROR (Address not a word boundary) | DST_ADDR_ERROR (Address not on correct boundary) | SRC_ADDR_NOT_MAPPED | DST_ADDR_NOT_MAPPED | COUNT_ERROR (Byte count is not 256 | 512 | 1024 | 4096) | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | BUSY |
Result	None
Description	This command is used to program the flash memory. The affected sectors should be prepared first by calling "Prepare Sector for Write Operation" command. The affected sectors are automatically protected again once the copy command is successfully executed. The boot sector can not be written by this command.

10.3 Erase sector(s)

Table 216. IAP Erase sector(s) command

Command	Erase Sector(s)
Input	**Command code: 52** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number). **Param2:** System Clock Frequency (CCLK) in kHz.
Return Code	CMD_SUCCESS | BUSY | SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION | INVALID_SECTOR
Result	None
Description	This command is used to erase a sector or multiple sectors of on-chip Flash memory. The boot sector can not be erased by this command. To erase a single sector use the same "Start" and "End" sector numbers.

10.4 Blank check sector(s)

Table 217. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	**Command code: 53** **Param0:** Start Sector Number **Param1:** End Sector Number (should be greater than or equal to start sector number).
Return Code	CMD_SUCCESS | BUSY | SECTOR_NOT_BLANK | INVALID_SECTOR
Result	**Result0:** Offset of the first non blank word location if the Status Code is SECTOR_NOT_BLANK. **Result1:** Contents of non blank word location.
Description	This command is used to blank check a sector or multiple sectors of on-chip Flash memory. To blank check a single sector use the same "Start" and "End" sector numbers.

10.5 Read Part Identification number

Table 218. IAP Read Part Identification command

Command	Read part identification number
Input	**Command code: 54** **Parameters:** None
Return Code	CMD_SUCCESS |
Result	**Result0:** Part Identification Number (see Table 18–209 “LPC2104/05/06 Part identification numbers” on page 238 for details)
Description	This command is used to read the part identification number.

10.6 Read Boot code version number

Table 219. IAP Read Boot code version number command

Command	Read boot code version number
Input	**Command code: 55** **Parameters:** None
Return Code	CMD_SUCCESS |
Result	**Result0:** 2 bytes of boot code version number in ASCII format. It is to be interpreted as <byte1(Major)>.<byte0(Minor)>
Description	This command is used to read the boot code version number.

10.7 Compare <address1> <address2> <no of bytes>

Table 220. IAP Compare command

Command	Compare
Input	**Command code: 56** **Param0(DST):** Starting Flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param1(SRC):** Starting Flash or RAM address of data bytes to be compared. This address should be a word boundary. **Param2:** Number of bytes to be compared; should be a multiple of 4.
Return Code	CMD_SUCCESS | COMPARE_ERROR | COUNT_ERROR (Byte count is not a multiple of 4) | ADDR_ERROR | ADDR_NOT_MAPPED
Result	**Result0:** Offset of the first mismatch if the Status Code is COMPARE_ERROR.
Description	This command is used to compare the memory contents at two locations. **The result may not be correct when the source or destination includes any of the first 64 bytes starting from address zero. The first 64 bytes can be re-mapped to RAM.**

10.8 IAP Status codes

Table 221. IAP Status codes Summary

Status Code	Mnemonic	Description
0	CMD_SUCCESS	Command is executed successfully.
1	INVALID_COMMAND	Invalid command.
2	SRC_ADDR_ERROR	Source address is not on a word boundary.
3	DST_ADDR_ERROR	Destination address is not on a correct boundary.
4	SRC_ADDR_NOT_MAPPED	Source address is not mapped in the memory map. Count value is taken in to consideration where applicable.
5	DST_ADDR_NOT_MAPPED	Destination address is not mapped in the memory map. Count value is taken in to consideration where applicable.
6	COUNT_ERROR	Byte count is not multiple of 4 or is not a permitted value.
7	INVALID_SECTOR	Sector number is invalid.
8	SECTOR_NOT_BLANK	Sector is not blank.
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	Command to prepare sector for write operation was not executed.
10	COMPARE_ERROR	Source and destination data is not same.
11	BUSY	Flash programming hardware interface is busy.

11. JTAG Flash programming interface

Debug tools can write parts of the flash image to the RAM and then execute the IAP call "Copy RAM to Flash" repeatedly with proper offset.

1. Features

- No target resources are required by the software debugger in order to start the debugging session.
- The software debugger talks via a JTAG (Joint Test Action Group) port directly to the core.
- Instructions are inserted directly in to the ARM7TDMI-S core.
- The ARM7TDMI-S core or the System state can be examined, saved, or changed depending on the type of instruction inserted.
- Instructions can be executed at a slow debug speed or at a fast system speed.

2. Applications

The EmbeddedICE logic provides on-chip debug support. The debugging of the target system requires a host computer running the debugger software and an EmbeddedICE protocol convertor. EmbeddedICE protocol convertor converts the Remote Debug Protocol commands to the JTAG data needed to access the ARM7TDMI-S core present on the target system.

3. Description

The ARM7TDMI-S Debug Architecture uses the existing JTAG[1] port as a method of accessing the core. The scan chains that are around the core for production test are reused in the debug state to capture information from the data bus and to insert new information into the core or the memory. There are two JTAG-style scan chains within the ARM7TDMI-S. A JTAG-style Test Access Port Controller controls the scan chains. In addition to the scan chains, the debug architecture uses EmbeddedICE logic which resides on chip with the ARM7TDMI-S core. The EmbeddedICE has its own scan chain that is used to insert watchpoints and breakpoints for the ARM7TDMI-S core. The EmbeddedICE logic consists of two real time watchpoint registers, together with a control and status register. One or both of the watchpoint registers can be programmed to halt the ARM7TDMI-S core. Execution is halted when a match occurs between the values programmed into the EmbeddedICE logic and the values currently appearing on the address bus, data bus and some control signals. Any bit can be masked so that its value does not affect the comparison. Either watchpoint register can be configured as a watchpoint (i.e. on a data access) or a break point (i.e. on an instruction fetch). The watchpoints and breakpoints can be combined such that:

- The conditions on both watchpoints must be satisfied before the ARM7TDMI core is stopped. The CHAIN functionality requires two consecutive conditions to be satisfied before the core is halted. An example of this would be to set the first breakpoint to

1. For more details refer to *IEEE Standard 1149.1 - 1990 Standard Test Access Port and Boundary Scan Architecture*.

trigger on an access to a peripheral and the second to trigger on the code segment that performs the task switching. Therefore when the breakpoints trigger the information regarding which task has switched out will be ready for examination.

- The watchpoints can be configured such that a range of addresses are enabled for the watchpoints to be active. The RANGE function allows the breakpoints to be combined such that a breakpoint is to occur if an access occurs in the bottom 256 bytes of memory but not in the bottom 32 bytes.

The ARM7TDMI-S core has a Debug Communication Channel function in-built. The debug communication channel allows a program running on the target to communicate with the host debugger or another separate host without stopping the program flow or even entering the debug state. The debug communication channel is accessed as a co-processor 14 by the program running on the ARM7TDMI-S core. The debug communication channel allows the JTAG port to be used for sending and receiving data without affecting the normal program flow. The debug communication channel data and control registers are mapped in to addresses in the EmbeddedICE logic.

4. Pin description

Table 222. EmbeddedICE pin description

Pin Name	Type	Description
TMS	Input	**Test Mode Select.** The TMS pin selects the next state in the TAP state machine.
TCK	Input	**Test Clock.** This allows shifting of the data in, on the TMS and TDI pins. It is a positive edge triggered clock with the TMS and TCK signals that define the internal state of the device. **Remark:** This clock must be slower than $\frac{1}{6}$ of the CPU clock (CCLK) for the JTAG interface to operate.
TDI	Input	**Test Data In.** This is the serial data input for the shift register.
TDO	Output	**Test Data Output.** This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal.
TRST	Input	**Test Reset.** The TRST pin can be used to reset the test logic within the EmbeddedICE logic.
DBGSEL	Input	**Debug Select.** When LOW at Reset, the P0.17 - P0.31 pins are configured for alternate use via the Pin Connect Block. When HIGH at Reset, the debug mode is entered. For functionality provided by DBGSEL, see Section 19–8
RTCK	Output	**Returned Test Clock.** Extra signal added to the JTAG port. Required for designs based on ARM7TDMI-S processor core. Multi-ICE (Development system from ARM) uses this signal to maintain synchronization with targets having slow or widely varying clock frequency. For details refer to "Multi-ICE System Design considerations Application Note 72 (ARM DAI 0072A)". Also used during entry into debug mode.

5. Reset state of multiplexed pins

On the LPC2104/05/06, the pins TMS, TCK, TDI, TDO, and TRST are multiplexed with P0.17 - P0.21 (primary debug interface) and P0.27 - P0.31 (secondary debug interface). To have them come up as a primary debug port, DBGSEL needs to be held HIGH during and after reset. Additionally, the RTCK pin needs to be HIGH when the reset is released.

The RTCK pin can be driven HIGH externally or allowed to float HIGH via its on-chip pull-up. To have them come up as GPIO pins, do not connect a bias resistor, and ensure that any external driver connected to Pin 26 (RTCK) is either driving high or is in high-impedance state during Reset. For more details, see Section 19–8.

6. Register description

The EmbeddedICE logic contains 16 registers as shown in Table 19–223 below. The ARM7TDMI-S debug architecture is described in detail in "ARM7TDMI-S (rev 4) Technical Reference Manual" (ARM DDI 0234A) published by ARM Limited.

Table 223. EmbeddedICE logic registers

Name	Width	Description	Address
Debug Control	6	Force debug state, disable interrupts	00000
Debug Status	5	Status of debug	00001
Debug Comms Control Register	32	Debug communication control register	00100
Debug Comms Data Register	32	Debug communication data register	00101
Watchpoint 0 Address Value	32	Holds watchpoint 0 address value	01000
Watchpoint 0 Address Mask	32	Holds watchpoint 0 address mask	01001
Watchpoint 0 Data Value	32	Holds watchpoint 0 data value	01010
Watchpoint 0 Data Mask	32	Holds watchpoint 0 data mask	01011
Watchpoint 0 Control Value	9	Holds watchpoint 0 control value	01100
Watchpoint 0 Control Mask	8	Holds watchpoint 0 control mask	01101
Watchpoint 1 Address Value	32	Holds watchpoint 1 address value	10000
Watchpoint 1 Address Mask	32	Holds watchpoint 1 address mask	10001
Watchpoint 1 Data Value	32	Holds watchpoint 1 data value	10010
Watchpoint 1 Data Mask	32	Holds watchpoint 1 data mask	10011
Watchpoint 1 Control Value	9	Holds watchpoint 1 control value	10100
Watchpoint 1 Control Mask	8	Holds watchpoint 1 control mask	10101

7. Block diagram

The block diagram of the debug environment is shown below in Figure 19–62.

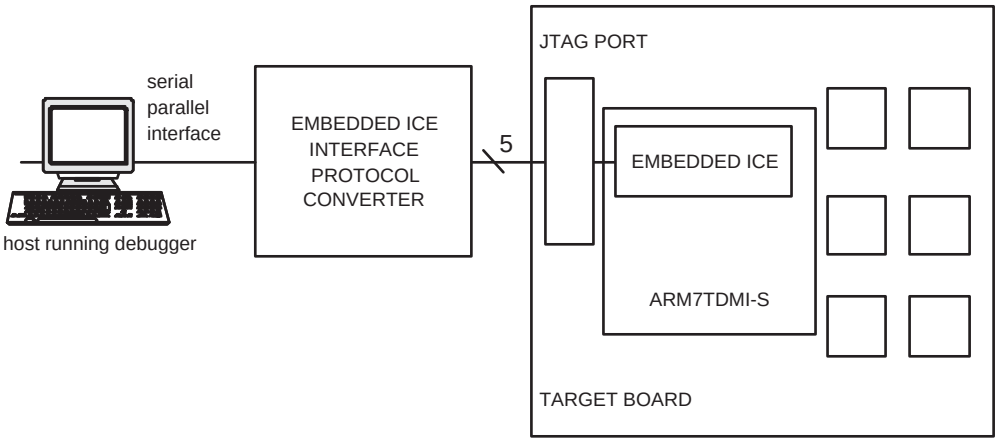


Fig 62. EmbeddedICE debug environment block diagram

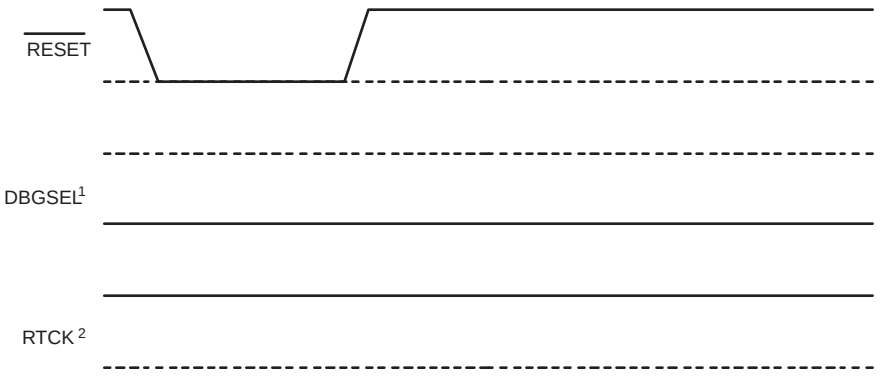
8. Debug mode

The Debug mode connects the JTAG pins to the embedded ICE for program debugging using an emulator or other development tool.

8.1 Enable Debug mode

The Debug mode can use the primary or secondary set of debug pins (see Section 19–8.2). The primary debug port is enabled through the use of the DBGSEL and RTCK pins. The secondary debug port must be configured using the pin connect block (see Section 7–2.2).

For normal (non-debug) operation, DBGSEL must be kept LOW at all times (see Figure 19–63)



(1) DBGSEL is tied or pulled LOW at all times. An internal pull-down will cause DBGSEL to be LOW if it is not pulled HIGH externally.

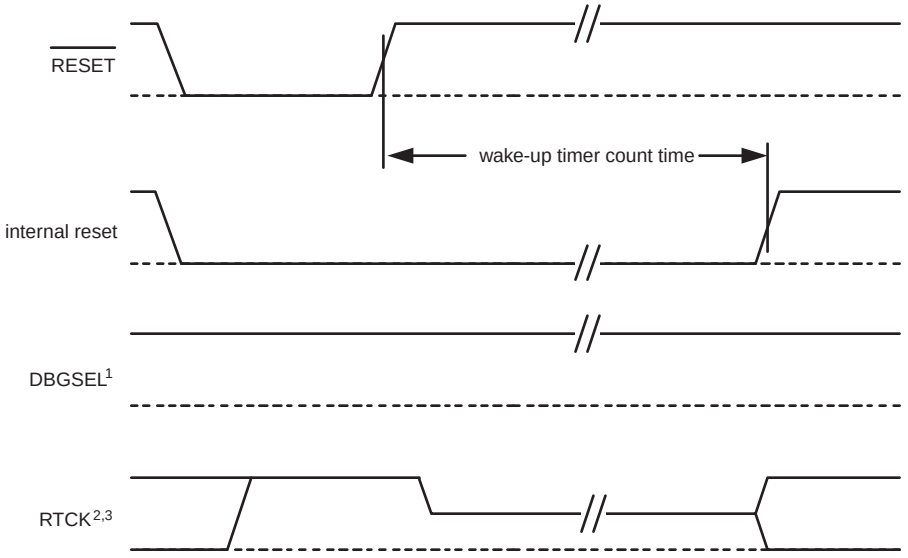
(2) RTCK is not connected in the application and is pulled up internally.

Fig 63. Waveforms for normal operations (not in debug mode)

Debugging using the primary JTAG port (P0.17 - P0.21)

To enable the debug mode, DBGSEL must be HIGH during and after the CPU is reset. For debugging with the primary JTAG pins, RTCK must be HIGH as the RESET pin is released (see Figure 19–64). RTCK may be driven HIGH externally or allowed to float HIGH via its on-chip pull-up. The RTCK output driver is disabled until the internal wake-up time has expired, allowing an interval between the release of the external reset and the release of the internal reset during which RTCK may be driven by an external signal if necessary.

This procedure establishes the P0.17 - P0.31 pins as the JTAG Test/Debug interface and enabled the trace port. Pin connect block settings have no effect on P0.17 - P0.31 pins if they are initialized as JTAG pins.



(1) DBGSEL must be HIGH.

(2) RTCK must be HIGH as $\overline{\text{RESET}}$ is released. An internal pull-up will cause RTCK to be HIGH if it is not pulled LOW externally.

(3) The RTCK output driver will be turned on when the internal chip reset is released by the wake-up timer.

Fig 64. Waveforms for Debug mode using the primary JTAG pins.

Debugging using the secondary JTAG port (P0.27 - P0.31)

The secondary debug port can be selected through the pin connect block by software. If the secondary debug port is selected, the ETM is not available because its pins are shared with the secondary JTAG port (see Table 6–54).

8.2 JTAG pin selection

The primary JTAG port can be selected for debugging only when DBGSEL and RTCK pins are HIGH at reset (see Figure 19–64). If at least one of the DBGSEL or RTCK lines is LOW at reset, JTAG will not be enabled and can not be used for later debugging. However, in this case software can assign a secondary JTAG port to pins P0.27 - P0.31.

Table 224. JTAG pin selection

DBGSEL (after RESET)	Latched RTCK value	JTAG primary pins	JTAG secondary pins	ETM available
HIGH	HIGH	yes	no	yes
LOW	HIGH	no	software configuration[1]	no
HIGH	LOW	no	software configuration[1]	no
LOW	LOW	no	software configuration[1]	no

[1] Start-up code residing in flash should configure port pins P0.27 to P0.31 for JTAG function by setting the appropriate bits in the PINSEL1 register.

1. Features

- Closely track the instructions that the ARM core is executing.
- One external trigger input
- 10 pin interface
- All registers are programmed through JTAG interface.
- Does not consume power when trace is not being used.
- THUMB instruction set support

2. Applications

As the microcontroller has significant amounts of on-chip memories, it is not possible to determine how the processor core is operating simply by observing the external pins. The ETM provides real-time trace capability for deeply embedded processor cores. It outputs information about processor execution to a trace port. 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.

3. 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 through a narrow trace port. An external Trace Port Analyzer captures the trace information under software debugger control. Trace port can broadcast the Instruction trace information. 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. Trace information generation can be controlled by selecting the trigger resource. Trigger resources include address 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.

3.1 ETM configuration

The following standard configuration is selected for the ETM macrocell.

Table 225. ETM configuration

Resource number/type	Small[1]
Pairs of address comparators	1
Data Comparators	0 (Data tracing is not supported)
Memory Map Decoders	4
Counters	1
Sequencer Present	No

Table 225. ETM configuration

Resource number/type	Small[1]
External Inputs	2
External Outputs	0
FIFOFULL Present	Yes (Not wired)
FIFO depth	10 bytes
Trace Packet Width	4/8

[1] For details refer to ARM documentation "Embedded Trace Macrocell Specification (ARM IHI 0014E)".

4. Pin description

Table 226. ETM Pin Description

Pin Name	Type	Description
TRACECLK	Output	**Trace Clock.** The trace clock signal provides the clock for the trace port. PIPESTAT[2:0], TRACESYNC, and TRACEPKT[3:0] signals are referenced to the rising edge of the trace clock. This clock is not generated by the ETM block. It is to be derived from the system clock. The clock should be balanced to provide sufficient hold time for the trace data signals. Half rate clocking mode is supported. Trace data signals should be shifted by a clock phase from TRACECLK. Refer to Figure 3.14 page 3.26 and figure 3.15 page 3.27 in "*ETM7 Technical Reference Manual" (ARM DDI 0158B)*, for example circuits that implements both half-rateclocking and shifting of the trace data with respect to the clock. For TRACECLK timings refer to section 5.2 on page 5-13 in "Embedded Trace Macrocell Specification" (ARM IHI 0014E).
PIPESTAT[2:0]	Output	**Pipe Line status.** The pipeline status signals provide a cycle-by-cycle indication of what is happening in the execution stage of the processor pipeline.
TRACESYNC	Output	**Trace synchronization.** The trace sync signal is used to indicate the first packet of a group of trace packets and is asserted HIGH only for the first packet of any branch address.
TRACEPKT[3:0]	Output	**Trace Packet.** The trace packet signals are used to output packaged address and data information related to the pipeline status. All packets are eight bits in length. A packet is output over two cycles. In the first cycle, Packet[3:0] is output and in the second cycle, Packet[7:4] is output.
EXTIN[0]	Input	**External Trigger Input**

5. Reset state of multiplexed pins

On the LPC2104/05/06, the ETM pin functions are multiplexed with P0.27 - P0.31. To use these pins as a Trace port, the DBGSEL pin must be HIGH, and Debug mode must be entered. For details, see Section 19–8.

6. Register description

The ETM contains 29 registers as shown in Table 20–227 below. They are described in detail in the ARM IHI 0014E document published by ARM Limited, which is available via the Internet.

Table 227. ETM Registers

Name	Description	Access	Register encoding
ETM Control	Controls the general operation of the ETM.	R/W	000 0000
ETM Configuration Code	Allows a debugger to read the number of each type of resource.	RO	000 0001
Trigger Event	Holds the controlling event.	WO	000 0010
Memory Map Decode Control	Eight-bit register, used to statically configure the memory map decoder.	WO	000 0011
ETM Status	Holds the pending overflow status bit.	RO	000 0100
System Configuration	Holds the configuration information using the SYSOPT bus.	RO	000 0101
Trace Enable Control 3	Holds the trace on/off addresses.	WO	000 0110
Trace Enable Control 2	Holds the address of the comparison.	WO	000 0111
Trace Enable Event	Holds the enabling event.	WO	000 1000
Trace Enable Control 1	Holds the include and exclude regions.	WO	000 1001
FIFOFULL Region	Holds the include and exclude regions.	WO	000 1010
FIFOFULL Level	Holds the level below which the FIFO is considered full.	WO	000 1011
ViewData event	Holds the enabling event.	WO	000 1100
ViewData Control 1	Holds the include/exclude regions.	WO	000 1101
ViewData Control 2	Holds the include/exclude regions.	WO	000 1110
ViewData Control 3	Holds the include/exclude regions.	WO	000 1111
Address Comparator 1 to 16	Holds the address of the comparison.	WO	001 xxxx
Address Access Type 1 to 16	Holds the type of access and the size.	WO	010 xxxx
Reserved	-	-	000 xxxx
Reserved	-	-	100 xxxx
Initial Counter Value 1 to 4	Holds the initial value of the counter.	WO	101 00xx
Counter Enable 1 to 4	Holds the counter clock enable control and event.	WO	101 01xx
Counter reload 1 to 4	Holds the counter reload event.	WO	101 10xx
Counter Value 1 to 4	Holds the current counter value.	RO	101 11xx
Sequencer State and Control	Holds the next state triggering events.	-	110 00xx
External Output 1 to 4	Holds the controlling events for each output.	WO	110 10xx
Reserved	-	-	110 11xx
Reserved	-	-	111 0xxx
Reserved	-	-	111 1xxx

7. Block diagram

The block diagram of the ETM debug environment is shown below in Figure 20–65.

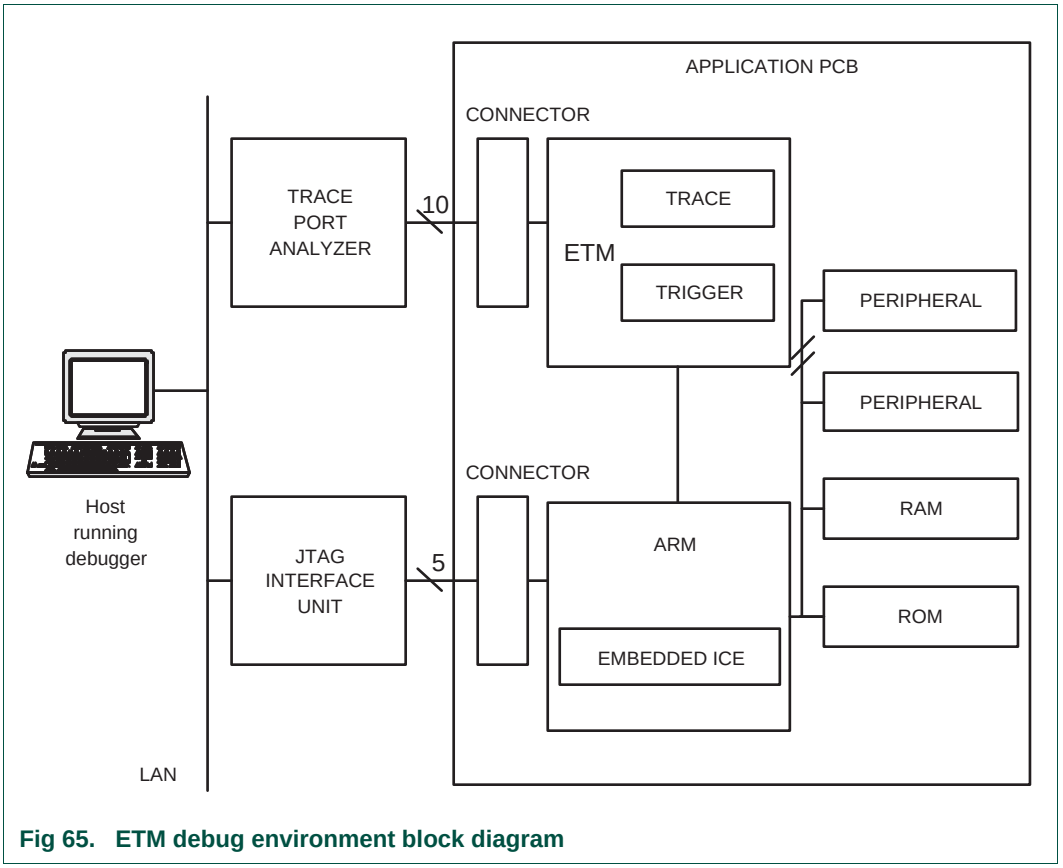


Fig 65. ETM debug environment block diagram

1. Features

- Allows user to establish a debug session to a currently running system without halting or resetting the system.
- Allows user time-critical interrupt code to continue executing while other user application code is being debugged.

2. Applications

Real time debugging.

3. Description

RealMonitor is a lightweight debug monitor that allows interrupts to be serviced while user debug their foreground application. It communicates with the host using the DCC (Debug Communications Channel), which is present in the EmbeddedICE logic. RealMonitor provides advantages over the traditional methods for debugging applications in ARM systems. The traditional methods include:

- Angel (a target-based debug monitor)
- Multi-ICE or other JTAG unit and EmbeddedICE logic (a hardware-based debug solution).

Although both of these methods provide robust debugging environments, neither is suitable as a lightweight real-time monitor.

Angel is designed to load and debug independent applications that can run in a variety of modes, and communicate with the debug host using a variety of connections (such as a serial port or ethernet). Angel is required to save and restore full processor context, and the occurrence of interrupts can be delayed as a result. Angel, as a fully functional target-based debugger, is therefore too heavyweight to perform as a real-time monitor.

Multi-ICE is a hardware debug solution that operates using the EmbeddedICE unit that is built into most ARM processors. To perform debug tasks such as accessing memory or the processor registers, Multi-ICE must place the core into a debug state. While the processor is in this state, which can be millions of cycles, normal program execution is suspended, and interrupts cannot be serviced.

RealMonitor combines features and mechanisms from both Angel and Multi-ICE to provide the services and functions that are required. In particular, it contains both the Multi-ICE communication mechanisms (the DCC using JTAG), and Angel-like support for processor context saving and restoring. RealMonitor is pre-programmed in the on-chip ROM memory (boot sector). When enabled It allows user to observe and debug while parts of application continue to run. Refer to Section 21–4 “How To Enable RealMonitor” on page 260 for details.

3.1 RealMonitor Components

As shown in Figure 21–66, RealMonitor is split in to two functional components:

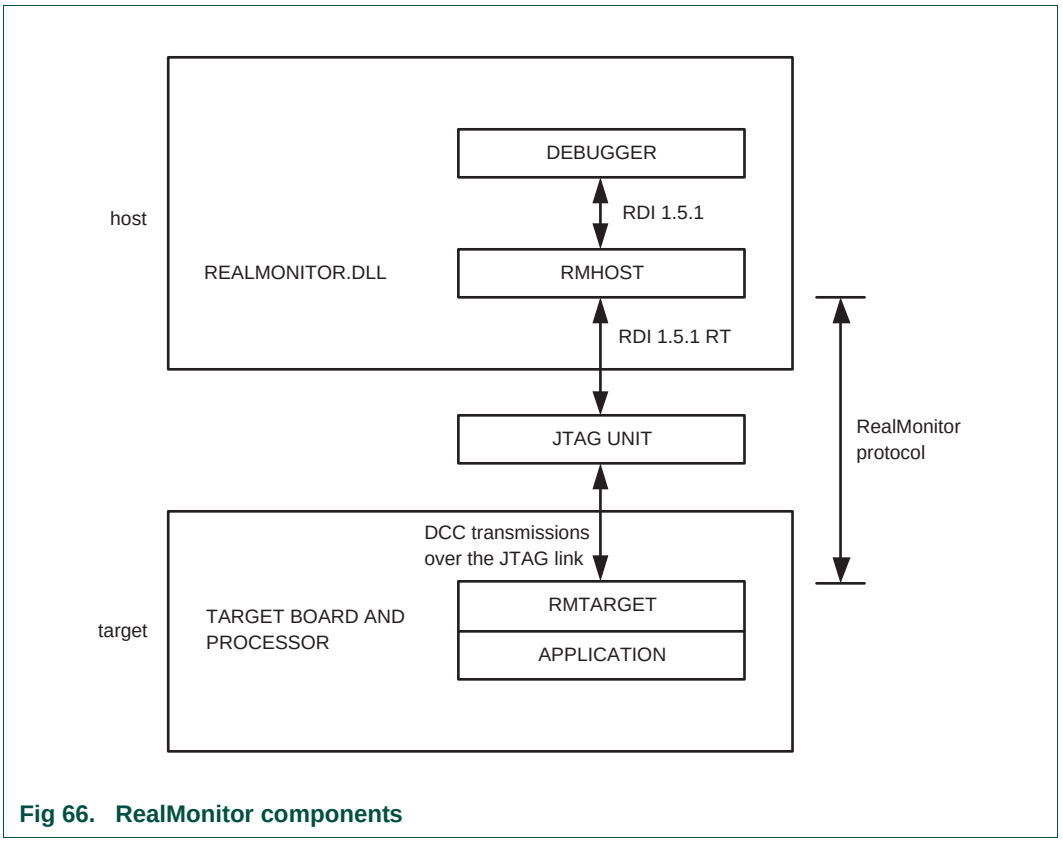


Fig 66. RealMonitor components

3.2 RMHost

This is located between a debugger and a JTAG unit. The RMHost controller, RealMonitor.dll, converts generic Remote Debug Interface (RDI) requests from the debugger into DCC-only RDI messages for the JTAG unit. For complete details on debugging a RealMonitor-integrated application from the host, see the ARM RMHost User Guide (ARM DUI 0137A).

3.3 RMTarget

This is pre-programmed in the on-chip ROM memory (boot sector), and runs on the target hardware. It uses the EmbeddedICE logic, and communicates with the host using the DCC. For more details on RMTarget functionality, see the RealMonitor Target Integration Guide (ARM DUI 0142A).

3.4 How RealMonitor works

In general terms, the RealMonitor operates as a state machine, as shown in Figure 21–67. RealMonitor switches between running and stopped states, in response to packets received by the host, or due to asynchronous events on the target. RMTarget supports the triggering of only one breakpoint, watchpoint, stop, or semihosting SWI at a time. There is no provision to allow nested events to be saved and restored. So, for

example, if user application has stopped at one breakpoint, and another breakpoint occurs in an IRQ handler, RealMonitor enters a panic state. No debugging can be performed after RealMonitor enters this state.

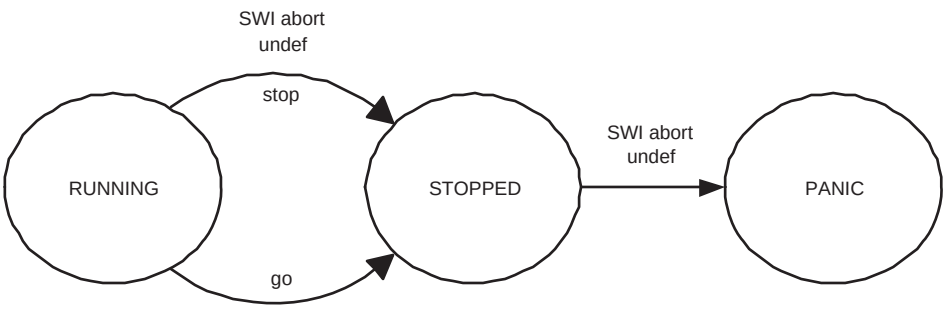


Fig 67. RealMonitor as a state machine

A debugger such as the ARM eXtended Debugger (AXD) or other RealMonitor aware debugger, that runs on a host computer, can connect to the target to send commands and receive data. This communication between host and target is illustrated in Figure 21–66.

The target component of RealMonitor, RMTarget, communicates with the host component, RMHost, using the Debug Communications Channel (DCC), which is a reliable link whose data is carried over the JTAG connection.

While user application is running, RMTarget typically uses IRQs generated by the DCC. This means that if user application also wants to use IRQs, it must pass any DCC-generated interrupts to RealMonitor.

To allow nonstop debugging, the EmbeddedICE-RT logic in the processor generates a Prefetch Abort exception when a breakpoint is reached, or a Data Abort exception when a watchpoint is hit. These exceptions are handled by the RealMonitor exception handlers that inform the user, by way of the debugger, of the event. This allows user application to continue running without stopping the processor. RealMonitor considers user application to consist of two parts:

- A foreground application running continuously, typically in User, System, or SVC mode
- A background application containing interrupt and exception handlers that are triggered by certain events in user system, including:
  - IRQs or FIQs
  - Data and Prefetch aborts caused by user foreground application. This indicates an error in the application being debugged. In both cases the host is notified and the user application is stopped.
  - Undef exception caused by the undefined instructions in user foreground application. This indicates an error in the application being debugged. RealMonitor stops the user application until a "Go" packet is received from the host.

When one of these exceptions occur that is not handled by user application, the following happens:

- RealMonitor enters a loop, polling the DCC. If the DCC read buffer is full, control is passed to rm_ReceiveData() (RealMonitor internal function). If the DCC write buffer is free, control is passed to rm_TransmitData() (RealMonitor internal function). If there is nothing else to do, the function returns to the caller. The ordering of the above comparisons gives reads from the DCC a higher priority than writes to the communications link.
- RealMonitor stops the foreground application. Both IRQs and FIQs continue to be serviced if they were enabled by the application at the time the foreground application was stopped.

4. How To Enable RealMonitor

The following steps must be performed to enable RealMonitor. A code example which implements all the steps can be found at the end of this section.

4.1 Adding stacks

User must ensure that stacks are set up within application for each of the processor modes used by RealMonitor. For each mode, RealMonitor requires a fixed number of words of stack space. User must therefore allow sufficient stack space for both RealMonitor and application.

RealMonitor has the following stack requirements:

Table 228. RealMonitor stack requirement

Processor mode	RealMonitor stack usage (bytes)
Undef	48
Prefetch Abort	16
Data Abort	16
IRQ	8

4.2 IRQ mode

A stack for this mode is always required. RealMonitor uses two words on entry to its interrupt handler. These are freed before nested interrupts are enabled.

4.3 Undef mode

A stack for this mode is always required. RealMonitor uses 12 words while processing an undefined instruction exception.

4.4 SVC mode

RealMonitor makes no use of this stack.

4.5 Prefetch Abort mode

RealMonitor uses four words on entry to its Prefetch abort interrupt handler.

4.6 Data Abort mode

RealMonitor uses four words on entry to its data abort interrupt handler.

4.7 User/System mode

RealMonitor makes no use of this stack.

4.8 FIQ mode

RealMonitor makes no use of this stack.

4.9 Handling exceptions

This section describes the importance of sharing exception handlers between RealMonitor and user application.

4.10 RealMonitor exception handling

To function properly, RealMonitor must be able to intercept certain interrupts and exceptions. Figure 21–68 illustrates how exceptions can be claimed by RealMonitor itself, or shared between RealMonitor and application. If user application requires the exception sharing, they must provide function (such as app_IRQDispatch ()). Depending on the nature of the exception, this handler can either:

- Pass control to the RealMonitor processing routine, such as rm_irqhandler2().
- Claim the exception for the application itself, such as app_IRQHandler ().

In a simple case where an application has no exception handlers of its own, the application can install the RealMonitor low-level exception handlers directly into the vector table of the processor. Although the IRQ handler must get the address of the Vectored Interrupt Controller. The easiest way to do this is to write a branch instruction (<address>) into the vector table, where the target of the branch is the start address of the relevant RealMonitor exception handler.

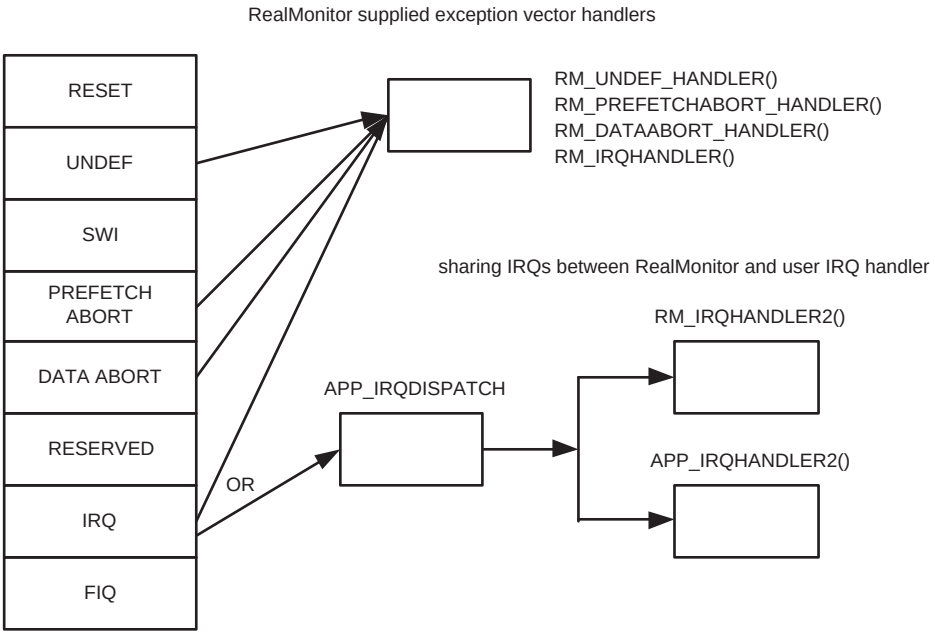


Fig 68. Exception handlers

4.11 RMTarget initialization

While the processor is in a privileged mode, and IRQs are disabled, user must include a line of code within the start-up sequence of application to call rm_init_entry().

4.12 Code Example

The following example shows how to setup stack, VIC, initialize RealMonitor and share non vectored interrupts:

```
IMPORT rm_init_entry
IMPORT rm_prefetchabort_handler
IMPORT rm_dataabort_handler
IMPORT rm_irqhandler2
IMPORT rm_undef_handler
IMPORT User_Entry ;Entry point of user application.
CODE32
ENTRY
;Define exception table. Instruct linker to place code at address 0x0000 0000

AREA exception_table, CODE

LDR pc, Reset_Address
LDR pc, Undefined_Address
LDR pc, SWI_Address
LDR pc, Prefetch_Address
LDR pc, Abort_Address
```

```
     NOP ; Insert User code valid signature here.
     LDR pc, [pc, #-0xFF0] ;Load IRQ vector from VIC
     LDR PC, FIQ_Address

Reset_Address         DCD __init            ;Reset Entry point
Undefined_Address     DCD rm_undef_handler ;Provided by RealMonitor
SWI_Address           DCD 0                 ;User can put address of SWI handler here
Prefetch_Address      DCD rm_prefetchabort_handler     ;Provided by RealMonitor
Abort_Address         DCD rm_dataabort_handler         ;Provided by RealMonitor
FIQ_Address           DCD 0                 ;User can put address of FIQ handler here

AREA init_code, CODE

ram_end EQU 0x4000xxxx ; Top of on-chip RAM.
__init
; /*********************************************************************
; * Set up the stack pointers for various processor modes. Stack grows
; * downwards.
; *********************************************************************/
     LDR  r2, =ram_end ;Get top of RAM
     MRS r0, CPSR ;Save current processor mode

     ; Initialize the Undef mode stack for RealMonitor use
     BIC  r1, r0, #0x1f
     ORR  r1, r1, #0x1b
     MSR  CPSR_c, r1
     ;Keep top 32 bytes for programming routines.
     ;Refer to On-chip Serial Bootloader chapter
     SUB  sp,r2,#0x1F

     ; Initialize the Abort mode stack for RealMonitor
     BIC  r1, r0, #0x1f
     ORR  r1, r1, #0x17
     MSR  CPSR_c, r1
     ;Keep 64 bytes for Undef mode stack
     SUB  sp,r2,#0x5F

     ; Initialize the IRQ mode stack for RealMonitor and User
     BIC  r1, r0, #0x1f
     ORR  r1, r1, #0x12
     MSR  CPSR_c, r1
     ;Keep 32 bytes for Abort mode stack
     SUB  sp,r2,#0x7F

     ; Return to the original mode.
     MSR  CPSR_c, r0

     ; Initialize the stack for user application
     ; Keep 256 bytes for IRQ mode stack
     SUB  sp,r2,#0x17F
```

```
; /*********************************************************************
; * Setup Vectored Interrupt controller. DCC Rx and Tx interrupts
; * generate Non Vectored IRQ request. rm_init_entry is aware
; * of the VIC and it enables the DBGCommRX and DBGCommTx interrupts.
; * Default vector address register is programmed with the address of
; * Non vectored app_irqDispatch mentioned in this example. User can setup
; * Vectored IRQs or FIQs here.
; *********************************************************************/

     VICBaseAddr          EQU 0xFFFFF000 ; VIC Base address
     VICDefVectAddrOffset  EQU 0x34

     LDR   r0, =VICBaseAddr
     LDR   r1, =app_irqDispatch
     STR   r1, [r0,#VICDefVectAddrOffset]

     BL    rm_init_entry   ;Initialize RealMonitor
     ;enable FIQ and IRQ in ARM Processor
     MRS   r1, CPSR        ; get the CPSR
     BIC   r1, r1, #0xC0   ; enable IRQs and FIQs
     MSR   CPSR_c, r1      ; update the CPSR
; /*********************************************************************
; * Get the address of the User entry point.
; *********************************************************************/
     LDR   lr, =User_Entry
     MOV   pc, lr
; /*********************************************************************
; * Non vectored irq handler (app_irqDispatch)
; *********************************************************************/

AREA app_irqDispatch, CODE
VICVectAddrOffset EQU 0x30
app_irqDispatch

     ;enable interrupt nesting
     STMFD sp!, {r12,r14}
     MRS   r12, spsr            ;Save SPSR in to r12
     MSR   cpsr_c,0x1F          ;Re-enable IRQ, go to system mode

;User should insert code here if non vectored Interrupt sharing is
;required. Each non vectored shared irq handler must return to
;the interrupted instruction by using the following code.
;    MSR   cpsr_c, #0x52                  ;Disable irq, move to IRQ mode
;    MSR   spsr, r12                      ;Restore SPSR from r12
;    STMFD sp!, {r0}
;    LDR   r0, =VICBaseAddr
;    STR   r1, [r0,#VICVectAddrOffset]    ;Acknowledge Non Vectored irq has finished
;    LDMFD sp!, {r12,r14,r0}              ;Restore registers
;    SUBS  pc, r14, #4                    ;Return to the interrupted instruction

     ;user interrupt did not happen so call rm_irqhandler2. This handler
```

```
;is not aware of the VIC interrupt priority hardware so trick
;rm_irqhandler2 to return here

STMFD sp!, {ip,pc}
LDR  pc, rm_irqhandler2
;rm_irqhandler2 returns here
MSR  cpsr_c, #0x52                  ;Disable irq, move to IRQ mode
MSR  spsr, r12                      ;Restore SPSR from r12
STMFD sp!, {r0}
LDR  r0, =VICBaseAddr
STR  r1, [r0,#VICVectAddrOffset]    ;Acknowledge Non Vectored irq has finished
LDMFD sp!, {r12,r14,r0}             ;Restore registers
SUBS  pc, r14, #4                   ;Return to the interrupted instruction

END
```

5. RealMonitor Build Options

RealMonitor was built with the following options:

RM_OPT_DATALOGGING=FALSE

This option enables or disables support for any target-to-host packets sent on a non RealMonitor (third-party) channel.

RM_OPT_STOPSTART=TRUE

This option enables or disables support for all stop and start debugging features.

RM_OPT_SOFTBREAKPOINT=TRUE

This option enables or disables support for software breakpoints.

RM_OPT_HARDBREAKPOINT=TRUE

Enabled for cores with EmbeddedICE-RT. This device uses ARM-7TDMI-S Rev 4 with EmbeddedICE-RT.

RM_OPT_HARDWATCHPOINT=TRUE

Enabled for cores with EmbeddedICE-RT. This device uses ARM-7TDMI-S Rev 4 with EmbeddedICE-RT.

RM_OPT_SEMIHOSTING=FALSE

This option enables or disables support for SWI semi-hosting. Semi-hosting provides code running on an ARM target use of facilities on a host computer that is running an ARM debugger. Examples of such facilities include the keyboard input, screen output, and disk I/O.

RM_OPT_SAVE_FIQ_REGISTERS=TRUE

This option determines whether the FIQ-mode registers are saved into the registers block when RealMonitor stops.

RM_OPT_READBYTES=TRUE

RM_OPT_WRITEBYTES=TRUE

RM_OPT_READHALFWORDS=TRUE

RM_OPT_WRITEHALFWORDS=TRUE

RM_OPT_READWORDS=TRUE

RM_OPT_WRITEWORDS=TRUE

Enables/Disables support for 8/16/32 bit read/write.

RM_OPT_EXECUTECODE=FALSE

Enables/Disables support for executing code from "execute code" buffer. The code must be downloaded first.

RM_OPT_GETPC=TRUE

This option enables or disables support for the RealMonitor GetPC packet. Useful in code profiling when real monitor is used in interrupt mode.

RM_EXECUTECODE_SIZE=NA

"execute code" buffer size. Also refer to RM_OPT_EXECUTECODE option.

RM_OPT_GATHER_STATISTICS=FALSE

This option enables or disables the code for gathering statistics about the internal operation of RealMonitor.

RM_DEBUG=FALSE

This option enables or disables additional debugging and error-checking code in RealMonitor.

RM_OPT_BUILDIDENTIFIER=FALSE

This option determines whether a build identifier is built into the capabilities table of RMTarget. Capabilities table is stored in ROM.

RM_OPT_SDM_INFO=FALSE

SDM gives additional information about application board and processor to debug tools.

RM_OPT_MEMORYMAP=FALSE

This option determines whether a memory map of the board is built into the target and made available through the capabilities table

RM_OPT_USE_INTERRUPTS=TRUE

This option specifies whether RMTarget is built for interrupt-driven mode or polled mode.

RM_FIFOSIZE=NA

This option specifies the size, in words, of the data logging FIFO buffer.

CHAIN_VECTORS=FALSE

This option allows RMTarget to support vector chaining through µHAL (ARM HW abstraction API).

1. Abbreviations

Table 229. Acronym list

Acronym	Description
ADC	Analog-to-Digital Converter
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
CISC	Complex Instruction Set Computer
FIFO	First In, First Out
GPIO	General Purpose Input/Output
I/O	Input/Output
JTAG	Joint Test Action Group
PLL	Phase-Locked Loop
PWM	Pulse Width Modulator
RISC	Reduced Instruction Set Computer
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSI	Synchronous Serial Interface
SSP	Synchronous Serial Port
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter

2. Legal information

2.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

2.2 Disclaimers

General — 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.

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 medical, military, aircraft, space or life support 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.

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.

2.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I^2C-bus — logo is a trademark of NXP B.V.

Notes

3. Tables

4. Figures

5. Contents

Chapter 1: Introductory information

Chapter 2: LPC2104/05/06 memory addressing

Chapter 3: LPC2104/05/06 system control block

Chapter 4: LPC2104/05/06 Memory Accelerator Module (MAM)

Chapter 5: LPC2104/05/06 Vectored Interrupt Controller (VIC)

Chapter 6: LPC2104/05/06 Pin configuration

Chapter 7: LPC2104/05/06 Pin connect block

Chapter 8: LPC2104/05/06 General Purpose Input/Output (GPIO)

Chapter 9: LPC2104/05/06 Universal Asynchronous Receiver/Transmitter (UART) 0

Chapter 10: LPC2104/05/06 Universal Asynchronous Receiver/Transmitter (UART) 1

Chapter 11: LPC2104/05/06 I^2C interface

Chapter 12: LPC2104/05/06 SPI

Chapter 13: LPC2104/05/06 SSP interface

Chapter 14: LPC2104/05/06 Timer0/1

Chapter 15: LPC2104/05/06 Pulse Width Modulator (PWM)

Chapter 16: LPC2104/05/06 Real Time Clock (RTC)

Chapter 17: LPC2104/05/06 WatchDog Timer (WDT)

Chapter 18: LPC2104/05/06 Flash memory and system programming

Chapter 19: LPC2104/05/06 EmbeddedICE logic

Chapter 20: LPC2104/05/06 Embedded Trace Macrocell (ETM)

Chapter 21: LPC2104/05/06 RealMonitor

Chapter 22: Supplementary information

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 8 April 2009
Document identifier: UM10275_2