UM10413

MPT612 User manual

Rev. 1 — 16 December 2011 User manual

Document information

Info	Content
Keywords	ARM, ARM7, embedded, 32-bit, MPPT, MPT612
Abstract	This document describes all aspects of the MPT612, an IC designed for applications using solar photovoltaic (PV) cells, or fuel cells.

Revision history

Rev	Date	Description
1	20111216	initial version

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 MPT612 is the first dedicated IC performing Maximum Power Point Tracking (MPPT) designed for applications using solar photovoltaic (PV) cells, or fuel cells. To simplify development and maximize system efficiency, the MPT612 is supported by:

- a patent-pending MPPT algorithm
- an application-specific software library
- easy-to-use application programming interfaces (APIs)

Dedicated hardware functions for PV panels, including voltage and current measurement and panel parameter configuration, simplify design and speed development.

MPT612 is based on a low-power ARM7TDMI-S RISC core operating up to 70 MHz achieving overall system efficiency ratings up to 98 %. It controls the external switching device through a signal derived from a patent-pending MPPT algorithm which delivers up to 99.5% Maximum Power Point Tracking (MPPT) efficiency. The solar PV DC source can be connected to the IC through appropriate voltage and current sensors. The IC dynamically extracts the maximum power from the PV panel without user intervention when enabled. The IC can be configured for boundary conditions set in software. There is up to 15 kB of flash memory available for application software.

In this user manual, solar PV terminology is primarily used as an example. However, the MPT612 is equally useful for fuel cells or any other DC source which has MPP characteristics.

2. Features

- ARM7TDMI-S 32-bit RISC core operating at up to 70 MHz
- 128-bit wide interface and accelerator enabling 70 MHz operation
- 10-bit ADC providing:
  - Conversion times as low as 2.44 μs per channel and dedicated result registers minimize interrupt overhead
  - Five analog inputs available for user-specific applications
- One 32-bit timer and external event counter with four capture and four compare channels
- One 16-bit timer and external event counter with four compare channels
- Low-power Real-Time Clock (RTC) with independent power supply and dedicated 32 kHz clock input
- Serial interfaces including:
  - Two UARTs (16C550)
  - Two Fast I^2C-buses (400 kbit/s)
  - SPI and SSP with buffering and variable data length capabilities
- Vectored interrupt controller with configurable priorities and vector addresses
- Up to 28, 5 V-tolerant fast general-purpose I/O pins
- Up to 13 edge- or level-sensitive external interrupt pins available

- Three levels of flash Code Read Protection (CRP)
- 70 MHz maximum clock available from programmable on-chip PLL with input frequencies between 10 MHz and 25 MHz and a settling time of 100 ms
- Integrated oscillator operates with an external crystal at between 1 MHz and 25 MHz
- Power-saving modes include:
  - Idle mode
  - Two Power-down modes; one with the RTC active and with the RTC deactivated
- Individual enabling/disabling of peripheral functions and peripheral clock scaling for additional power optimization
- Processor wake-up from Power-down and Deep power-down mode using an external interrupt or the RTC
- In-System/In-Application Programming (ISP/IAP) via on-chip bootloader software. Single flash sector or full chip erase in 100 ms and programming of 256 bytes in 1 ms

3. Applications

- Battery charge controller for solar PV power and fuel-cells. The use cases are
  - Battery charging for home appliances such as lighting, DC fans, DC TV, DC motors or any other DC appliance
  - Battery charging for public lighting and signaling, such as: LED street lighting, garden/driveway lighting, dusk-to-dawn lighting, railway signaling, traffic signaling, remote telecom terminals/towers
  - Battery charging for portable devices
- DC-to-DC converter per panel to provide improved efficiency
- Micro inverter per panel removes the need for one large system inverter

4. Device information

Table 1. MPT612 device information

Type number	Flash memory	RAM	ADC	Temperature range (°C)
MPT612FBD48	32 kB	8 kB	8 inputs	−40 to +85

5. Architectural overview

The MPT612 comprises:

- ARM7TDMI-S CPU with emulation support
- ARM7 Local Bus for interface to on-chip memory controllers
- AMBA Advanced High-performance Bus (AHB) to interface the interrupt controller
- ARM Peripheral Bus (APB, a compatible superset of ARM AMBA Advanced Peripheral Bus) for connecting on-chip peripheral functions

The MPT612 configures the ARM7TDMI-S processor core in little endian byte order.

AHB peripherals are allocated a 2 MB range of addresses at the top of the 4 GB ARM memory space. Each AHB peripheral is allocated a 16 kB address space within the AHB address space. A pin connect block controls on-chip peripheral connections to device pins (see Section 12.4 “Register description” on page 62) configured by software to specific application requirements for the use of peripheral functions and pins.

5.1 ARM7TDMI-S processor

The ARM7TDMI-S is a general purpose 32-bit processor core offering high performance and very low-power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles making the instruction set and decode mechanisms much simpler than micro programmed Complex Instruction Set Computers (CISC). This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small, cost-effective processor core.

Pipeline techniques are employed ensuring 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 read from memory.

The ARM7TDMI-S processor also employs a unique architectural strategy known as Thumb making it suitable for high-volume applications with memory restrictions, or applications where code density is an issue.

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

- the standard 32-bit ARM set
- the 16-bit Thumb set

The Thumb 16-bit instruction sets allow up to twice the density of standard ARM code while retaining most of the performance advantage of ARM over a traditional 16-bit processor using 16-bit registers, made possible using the ARM code 32-bit register set.

Thumb code provides up to 65 % of standard ARM code and 160 % of the performance of an equivalent ARM processor connected to a 16-bit memory system.

The particular flash implementation in the MPT612 also allows full speed execution in ARM mode. Programming performance-critical and short code sections in ARM mode is recommended. The impact on the overall code size is minimal but the speed can increase by 30 % over Thumb mode.

5.2 On-chip flash memory system

The MPT612 incorporates a 32 kB flash memory system. This memory can be used for both code and data storage. Various methods can be used to program flash memory, such as using:

- the built-in JTAG interface
- In Systems Programming (ISP)
- UART
- In Application Programming (IAP)

The application program can erase and/or program flash memory while the application is running using IAP, allowing greater flexibility, for example, data storage field firmware upgrades. The entire flash memory is available for user code as the bootloader resides in a separate memory.

The MPT612 flash memory provides a minimum of 100 000 erase/write cycles and 20 years of data-retention memory.

5.3 On-chip Static RAM (SRAM)

On-chip static RAM can be used for code and/or data storage. The SRAM can be accessed as 8-bit, 16-bit and 32-bit. The MPT612 provides 8 kB of static RAM.

6. Block diagram

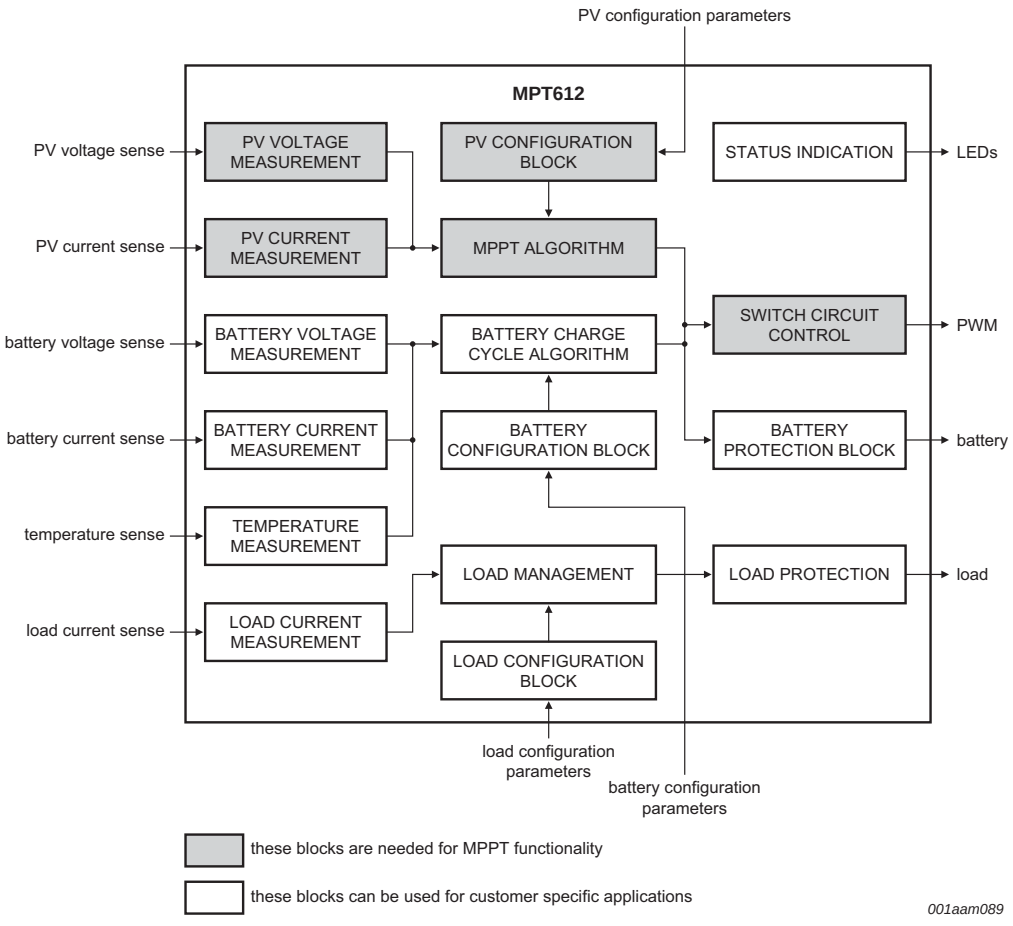


Fig 1. MPT612 block diagram

7. Memory addressing

7.1 Memory maps

The MPT612 incorporates several distinct memory regions, shown in Figure 2 to Figure 4. Figure 2 shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address remapping, which is described later in this section.

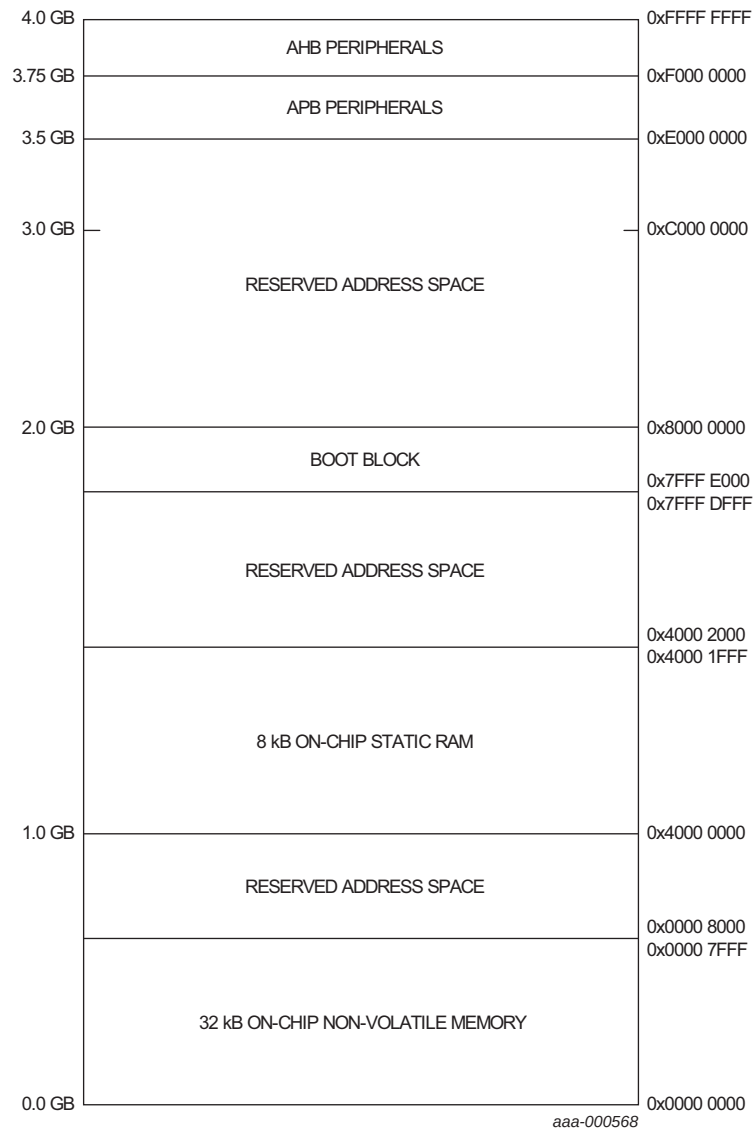
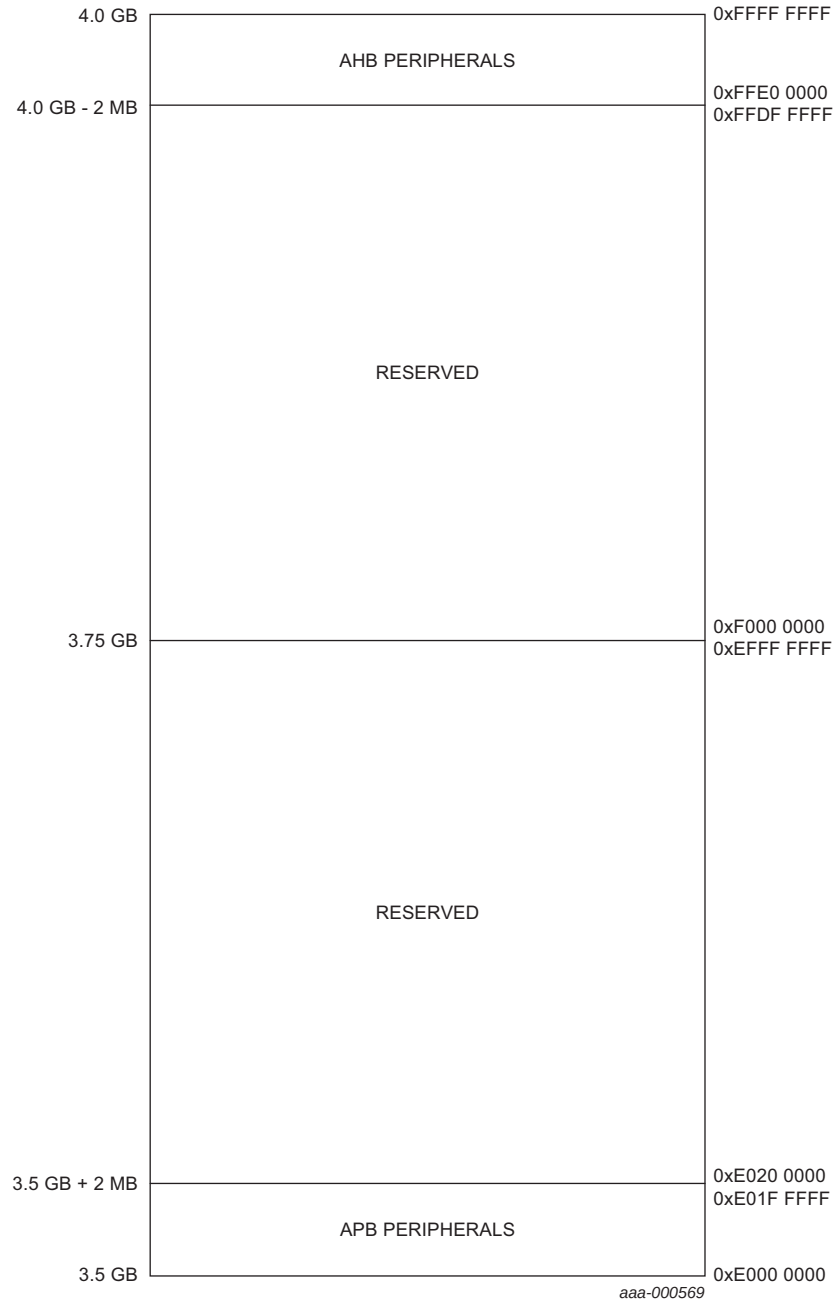


Fig 2. System memory map



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

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

Fig 3. Peripheral memory map

Figure 3, Figure 4, and Table 2 show different views of the peripheral address space. Both the AHB and APB peripheral areas are 2 MB spaces which are divided up into 128 peripherals. Each peripheral space is 16 kB in size, simplifying address decoding for each peripheral. All peripheral register addresses are word aligned (to 32-bit boundaries) regardless of their size, eliminating the need for byte lane mapping hardware to allow byte

(8-bit) or half-word (16-bit) accesses at smaller boundaries. This method requires all word and half-word registers to be accessed at once. For example, it is not possible to read or write the upper byte of a word register separately.

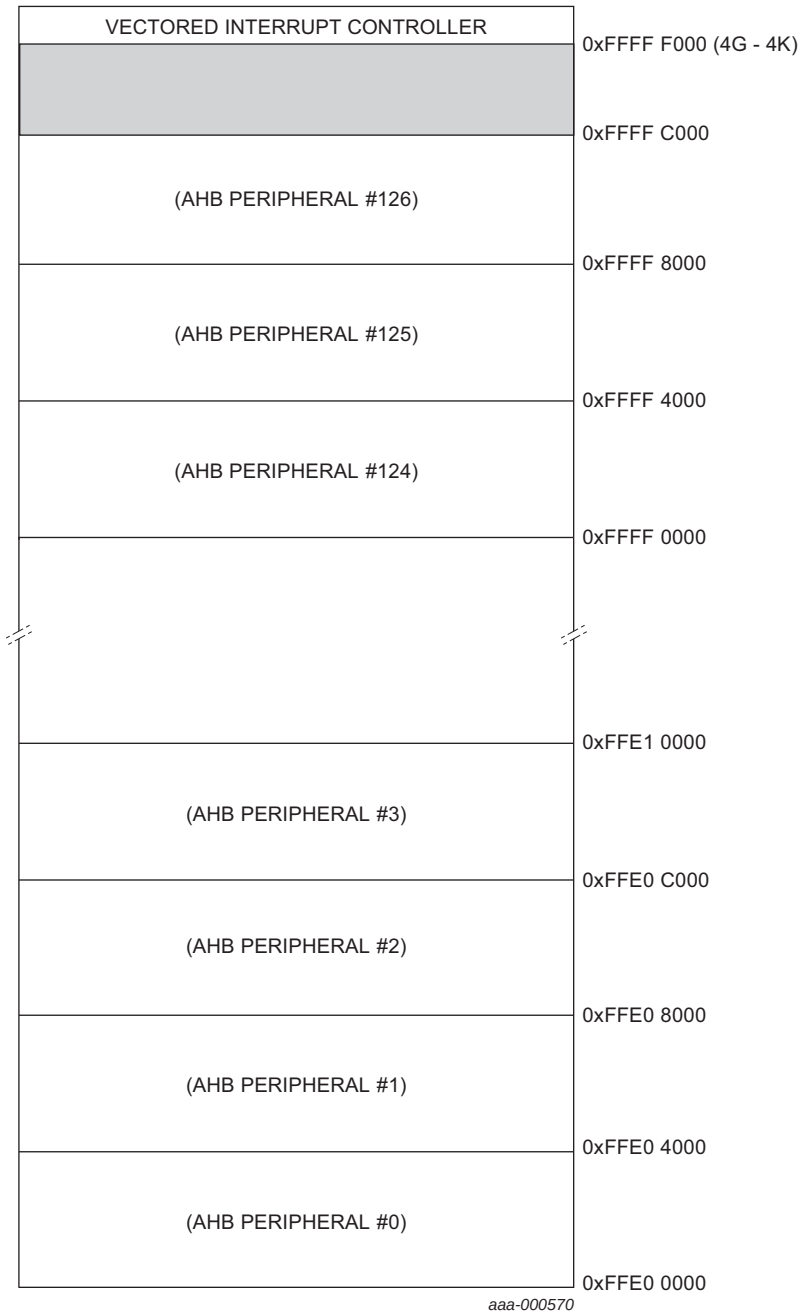


Fig 4. AHB peripheral map

Table 2. APB peripheries and base addresses

APB peripheral	Base address	Peripheral name
0	0xE000 0000	Watchdog timer
1	0xE000 4000	reserved
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	not used
6	0xE001 8000	not used
7	0xE001 C000	I^2C0
8	0xE002 0000	SPI0
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	pin connect block
12	0xE003 0000	not used
13	0xE003 4000	ADC
14 to 22	0xE003 8000 0xE005 8000	not used
23	0xE005 C000	I^2C1
24	0xE006 0000	not used
25	0xE006 4000	not used
26	0xE006 8000	SSP
27	0xE006 C000	not used
28	0xE007 0000	reserved
29	0xE007 4000	Timer 3
30 to 126	0xE007 8000 0xE01F 8000	not used
127	0xE01F C000	system control block

7.2 MPT612 memory remapping and boot block

7.2.1 Memory map concepts and operating modes

Basically, each memory area in the MPT612 has a "natural" location in the memory map, and is the address range for which code residing in that area is written. Most memory spaces remain permanently fixed in the same location, eliminating the need to design parts of the code to run in different address ranges.

Because of the ARM7 processor interrupt vector locations (at addresses 0x0000 0000 through 0x0000 001C, as shown in Table 3), a small portion of the boot block and SRAM spaces need remapping to allow alternative uses of interrupts in the different operating modes described in Table 4. Remapping of the interrupts is accomplished via the memory mapping control feature (Section 10.7 “Memory mapping control” on page 42).

Table 3. 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 **Remark:** Identified as reserved in ARM documentation, used by the bootloader as the valid user program key. Details described in Section 25.5.2 on page 218.
0x0000 0018	IRQ
0x0000 001C	FIQ

Table 4. MPT612 memory mapping modes

Mode	Activation	Usage
Boot loader mode	hardware activation by any reset	bootloader **always** executes after any reset. 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 bootloader when a valid user program signature is recognized in memory and bootloader operation is not forced. Interrupt vectors are not remapped and are found at the bottom of the flash memory.
User RAM mode	software activation by user program	activated by a user program as desired. Interrupt vectors are remapped to the bottom of the static RAM.

7.2.2 Memory remapping

In order to allow for compatibility with future derivatives, the entire boot block is mapped to the top of the on-chip memory space. This arrangement avoids larger or smaller flash modules having to change the location of the boot block (which requires changing the bootloader code) or changing the boot block interrupt vector mapping. Memory spaces other than the interrupt vectors remain in fixed locations. Figure 5 shows the on-chip memory mapping in the modes defined in Table 4.

The portion of memory remapped 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 remapped 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 in the SRAM, external memory, and boot block, must contain branches to the interrupt handlers, or to other instructions that establish the branch to the interrupt handlers.

There are three reasons this configuration was chosen:

- 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.
- Minimize the need for the SRAM and boot block vectors to deal with arbitrary boundaries in the middle of code space.

- To provide space to store constants, for jumping beyond the range of single word branch instructions.

Remapped memory areas, including the interrupt vectors, continue to appear in their original location in addition to the remapped address.

Details of remapping and examples can be found in Section 10.7 “Memory mapping control” on page 42.

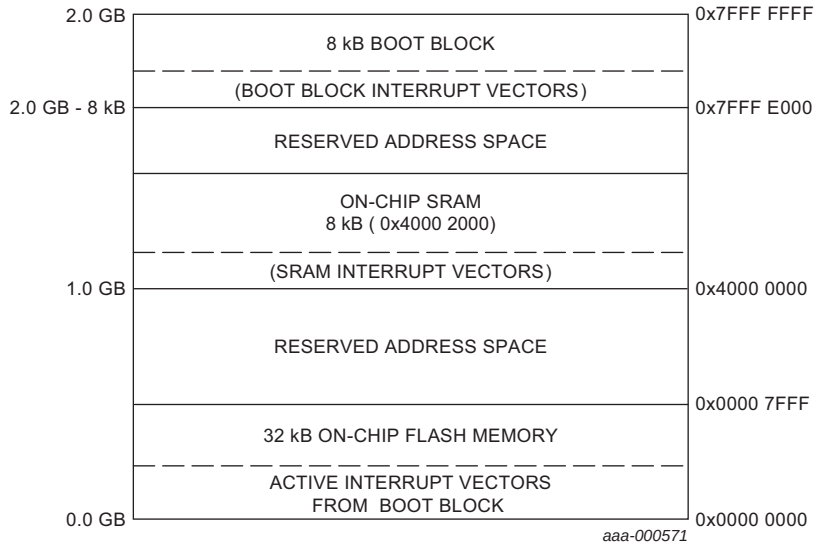


Fig 5. Map of lower memory showing remapped and remappable areas (MPT612 with 32 kB flash)

7.3 Prefetch abort and data abort exceptions

If an access is attempted for an address that is in a reserved or unassigned address region, the MPT612 generates the appropriate bus cycle abort exception. The regions are:

- Areas of the memory map that are not implemented for a specific ARM derivative. For the MPT612:
  - Address space between on-chip Non-Volatile Memory and on-chip SRAM, labeled "Reserved Address Space" in Figure 2. For this device, memory address range is from 0x0000 8000 to 0x3FFF FFFF.
  - Address space between on-chip static RAM and the boot block. Labeled "Reserved Address Space" in Figure 2. For this device, memory address range is from 0x4000 2000 to 0x7FFF DFFF.
  - Address space between 0x8000 0000 and 0xDFFF FFFF, labeled "Reserved Address Space".
  - Reserved regions of the AHB and APB spaces; see Figure 3.
- Unassigned AHB peripheral spaces; see Figure 4.
- Unassigned APB peripheral spaces; see Table 2.

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 distinguish only defined registers within the peripheral itself. For example, an access to address 0xE000 D000 (an undefined address within the UART0 space) can result in a register access defined at address 0xE000 C000. Details of such address aliasing within a peripheral space are not defined in the MPT612 documentation and are not a supported feature.

Remark: the ARM core stores the prefetch abort flag and the (meaningless) associated instruction in the pipeline, and processes the abort only if an attempt is made to execute the instruction fetched from the illegal address. This method prevents accidental aborts caused by prefetches occurring when code is executed very close to a memory boundary.

8. Memory Acceleration Module (MAM)

The MAM block in the MPT612 maximizes the performance of the ARM processor when it is running code in flash memory using a single flash bank.

8.1 Operation

Essentially, the Memory Accelerator Module (MAM) attempts to have the next ARM instruction that is needed in its latches in time to prevent CPU fetch stalls. The MPT612 uses one bank of flash memory, compared to the two banks used on predecessor devices. It includes three 128-bit buffers called the prefetch buffer, the branch trail buffer and the data buffer. The ARM is stalled while a fetch is initiated for the 128-bit line for an Instruction Fetch not satisfied by either the prefetch or branch trail buffer, or a prefetch not initiated for that line. If a prefetch is initiated but not yet completed, the ARM is stalled for a shorter time. Unless aborted by a data access, a prefetch is initiated when the flash has completed the previous access. The flash module latches the prefetched line, but the MAM does not capture the line in its prefetch buffer until the ARM core presents the address from which the prefetch is made. If the core presents a different address from the one from which the prefetch is made, the prefetched line is discarded.

The prefetch and branch trail buffers each include four 32-bit ARM instructions or eight 16-bit Thumb instructions. During sequential code execution, typically the prefetch buffer contains the current instruction and the entire flash line that contains it.

The MAM uses the LPROT[0] line to differentiate between instruction and data accesses. Code and data accesses use separate 128-bit buffers. Three of every four sequential 32-bit code or data accesses "hit" in the buffer without requiring a flash access (7 of 8 sequential 16-bit accesses, 15 of every 16 sequential byte accesses). The fourth (eighth, 16th) sequential data access must access flash, aborting any prefetch in progress. When a flash data access is concluded, any prefetch in progress is re-initiated.

Timing of flash read operations is programmable and is described later in this section.

There is no code fetch penalty for sequential instruction execution when the CPU clock period is greater than or equal to one fourth of the flash access time. The average amount of time spent processing program branches is relatively small (less than 25 %) and can be

minimized in ARM (rather than Thumb) code by using the conditional execution feature present in all ARM instructions. This conditional execution can often be used to avoid small forward branches that would otherwise be necessary.

Branches and other program flow changes cause a break in the sequential flow of instruction fetches previously described. The branch trail buffer captures the line to which such a non-sequential break occurs. If the same branch is taken again, the next instruction is taken from the branch trail buffer. When a branch outside the contents of the prefetch and branch trail buffer is taken, the branch trail buffer is loaded after several clock periods. Typically, there are no further instruction fetch delays until a new and different branch occurs.

8.2 MAM blocks

The MAM is divided into several functional blocks:

- A flash address latch and an incrementing function to form prefetch addresses
- A 128-bit prefetch buffer and an associated address latch and comparator
- A 128-bit branch trail buffer and an associated address latch and comparator
- A 128-bit data buffer and an associated address latch and comparator
- Control logic
- Wait logic

Figure 6 shows a simplified block diagram of the MAM 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.

8.2.1 Flash memory bank

There is one bank of flash memory on the MPT612 MAM.

Flash programming operations are handled as a separate function and not controlled by the MAM. A separate boot block in ROM contains flash programming algorithms that can be called by the application program, and a loader that can be run to allow serial programming of flash memory.

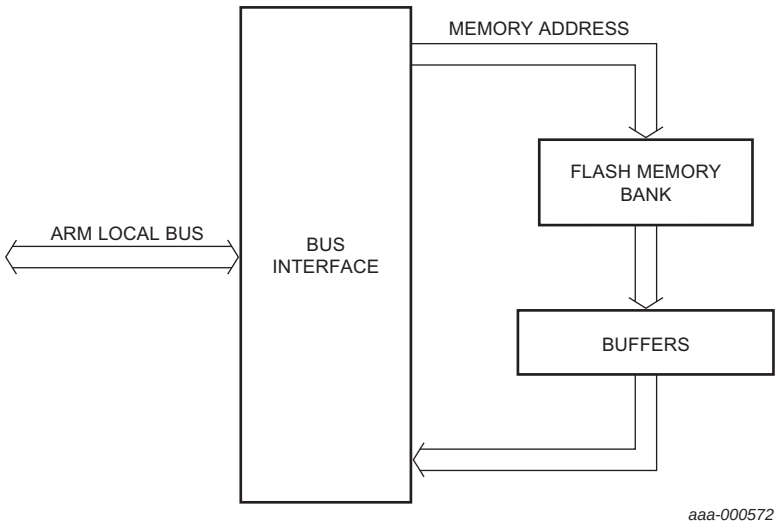


Fig 6. Simplified block diagram of the Memory Accelerator Module (MAM)

8.2.2 Instruction latches and data latches

The MAM treats code and data accesses separately. There is a 128-bit latch, a 15-bit address latch, and a 15-bit comparator associated with each buffer (prefetch, branch trail, and data). Each 128-bit latch holds 4 words (4 ARM instructions, or 8 Thumb instructions). Also associated with each buffer are 32 4:1 multiplexers that select the requested word from the 128-bit line.

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.

8.2.3 Flash programming issues

Since the flash memory does not allow access during programming and erase operations, the MAM must 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 can result in a watchdog time-out. You must ensure that an unwanted watchdog reset does not cause a system failure while programming or erasing the flash memory.

To preclude the possibility of stale data being read from the flash memory, the MPT612 MAM holding latches are automatically invalidated at the beginning of any flash programming or erase operation. Any subsequent read from a flash address initiates a new fetch after the flash operation has completed.

8.3 MAM operating modes

There are three MAM modes of operation defined, trading off performance for ease of predictability:

Mode 0: MAM off. All memory requests result in a flash read operation (see Table 5, note 2). No instruction prefetches are performed.

Mode 1: MAM partially enabled. If the data is present, sequential instruction accesses are fulfilled by the holding latches. Instruction prefetch is enabled. Non-sequential instruction accesses initiate flash read operations (see Table 5, note 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 5. 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] If available, the MAM actually uses latched data, but mimics the timing of a flash read operation. This method 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 6. 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] If available, the MAM actually uses latched data, but it mimics the timing of a flash read operation. This method 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.

8.4 MAM configuration

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

8.5 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 MAM registers

Name	Description	Access	Reset value[1]	Address
MAMCR	MAM control register. Determines MAM functional mode: to what extent the MAM performance enhancements are enabled; see Table 8.	R/W	0x0	0xE01F C000
MAMTIM	MAM timing control. Determines 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.

8.6 MAM Control register (MAMCR - 0xE01F C000)

Two configuration bits select the three MAM operating modes, as shown in Table 8. 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 8. 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 application	
7:2	-	-	reserved; user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

8.7 MAM Timing register (MAMTIM - 0xE01F C004)

The MAM Timing register determines how many CCLK cycles are used to access the flash memory. This method 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 removes the MAM from timing calculations. In this case, the MAM mode can be selected to optimize power usage.

Table 9. 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	
		Remark: these bits set duration of MAM flash fetch operations as listed. Improper setting of values can result in incorrect operation of the device.		
7:3	-	-	reserved; user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

8.8 MAM usage notes

When changing MAM timing, the MAM is turned off by writing a zero to MAMCR. A new value can then be written to MAMTIM. Finally, the MAM can 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. A suggested flash access time for a system clock between 20 MHz and 40 MHz is 2 CCLKs, while systems with a system clock faster than 40 MHz, 3 CCLKs are proposed. System clocks of 60 MHz and above require 4CCLKs.

Table 10. 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

9. Vectored Interrupt Controller (VIC)

9.1 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

9.2 Description

The Vectored Interrupt Controller (VIC) takes 32 interrupt request inputs and assigns them to 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 the FIQ service routine then deals with that device. If the FIQ class is assigned several requests, 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 where slot 0 has the highest priority and slot 15 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 a vectored IRQ is requesting, the VIC provides the address of the highest-priority requesting IRQ service routine, otherwise it provides the address of a default routine 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 read/write are not supported.

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

9.3 Register description

The VIC implements the registers shown in Table 11. More detailed descriptions follow.

Table 11. VIC register map

Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ status register. Reads out status of interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ status requests. Reads out status of interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	raw interrupt status register. Reads out status of the 32 interrupt requests/software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	interrupt select register. Classifies each of the 32 interrupt requests contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	interrupt enable 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. Allows software to clear one or more bits in the interrupt enable register.	WO	0	0xFFFF F014
VICSoftInt	software interrupt register. Contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018

Table 11. VIC register map *…continued*

Name	Description	Access	Reset value[1]	Address
VICSoftIntClear	software interrupt clear register. Allows software to clear one or more bits in the software interrupt register.	WO	0	0xFFFF F01C
VICProtection	protection enable 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
VICDefVectAddr	default vector address register. Holds address of Interrupt Service Routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034
VICVectAddr0	vector address 0 register. Vector address registers 0 to 15 hold addresses of 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 to 15 each control one of the 16 vectored IRQ slots. Slot 0 has 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

Table 11. VIC register map *...continued*

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

9.4 VIC registers

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

9.4.1 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 12. Software interrupt register (VICSoftInt - address 0xFFFF F018) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
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	-	-	-	-	I2C1	AD0	-	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	SSP/SPI1	SPI0	I2C0	-
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	reserved	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 13. Software interrupt register (VICSoftInt - address 0xFFFF F018) bit description

Bit	Symbol	Value	Description	Reset value
31:0	see Table 12	0	do not force interrupt request with this bit number; writing logic 0s to bits in VICSoftInt has no effect (see VICSoftIntClear); see Section 9.4.2.	0
		1	force interrupt request with this bit number	

9.4.2 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 read it first.

Table 14. Software interrupt clear register (VICSoftIntClear - address 0xFFFF F01C) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	I2C1	AD0	-	EINT2
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP/SPI1	SPI0	I2C0	-
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	reserved	ARMCore1	ARMCore0	-	WDT
Access	WO	WO	WO	WO	WO	WO	WO	WO

Table 15. Software interrupt clear register (VICSoftIntClear - address 0xFFFF F01C) bit description

Bit	Symbol	Value	Description	Reset value
31:0	see Table 14	0	writing logic 0 leaves corresponding bit in VICSoftInt unchanged	0
		1	writing logic 1 clears corresponding bit in software interrupt register, releasing forced request	

9.4.3 Raw interrupt status register (VICRawIntr - 0xFFFF F008)

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

Table 16. Raw interrupt status register (VICRawIntr - address 0xFFFF F008) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	I2C1	AD0	-	EINT2
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP/SPI1	SPI0	I2C0	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	reserved	ARMCore1	ARMCore0	-	WDT
Access	RO	RO	RO	RO	RO	RO	RO	RO

Table 17. Raw interrupt status register (VICRawIntr - address 0xFFFF F008) bit description

Bit	Symbol	Value	Description	Reset value
31:0	see Table 16	0	does not assert hardware or software interrupt request with this bit number	0
		1	asserts hardware or software interrupt request with this bit number	

9.4.4 Interrupt enable register (VICIntEnable - 0xFFFF F010)

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

Table 18. Interrupt enable register (VICIntEnable - address 0xFFFF F010) bit allocation

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
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	-	-	-	-	I2C1	AD0	-	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	SSP/SPI1	SPI0	I2C0	-
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	reserved	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 19. Interrupt enable register (VICIntEnable - address 0xFFFF F010) bit description

Bit	Symbol	Description	Reset value
31:0	see Table 18	when this register is read, 1s indicate interrupt requests or software interrupts enabled to contribute to FIQ or IRQ. when this register is written, 1s enable interrupt requests or software interrupts to contribute to FIQ or IRQ, 0s have no effect. To disable interrupts, see Section 9.4.5 and Table 21.	0

9.4.5 Interrupt enable clear register (VICIntEnClear - 0xFFFF F014)

This register is write only. It allows software to clear one or more bits in the interrupt enable register without having to read it first; see Section 9.4.4).

Table 20. Software interrupt clear register (VICIntEnClear - address 0xFFFF F014) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	I2C1	AD0	-	EINT2
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP/SPI1	SPI0	I2C0	-
Access	WO	WO	WO	WO	WO	WO	WO	WO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	reserved	ARMCore1	ARMCore0	-	WDT
Access	WO	WO	WO	WO	WO	WO	WO	WO

Table 21. Software interrupt clear register (VICIntEnClear - address 0xFFFF F014) bit description

Bit	Symbol	Value	Description	Reset value
31:0	see Table 20	0	writing logic 0 leaves corresponding bit in VICIntEnable unchanged	0
		1	writing logic 1 clears corresponding bit in interrupt enable register, disabling interrupts for this request	

9.4.6 Interrupt select register (VICIntSelect - 0xFFFF F00C)

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

Table 22. Interrupt select register (VICIntSelect - address 0xFFFF F00C) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
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	-	-	-	-	I2C1	AD0	-	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	SSP/SPI1	SPI0	I2C0	-
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	reserved	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 23. Interrupt select register (VICIntSelect - address 0xFFFF F00C) bit description

Bit	Symbol	Value	Description	Reset value
31:0	see Table 22	0	assigns interrupt request with this bit number to IRQ category	0
		1	assigns interrupt request with this bit number to FIQ category	

9.4.7 IRQ Status register (VICIRQStatus - 0xFFFF F000)

This register is read only. It 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 24. IRQ Status register (VICIRQStatus - address 0xFFFF F000) bit allocation

Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	I2C1	AD0	-	EINT2
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP/SPI1	SPI0	I2C0	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	reserved	ARMCore1	ARMCore0	-	WDT
Access	RO	RO	RO	RO	RO	RO	RO	RO

Table 25. IRQ Status register (VICIRQStatus - address 0xFFFF F000) bit description

Bit	Symbol	Description	Reset value
31:0	see Table 24	a bit read as logic 1 indicates a corresponding interrupt request being enabled, classified as IRQ, and asserted	0

9.4.8 FIQ Status register (VICFIQStatus - 0xFFFF F004)

This register is read only. It 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 26. FIQ Status register (VICFIQStatus - address 0xFFFF F004) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	TIMER3	reserved	-	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	-	-	I2C1	AD0	-	EINT2
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SSP/SPI1	SPI0	I2C0	-0
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	reserved	ARMCore1	ARMCore0	-	WDT
Access	RO	RO	RO	RO	RO	RO	RO	RO

Table 27. FIQ Status register (VICFIQStatus - address 0xFFFF F004) bit description

Bit	Symbol	Description	Reset value
31:0	see Table 26	a bit read as logic 1 indicates a corresponding interrupt request being enabled, classified as FIQ, and asserted	0

9.4.9 Vector control registers 0 to 15 (VICVectCntl0-15 0xFFFF F200 to 23C)

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

Table 28. Vector control registers 0 to 15 (VICVectCntl0 to 15 - 0xFFFF F200 to 23C) bit description

Bit	Symbol	Description	Reset value
4:0	int_request/ sw_int_assig	the number of interrupt requests or software interrupts assigned to this vectored IRQ slot. Software must not assign the same interrupt number to more than one enabled vectored IRQ slot, otherwise a lower numbered slot is used when interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
5	IRQslot_en	if logic 1, enables vectored IRQ slot 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

9.4.10 Vector address registers 0 to 15 (VICVectAddr0 to 15 0xFFFF F100 to 13C)

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

Table 29. Vector address registers 0 to 15 (VICVectAddr0 to 15 - addresses 0xFFFF F100 to 13C) bit description

Bit	Symbol	Description	Reset value
31:0	IRQ_vector	if an interrupt request or software interrupt is enabled, classified as IRQ, asserted and assigned to an enabled vectored IRQ slot, the value from this register is used for the highest priority slot, and is provided when IRQ service routine reads vector address register -VICVectAddr; see Section 9.4.10.	0x0000 0000

9.4.11 Default vector address register (VICDefVectAddr - 0xFFFF F034)

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

Table 30. Default vector address register (VICDefVectAddr - address 0xFFFF F034) bit description

Bit	Symbol	Description	Reset value
31:0	IRQ_vector	if an IRQ service routine reads the vector address register (VICVectAddr), and no IRQ slot responds as described above, this address is returned	0x0000 0000

9.4.12 Vector address register (VICVectAddr - 0xFFFF F030)

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

Table 31. Vector address register (VICVectAddr - address 0xFFFF F030) bit description

Bit	Symbol	Description	Reset value
31:0	IRQ_vector	if an interrupt request or software interrupt assigned to a vectored IRQ slot is enabled, classified as IRQ and asserted, reading this register returns the address in this register for the highest priority (lowest-numbered) 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. Instead, write to this register near the end of an ISR to update the priority hardware.	0x0000 0000

9.4.13 Protection enable register (VICProtection - 0xFFFF F020)

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

Table 32. Protection enable register (VICProtection - address 0xFFFF F020) bit description

Bit	Symbol	Value	Description	Reset value
0	VIC_access	0	VIC registers can be accessed in User or Privileged mode	0
		1	VIC registers can only be accessed in Privileged mode	
31:1	-		reserved; user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

9.5 Interrupt sources

Table 33 lists the interrupt sources for each peripheral function. Each peripheral device has one interrupt line connected to the Vectored Interrupt Controller, but can have several internal interrupt flags. Individual interrupt flags can represent more than one interrupt source.

Table 33. Connection of interrupt sources to the Vectored Interrupt Controller (VIC)

Block	Flag(s)	VIC Channel # and Hex mask	
WDT	Watchdog Interrupt (WDINT)	0	0x0000 0001
-	reserved for software interrupts only	1	0x0000 0002
ARM Core	EmbeddedICE, DbgCommRx	2	0x0000 0004
ARM Core	EmbeddedICE, DbgCommTX	3	0x0000 0008
-	reserved for internal use; do not modify	4	0x0000 0010
TIMER1	Match 0 to 3 (MR0, MR1, MR2, MR3) Capture 0 to 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)	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)	7	0x0000 0080
-	reserved	8	0x0000 0100
I^2C0	SI (state change)	9	0x0000 0200
SPI0	SPI0 Interrupt Flag (SPI0F) Mode Fault (MODF)	10	0x0000 0400
SPI1 (SSP)	Tx FIFO at least half empty (TXRIS) Rx FIFO at least half full (RXRIS) Receive Timeout condition (RTRIS) Receive overrun (RORRIS)	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
	reserved	17	0x0002 0000
ADC	A/D Converter 0 end of conversion	18	0x0004 0000
I^2C1	SI (state change)	19	0x0008 0000
-	reserved	20-25	0x0010 0000 0x0200 0000
-	reserved for internal use; do not modify	26	0x0400 0000
TIMER3	Match 0 - 3 (MR0, MR1, MR2, MR3)	27	0x0800 0000

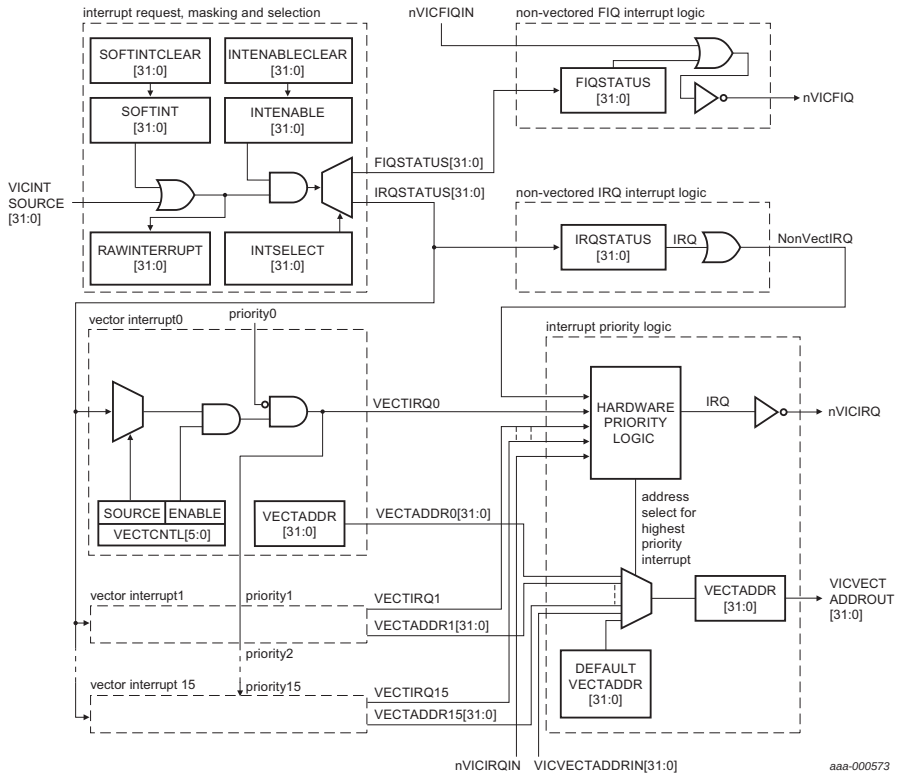


Fig 7. Block diagram of the Vectored Interrupt Controller (VIC)

9.6 Spurious interrupts

Spurious interrupts are possible in the ARM7TDMI based ICs such as the MPT612 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 can be generated.

Real-life applications can 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 is not able to identify clearly the interrupt that generated the interrupt request, and as a result the VIC returns the default interrupt VicDefVectAddr (0xFFFF F034).

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

- Application code must be set up in a way to prevent the spurious interrupts from occurring. Simple guarding of changes to the VIC cannot be enough since, for example, glitches on level-sensitive interrupts can also cause spurious interrupts.
- Correctly set up and test the VIC default handler.

9.6.1 Details and case studies on spurious interrupts

This chapter contains details that can be obtained from the official ARM website, FAQ section.

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

Applies to: ARM7TDMI

If an interrupt received by the core during execution of an instruction disables interrupts, the ARM7 family still takes the interrupt (IRQ or FIQ).

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 is as follows:

1. The IRQ interrupt is latched.
2. The MSR cpsr, r0 executes to completion setting both bit I and bit F in the CPSR.
3. The IRQ interrupt is taken because the core was committed to taking the interrupt exception before bit I was set in the CPSR.
4. The CPSR (with bit I and bit F 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 been taken while bit I in SPSR is set. In the example above, bit F is also set in both CPSR and SPSR. This means that FIQs are disabled upon entry to the IRQ service routine until explicitly re-enabled. The IRQ return sequence does not automatically re-enable FIQs.

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 core processes the IRQ after completing the MSR instruction which disables IRQs, and does not normally cause a problem, as an interrupt arriving one cycle earlier is 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 now has bit I and bit F set, and therefore execution continues with all interrupts disabled. However, problems can be caused 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 have been disabled before the routine being called. The routine exploits this restriction to determine how it was called (by examining bit I of SPSR), and returns using the appropriate instruction. If the routine is

entered due to an IRQ being received when executing the MSR instruction which disables IRQs, then bit I is set in SPSR. The routine therefore assumes 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 are disabled for the execution time of the IRQ handler. This arrangement cannot be acceptable in a system where FIQs must not be disabled for more than a few cycles.

9.6.2 Workaround

There are 3 suggested workarounds. The one which is most applicable depends upon the requirements of the particular system.

9.6.3 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 legs
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 remains pending since we have not
                          ; acknowledged it and is reissued when interrupts
                          ; are next enabled.
                          ; Rest of interrupt routine
```

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

In order to resolve the first issue, similar code can also be applied to the FIQ handler.

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

9.6.4 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 arrangement 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.

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

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

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

This arrangement requires modification of only the IRQ handler, and FIQs can be re-enabled more quickly than by using workaround 1. However, use it only if the system can guarantee that FIQs are never disabled while IRQs are enabled. It does not address problem one.

9.7 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 method is necessary because all the exception vectors are at addresses 0x0 and above, and easily achieved by configuring register MEMMAP (see Section 10.7.1 “Memory mapping control register (MEMMAP - 0xE01F C040)” on page 43) to User RAM mode. Link the application code so that 0x4000 0000 resides at the Interrupt Vector Table (IVT).

Although multiple sources can be selected (VICIntSelect) to generate FIQ request, use one dedicated interrupt service routine to service all available/present FIQ request(s). Therefore, if several 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 is classified as FIQ. Classifying more than one interrupt source as FIQ increases the interrupt latency.

Following the completion of the desired interrupt service routine, clearing of the interrupt flag on the peripheral level propagates 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 register VICVectAddr before the return from interrupt is executed. This write clears the respective interrupt flag in the internal interrupt priority hardware.

In order to disable the interrupt at the VIC, clear the corresponding bit in register VICIntEnClr, which in turn clears the related bit in register VICIntEnable. This also applies to the VICSoftInt and VICSoftIntClear in which VICSoftIntClear clears the respective bits in VICSoftInt. For example, VICSoftIntClear = 0x0000 0001 clears VICSoftInt = 0x0000 0005 if bit 0 must be cleared. Assign VICSoftIntClear = 0x0000 0000 before the new clear operation is next performed on the same bit in VICSoftInt by writing to VICSoftIntClear. Therefore writing logic 1 to any bit in register Clear has a 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 can 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 is one possibility for VIC setup:

```
VICIntSelect = 0x0000 0000     ; SPI0, I2C0, UART1 and UART0 are IRQ =>
                               ; bit10, bit9, bit7 and bit6=0
```

```
VICIntEnable = 0x0000 06C0      ; SPI0, I2C0, 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 (that is, 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 IRQ requests (SPI0, I^2C, UART0 or UART1) are made, the MPT612 redirects code execution to the address specified at location 0x0000 0018. For vectored and non-vectored IRQs the following instruction can be placed at 0x0000 0018:

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

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

In case UART0, request is made, VICVectAddr is identical to VICVectAddr0, while in case SPI0 request is made, the value from VICVectAddr1 is found here. If either UART0 or SPI0 have not generated an IRQ request but UART1 and/or I^2C are the reason, the content of VICVectAddr is identical to VICDefVectAddr.

10. System control block

10.1 Summary of system control block functions

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
- Wake-up timer

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

10.2 Pin description

Table 34 shows pins that are associated with system control block functions.

Table 34. 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**: active LOW/HIGH level or falling/rising edge general purpose interrupt input. Pin can be used to wake up the processor from Idle or Power-down modes. pin PIO16 can be selected to perform EINT0 function.
EINT1	input	**external interrupt input 1**: see EINT0 description. pin PIO14 can be selected to perform EINT1 function. **Remark:** LOW level on pin PIO14 immediately after reset is considered as external hardware request to start ISP command handler. see Section 25.5 “Description” on page 217 for more details on ISP and serial boot loader.
EINT2	input	**external interrupt input 2**: see EINT0 description. pin PIO15 can be selected to perform EINT2 function.
$\overline{\text{RESET}}$	input	**external reset input**: a LOW on this pin resets the chip changing I/O ports and peripherals to their default states, and processor to start execution at address 0x0000 0000.

10.3 Register description

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

Table 35. Summary of system control registers

Name	Description	Access	Reset value[1]	Address
External interrupts				
EXTINT	external interrupt flag register	R/W	0	0xE01F C140
INTWAKE	interrupt wake-up register	R/W	0	0xE01F C144
EXTMODE	external interrupt mode register	R/W	0	0xE01F C148
EXTPOLAR	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	0x03BE	0xE01F C0C4
APB divider				
APBDIV	APB divider control	R/W	0	0xE01F C100

Table 35. Summary of system control registers *…continued*

Name	Description	Access	Reset value[1]	Address
Reset				
RSIR	reset source identification register	R/W	0	0xE01F C180
Code security/debugging				
CSPR	code security protection register	RO	0	0xE01F C184
Syscon miscellaneous registers				
SCS	system controls and status	R/W	0	0xE01F C1A0

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

10.4 Crystal oscillator

The MPT612 onboard oscillator circuit supports external crystals in the range 1 MHz to 25 MHz. If the on-chip PLL system or the bootloader is used, the input clock frequency is limited to an exclusive range 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 such as rate equations found elsewhere in this document. f_{osc} and CCLK are the same value unless the PLL is running and connected. Refer to the Section 10.8 “Phase-Locked Loop (PLL)” on page 43 for details and frequency limitations.

The onboard oscillator in the MPT612 can operate in one of two modes: slave mode and oscillation mode.

In slave mode, couple the input clock signal with a capacitor of 100 pF (C_C in Figure 8, drawing a), with an amplitude of at least 200 mV (RMS). Pin XTAL2 in this configuration can be left unconnected. If slave mode is selected, the f_{osc} signal of 50-to-50 duty cycle can range from 1 MHz to 25 MHz.

External components and models used in oscillation mode are shown in Figure 8, drawings b and c, and in Table 36. Since the feedback resistance is integrated on chip, only a crystal and the capacitors C_{X1} and C_{X2} must be connected externally in case of fundamental mode oscillation (L, C_L and R_S represent the fundamental frequency). Capacitance C_P in Figure 8, drawing c, represents the parallel package capacitance and must not be larger than 7 pF. Parameters f_C, C_L, R_S and C_P are supplied by the crystal manufacturer.

Choosing an oscillation mode as an on-board oscillator mode of operation limits f_{osc} clock selection to 1 MHz to 25 MHz.

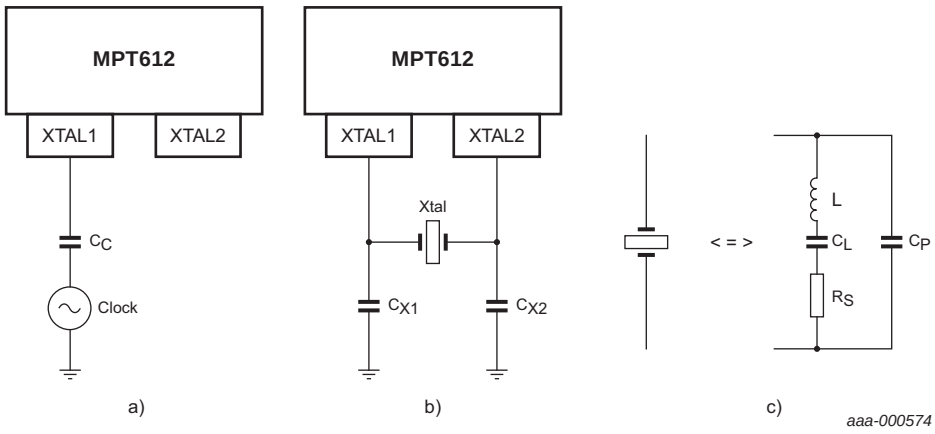


Fig 8. Oscillator modes and models: a) slave mode of operation, b) oscillation mode of operation, c) external crystal model used for $C_{X1}/_{X2}$ evaluation

Table 36. 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 to 5 MHz	10 pF	n/a	n/a
	20 pF	n/a	n/a
	30 pF	< 300 Ω	58 pF, 58 pF
5 MHz to 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 to 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 to 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 to 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

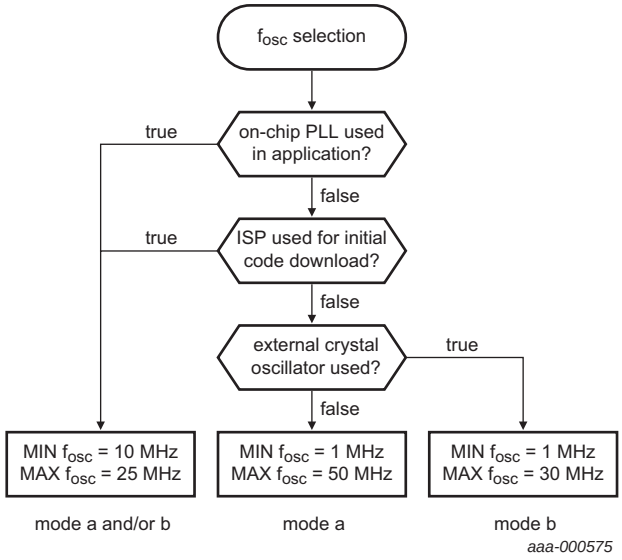


Fig 9. f_{osc} selection algorithm

10.4.1 XTAL1 input

The input voltage to the on-chip oscillators is limited to 1.8 V. If a clock drives the oscillator in slave mode, it is recommended that the input is coupled through a capacitor with C_i = 100 pF. To limit the input voltage to the specified range, choose an additional capacitor to ground C_g which attenuates the input voltage by a factor $C_i/(C_i + C_g)$. In slave mode, a minimum of 200 mV_{rms} is needed.

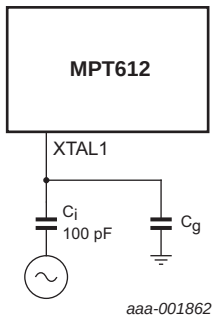


Fig 10. Slave mode operation of the on-chip oscillator

10.4.2 XTAL and RTC Printed-Circuit Board (PCB) layout guidelines

Place the crystal on the PCB as close as possible to the chip oscillator input and output pins. Take care that the load capacitors C_{x1} and C_{x2}, and C_{x3} in case of third overtone crystal usage, have a common ground plane. Connect the external components to the ground plain. Make loops as small as possible to reduce the noise coupled in via the PCB as small as possible. Parasitics must be as small as possible. Start with small values for C_{x1} and C_{x2} and, if necessary, increase the value in proportion to the level of parasitics on the PCB layout.

10.5 External interrupt inputs

The MPT612 includes up to three external interrupt inputs as selectable pin functions. When the pins are combined, external events can be processed as three independent interrupt signals. The external interrupt inputs can optionally be used to wake up the processor from Power-down or Deep power-down mode.

Additionally, all 10 capture inputs can also be used as external interrupts without the option to wake up the device from Power-down mode.

10.5.1 Register description

The external interrupt function has four registers associated with it. Register EXTINT contains the interrupt flags and register EXTWAKE contains bits that enable individual external interrupts to wake up the MPT612 from Power-down mode. Registers EXTMODE and EXTPOLAR specify the level and edge sensitivity parameters.

Table 37. External interrupt registers

Name	Description	Access	Reset value[1]	Address
EXTINT	external interrupt flag register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3; see Table 38	R/W	0	0xE01F C140
INTWAKE	interrupt wake-up register contains four enable bits that control whether each external interrupt causes the processor to wake up from Power-down mode; see Table 39.	R/W	0	0xE01F C144
EXTMODE	external interrupt mode register controls whether each pin is edge- or level sensitive.	R/W	0	0xE01F C148
EXTPOLAR	external interrupt polarity register controls which level or edge on each pin causes 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.

10.5.2 External interrupt flag register (EXTINT - 0xE01F C140)

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

Writing ones to bits EINT0 through EINT2 in register EXTINT clears the corresponding bits. In level-sensitive mode, this action is efficacious 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 wake-up and/or external interrupt), this bit in register EXTINT must be cleared. Otherwise the event triggered by activity on pin EINT is not recognized in the future.

Remark: when a change of external interrupt operating mode (that is, active level/edge) is performed (including the initialization of an external interrupt), the corresponding bit in register EXTINT must be cleared. For details, see Section 10.5.4 and Section 10.5.5.

For example, if a system wakes up from power-down using a LOW level on external interrupt 0 pin, its post-wake-up code must reset bit EINT0 to allow future entry into the power-down mode. If bit EINT0 is left set to logic 1, subsequent attempt(s) to invoke power-down mode fail. This also applies to external interrupt handling.

More details on power-down mode are discussed in the following chapters.

Table 38. 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 EINT0 function is selected for its pin, and pin is in its active state. In edge-sensitive mode, this bit is set if EINT0 function is selected for its pin, and the selected edge occurs on the pin. cleared by writing a logic 1 to it, except in level-sensitive mode when pin is in its active state (for example, if EINT0 is selected to be LOW, level-sensitive and a LOW level is present on the corresponding pin, this bit cannot be cleared; it can only be cleared when signal on pin is HIGH).	0
1	EINT1	in level-sensitive mode, this bit is set if EINT1 function is selected for its pin, and pin is in its active state. In edge-sensitive mode, this bit is set if EINT1 function is selected for its pin, and the selected edge occurs on the pin. cleared by writing a logic 1 to it, except in level-sensitive mode when pin is in its active state (for example, if EINT1 is selected to be LOW, level-sensitive and a LOW level is present on the corresponding pin, this bit cannot be cleared; it can only be cleared when signal on pin is HIGH).	0
2	EINT2	in level-sensitive mode, this bit is set if EINT2 function is selected for its pin, and the pin is in its active state. In edge-sensitive mode, this bit is set if EINT2 function is selected for its pin, and the selected edge occurs on the pin. cleared by writing a logic 1 to it, except in level-sensitive mode when pin is in its active state (for example, if EINT2 is selected to be LOW, level-sensitive and a LOW level is present on the corresponding pin, this bit cannot be cleared; it can be cleared only when the signal on the pin is HIGH).	0
7:3	-	reserved; user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.5.3 Interrupt wake-up register (INTWAKE - 0xE01F C144)

Enable bits in register INTWAKE allow the external interrupts and other sources to wake up the processor if it is in Power-down mode. Map the related EINTn function to the pin in order for the wake-up process to take place. It is not necessary for the interrupt to be enabled in the Vectored Interrupt Controller for a wake-up to take place. This arrangement allows capabilities, such as 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 if it is asserted (eliminating the need to disable the interrupt if the wake-up feature is not desirable in the application).

If an external interrupt pin is required to be a source for waking the MPT612 from Power-down mode, the corresponding bit in register External Interrupt Flag must be cleared; see Section 10.5.2.

Table 39. Interrupt wake-up register (INTWAKE - address 0xE01F C144) bit description

Bit	Symbol	Description	Reset value
0	EXTWAKE0	if logic 1, assertion of EINT0 wakes up processor from Power-down mode	0
1	EXTWAKE1	if logic 1, assertion of EINT1 wakes up the processor from Power-down mode	0
2	EXTWAKE2	if logic 1, assertion of EINT2 wakes up processor from Power-down mode	0
14:3	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
15	RTCWAKE	when logic 1, assertion of an RTC interrupt wakes up processor from Power-down mode	0

Remark: A LOW level applied to the external interrupt inputs EINT[2:0] always wakes the chip from Deep power-down mode regardless of the settings in registers INTWAKE or PINSEL. Waking up from Deep power-down mode through the EINT pins cannot be disabled.

10.5.4 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 12.4 “Register description” on page 62) and enabled via register VICIntEnable (see Section 9.4 “VIC registers” on page 21) can cause interrupts from the External Interrupt function (though pins selected for other functions can cause interrupts from those functions).

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

Table 40. 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.5.5 External interrupt polarity register (EXTPOLAR - 0xE01F C14C)

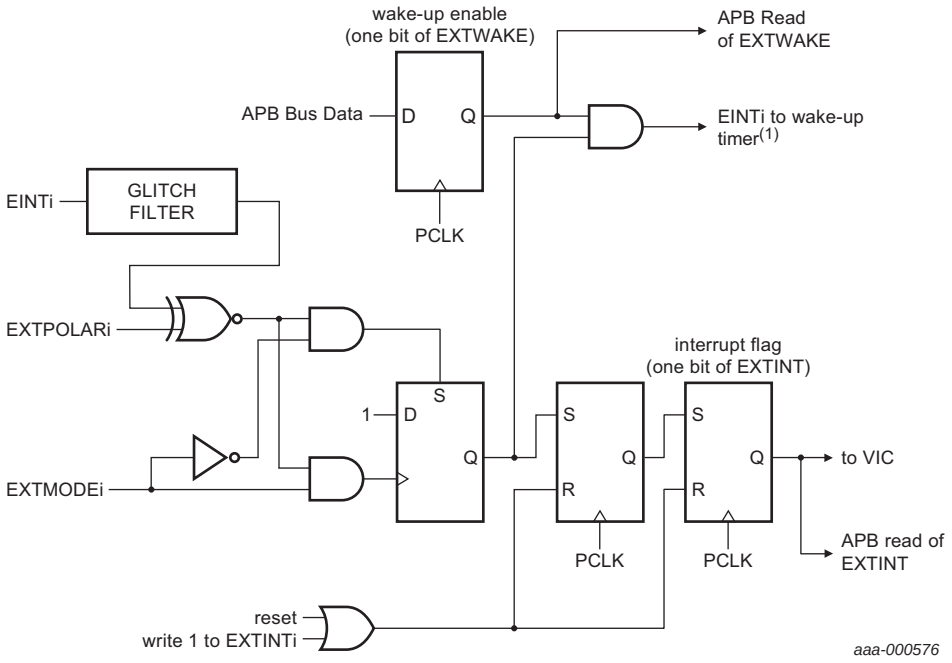
In level-sensitive mode, the bits in this register select whether the corresponding pin is high- or low-active. In edge-sensitive mode, they select whether the pin is rising- or falling-edge sensitive. Only pins that are selected for the EINT function (see Section 12.4

"Register description" on page 62) and enabled in register VICIntEnable (see Section 9.4 "VIC registers" on page 21) can cause interrupts from the External Interrupt function (though pins selected for other functions can cause interrupts from those functions).

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

Table 41. 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a



(1) See Figure 14 “Reset block diagram including the wake-up timer” on page 54.

Fig 11. External interrupt logic

10.6 Other system controls

Some aspects of controlling MPT612 operation that do not fit into peripheral or other registers are grouped as shown below.

10.6.1 System control and status flags register (SCS - 0xE01F C1A0)

Table 42. 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	
		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 13.4.2 “Fast GPIO mask register (FIOMASK, FIO0MASK - 0x3FFF C010)” on page 69	
31:1	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.7 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.

10.7.1 Memory mapping control register (MEMMAP - 0xE01F C040)

Whenever an exception handling is necessary, the MPT612 fetches an instruction residing on the exception corresponding address as described in Table 3 “ARM exception vector locations” on page 11. Register MEMMAP determines the source of data that fills this table.

Table 43. 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 remapped to boot block.	00
		01	User flash mode. Interrupt vectors are not remapped and reside in flash.	
		10	User RAM mode. Interrupt vectors are remapped to static RAM.	
		11	reserved; do not use this option	
		Remark: improper setting of this value can result in incorrect operation of the device.		
7:2	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.7.2 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 always fetches 32-bit data "residing" on 0x0000 0008 see Table 3 “ARM exception vector locations” on page 11. This means that when MEMMAP[1:0] = 10 (User RAM Mode), a read/fetch from 0x0000 0008 provides data stored in 0x4000 0008. If MEMMAP[1:0] = 00 (Boot Loader Mode), a read/fetch from 0x0000 0008 provides data available also at 0x7FFF E008 (boot block remapped from on-chip Bootloader).

10.8 Phase-Locked Loop (PLL)

The PLL accepts an input clock frequency in the range 10 MHz to 25 MHz only. The input frequency is multiplied up in the range 10 MHz to 60 MHz using a Current Controlled Oscillator (CCO). The multiplier can be an integer from 1 to 32; in practice, the multiplier value cannot be higher than 6 on the MPT612 due to the upper frequency limit of the CPU. The CCO operates in the range 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 can 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 ensured that the PLL output has a 50 % duty cycle. A block diagram of the PLL is shown in Figure 12.

PLL activation is controlled via register PLLCON. Register PLLCFG control the PLL multiplier and divider values. In order to prevent accidental alteration of PLL parameters or deactivation of the PLL, these two registers are protected. Since all chip operations, including the watchdog timer, depend on the PLL when it is providing the chip clock,

accidental changes to the PLL setup can result in unexpected behavior of the MPT612. A feed sequence similar to that of the watchdog timer provides the protection. Details are provided in the description of register PLLFEED.

The PLL is turned off and bypassed following a chip reset and when 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.

10.8.1 Register description

The PLL is controlled by the registers shown in Table 44. More detailed descriptions follow.

Remark: Improper setting of the PLL values can result in incorrect operation of the device.

Table 44. PLL registers

Generic 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 occurs.	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 occurs.	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 do not reflect the current PLL state. Reading this register provides actual values controlling the PLL and status of PLL.	RO	0	0xE01F C088
PLLFEED	PLL feed register. Register enables loading of PLL control and configuration information from registers PLLCON and PLLCFG into shadow registers actually affecting PLL operation.	WO	n/a	0xE01F C08C

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

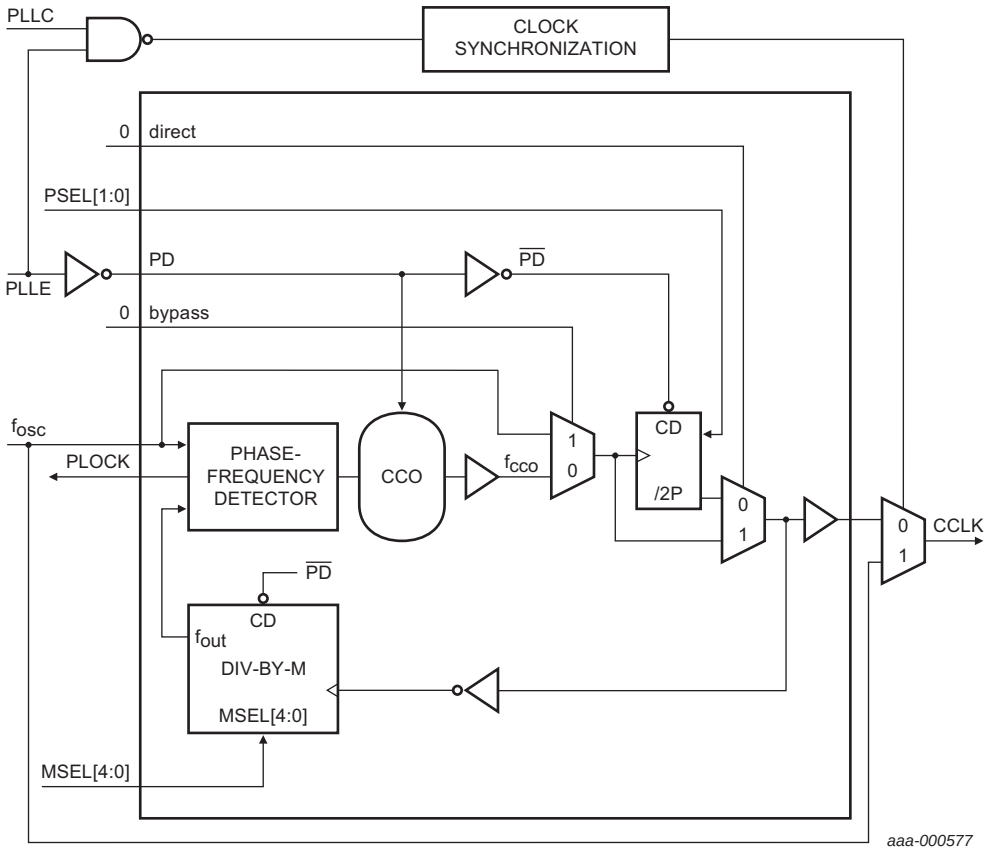


Fig 12. PLL block diagram

10.8.2 PLL control register (PLLCON - 0xE01F C080)

Register PLLCON 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 register PLLCON do not take effect until a correct PLL feed sequence is given (see Section 10.8.7 “PLL Feed register (PLLFEED - 0xE01F C08C)” on page 47 and Section 10.8.3).

Table 45. PLL control register (PLLCON - address 0xE01F C080) bit description

Bit	Symbol	Description	Reset value
0	PLLE	PLL enable. If logic 1, and after a valid PLL feed, this bit activates PLL and allows it to lock to requested frequency; see register PLLSTAT, Table 47.	0
1	PLLC	PLL connect. If PLLC and PLLE are both set to logic 1, and after a valid PLL feed, connects PLL as clock source for MPT612. Otherwise, oscillator clock is used directly by MPT612; see register PLLSTAT, Table 47.	0
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

The PLL must be set up, enabled, and lock established before it can 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. If lock is lost during operation, hardware does not ensure that the PLL is locked before it is connected nor automatically disconnects the PLL. If PLL lock is lost, it is likely that the oscillator clock will become unstable which cannot be remedied by disconnecting the PLL.

10.8.3 PLL configuration register (PLLCFG - 0xE01F C084)

Register PLLCFG contains the PLL multiplier and divider values. Changes to register PLLCFG do not take effect until a correct PLL feed sequence is given (see Section 10.8.7). Calculations for the PLL frequency, and multiplier and divider values are found in the PLL frequency calculation in Section 10.8.9.

Table 46. PLL configuration register (PLLCFG - address 0xE01F C084) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	PLL multiplier value. Supplies value "M" in PLL frequency calculations. **Remark:** For details on selecting correct value for MSEL, see Section 10.8.9.	0
6:5	PSEL	PLL divider value. Supplies value "P" in PLL frequency calculations. **Remark:** For details on selecting correct value for PSEL, see Section 10.8.9.	0
7	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.8.4 PLL status register (PLLSTAT - 0xE01F C088)

The read-only register PLLSTAT provides the actual PLL parameters that are in effect at the time it is read, as well as the PLL status. PLLSTAT can 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 10.8.7).

Table 47. PLL status register (PLLSTAT - address 0xE01F C088) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	read-back for the PLL multiplier value. Value currently used by PLL.	0
6:5	PSEL	read-back for the PLL divider value. Value currently used by PLL.	0
7	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
8	PLLE	read-back for bit PLL Enable. When logic 1, PLL is currently activated. When logic 0, PLL is off. Automatically cleared when Power-down mode activated.	0

Table 47. PLL status register (PLLSTAT - address 0xE01F C088) bit description ...continued

Bit	Symbol	Description	Reset value
9	PLLC	read-back for bit PLL Connect. When PLLC and PLLE are both logic 1, the PLL is connected as the clock source for the MPT612. When either PLLC or PLLE is logic 0, the PLL is bypassed and the oscillator clock is used directly by the MPT612. This bit is automatically cleared when Power-down mode is activated.	0
10	PLOCK	reflects the PLL Lock status. When logic 0, the PLL is not locked. When logic 1, the PLL is locked onto the requested frequency.	0
15:11	-	reserved, user software must not write logic 1s to reserved bits; the value read from a reserved bit is not defined	n/a

10.8.5 PLL interrupt

Bit PLOCK in register PLLSTAT 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 can be connected, and the interrupt disabled. For details on how to enable and disable the PLL interrupt, see Section 9.4.4 “Interrupt enable register (VICIntEnable - 0xFFFF F010)” on page 23 and Section 9.4.5 “Interrupt enable clear register (VICIntEnClear - 0xFFFF F014)” on page 23.

10.8.6 PLL modes

The combinations of PLLE and PLLC are shown in Table 48.

Table 48. PLL Control bit combinations

PLLC	PLLE	PLL Function
0	0	PLL is off and disconnected. CCLK equals the unmodified clock input.
0	1	PLL is active, but not yet connected. PLL can be connected after PLOCK is asserted.
1	0	same as 00 combination. Prevents possibility of PLL being connected without also being enabled.
1	1	PLL is active and connected. CCLK/system clock is sourced from the PLL.

10.8.7 PLL Feed register (PLLFEED - 0xE01F C08C)

In order for changes to registers PLLCON and PLLCFG to take effect, write a correct feed sequence to register PLLFEED. 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 register PLLCON or PLLCFG are not effective.

Table 49. PLL Feed register (PLLFEED - address 0xE01F C08C) bit description

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

10.8.8 PLL and Power-down mode

Power-down mode automatically turns off and disconnects activated PLL(s). Wake-up 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 wake-up. It is important not to attempt to restart the PLL by simply feeding it when execution resumes after a wake-up from Power-down mode. This enables and connects the PLL at the same time, before PLL lock is established.

10.8.9 PLL frequency calculation

The PLL equations use the following parameters:

Table 50. Parameters determining PLL frequency

Element	Description
f_{osc}	frequency of crystal oscillator/external oscillator
f_{CCO}	frequency of PLL current controlled oscillator
CCLK	PLL output frequency (also processor clock frequency)
M	PLL multiplier value from MSEL bits in register PLLCFG
P	PLL divider value from PSEL bits in register PLLCFG

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 requirements:

- f_{osc} is in the range 10 MHz to 25 MHz.
- CCLK is in the range 10 MHz to f_{max} (the maximum allowed frequency for the MPT612 - embedded in and determined by the system MPT612).
- f_{CCO} is in the range 156 MHz to 320 MHz.

10.8.10 Procedure for determining PLL settings

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

1. Choose the desired processor operating frequency (CCLK). This can be based on processor throughput requirements, such as the need to support a specific set of UART baud rates, and so on, Bear in mind that peripheral devices can be running from a lower clock than the processor (see Section 10.11 “APB Divider” on page 54).
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 1 to 32. The value written to the MSEL bits in PLLCFG is M − 1; see Table 52.
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 bits PSEL in PLLCFG is 00 for P = 1; 01 for P = 2; 10 for P = 4; 11 for P = 8 (see Table 51).

Table 51. PLL Divider values

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

Table 52. PLL Multiplier values

MSEL bits (PLLCFG bits [4:0])	Value of M
0 0000	1
0 0001	2
0 0010	3
0 0011	4
...	...
1 1110	31
1 1111	32

10.8.11 PLL configuration examples

Example: a PLL configuration application.

System design asks for f_{osc} = 10 MHz and requires CCLK = 60 MHz.

Based on these specifications, M = CCLK / f_{osc} = 60 MHz / 10 MHz = 6. Consequently, M − 1 = 5 is written as PLLCFG[4:0].

Value for P can be derived from P = f_{CCO} / (CCLK × 2), using condition that f_{CCO} must be in the range 156 MHz to 320 MHz. Assuming the lowest allowed frequency for f_{CCO} = 156 MHz, P = 156 MHz / (2 × 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 51 is P = 2. Therefore, PLLCFG[6:5] = 1 is used.

10.9 Power control

The MPT612 supports three reduced power modes: Idle mode, Power-down mode, and Deep 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 can generate interrupts to cause the processor to resume execution. Idle mode eliminates power used by the processor, 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.

If the RTC is running with its external 32 kHz oscillator at the time of entry into Power-down mode, operation can resume using an interrupt from the RTC; see Section 24.6.1 “RTC interrupts” on page 205.

Use program execution to coordinate entry to Power-down and Idle modes. Wake-up 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 10.12 “Wake-up timer” on page 56.

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.

The Deep power-down mode is controlled through the RTC block (see Section 24.6.14 “Power control register group” on page 210). In Deep power-down mode, all power is removed from the internal chip logic except for the RTC module, the I/O ports, the SRAM and the 32 kHz external oscillator. For additional power savings, SRAM and the 32 kHz oscillator can be powered down individually. The Deep power-down mode produces the lowest possible power consumption without actually removing power from the entire chip. In Deep power-down mode, the contents of registers and memory are not preserved except for SRAM, if selected, and three general-purpose registers. Therefore, to resume operations, a full chip reset process is required.

10.9.1 Register description

The Power Control function contains two registers, as shown in Table 53. More detailed descriptions follow. The Deep power-down mode is controlled through the RTC block (see Section 24.6.14 “Power control register group” on page 210).

Table 53. Power control registers

Name	Description	Access	Reset value[1]	Address
PCON	power control register. Contains control bits that enable the two reduced power operating modes of the MPT612; see Table 54.	R/W	0x00	0xE01F C0C0
PCONP	power control for peripherals register. Contains control bits that enable and disable individual peripheral functions, allowing elimination of power consumption by peripherals that are not needed.	R/W	0x0018 17BE	0xE01F C0C4

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

10.9.2 Power control register (PCON - 0xE01F COCO)

Register PCON contains 2 bits. Writing a logic 1 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 54. Power control register (PCON - address 0xE01F COCO) bit description

Bit	Symbol	Description	Reset value
0	IDL	Idle mode. If logic 1, causes processor clock to stop, while on-chip peripherals remain active. Any enabled interrupt from a peripheral or an external interrupt source causes processor to resume execution.	0
1	PD	Power-down mode. If logic 1, causes oscillator and all on-chip clocks to stop. A wake-up condition from an external interrupt can cause the oscillator to restart, bit PD to be cleared, and processor to resume execution.	0
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined.	n/a

10.9.3 Power control for peripherals register (PCONP - 0xE01F COC4)

To save power, register PCONP turns off selected peripheral functions. This is accomplished by gating off the clock source to the specified peripheral blocks. A few peripheral functions cannot be turned off (that is, the watchdog timer, GPIO, the pin connect block, and the system control block). Some peripherals, particularly analog functions, can consume power that is not clock-dependent. These peripherals can 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 “APB peripheries and base addresses” on page 10 in Section 7.1 “Memory maps” on page 7.

If a peripheral control bit is logic 1, that peripheral is enabled. If a peripheral bit is logic 0, that peripheral is disabled to conserve power. For example if bit 19 is logic 1, the I^2C1? interface is enabled. If bit 19 is logic 0, the I^2C1? interface is disabled.

Remark: A valid read from a peripheral register and a valid write to a peripheral register is possible only if the peripheral is enabled in register PCONP.

Table 55. Power control for peripherals register (PCONP - address 0xE01F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
1	-	reserved	1
2	PCTIM1	timer counter1 power/clock control bit	1
3	PCUART0	UART0 power/clock control bit	1
4	PCUART1	UART1 power/clock control bit	1
6:5	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7	PCI2C0	I^2C0? interface power/clock control bit	1
8	PCSPI	SPI interface power/clock control bit	1
9	PCRTC	RTC power/clock control bit	1
10	PCSPI	SSP interface power/clock control bit	1
11	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Table 55. Power control for peripherals register (PCONP - address 0xE01F C0C4) bit description *...continued*

Bit	Symbol	Description	Reset value
12	PCAD	A/D converter 0 (ADC0) power/clock control bit. **Remark:** Clear bit PDN in ADCR before clearing this bit, and set this bit before setting PDN.	1
18:13	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
19	PCI2C1	I^2C1 interface power/clock control bit	1
21:20	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
22	-	reserved	1
23	PCTIM3	timer counter3 power/clock control bit	1
31:24	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

10.9.4 Power control usage notes

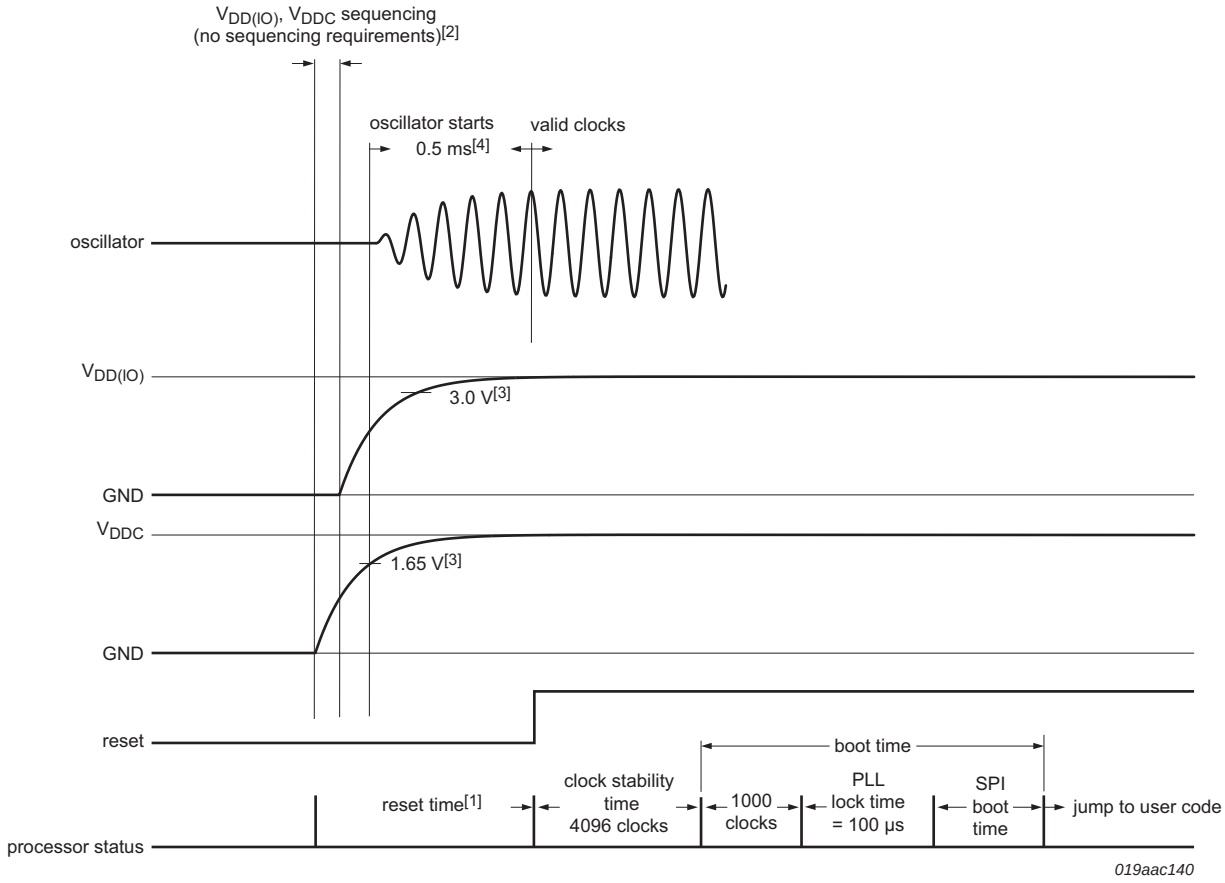
After every reset, register PCONP contains the value that enables all interfaces and peripherals controlled by the PCONP to be enabled. Therefore, in order to start using any of the on-board peripherals, apart from proper configuring via peripheral dedicated registers, the application has no need to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

Power saving oriented systems must have logic 1s in register PCONP only in positions that match peripherals 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 logic 0.

10.10 Reset

Reset has two sources on the MPT612: pin $\overline{\text{RESET}}$ and watchdog reset. Pin $\overline{\text{RESET}}$ is a Schmitt trigger input pin with an additional glitch filter. Assertion of chip reset by any source starts the wake-up timer (see Section 10.12 “Wake-up timer”), 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 wake-up timer are shown in Figure 13. The reset logic is shown in Figure 14.

The reset glitch filter allows the processor to ignore external reset pulses that are short, and also determines the minimum duration of $\overline{\text{RESET}}$ that must be asserted in order to guarantee a chip reset. Once asserted, pin $\overline{\text{RESET}}$ can be deasserted only when the crystal oscillator is fully running and an adequate signal is present on pin XTAL1 of the MPT612. Assuming that an external crystal is used in the crystal oscillator subsystem, after power-on, pin $\overline{\text{RESET}}$ must be asserted for 10 ms. For all subsequent resets when the crystal oscillator is already running and a stable signal is on pin XTAL1, pin $\overline{\text{RESET}}$ needs to be asserted for only 300 ns.



(1) Reset time: The reset time must be held LOW. This time depends on system parameters such as V_{DDC}, $V_{DD(IO)}$ rise time, and the oscillator start-up time. There are no restrictions from the MPT612 except that V_{DDC}, $V_{DD(IO)}$, and the oscillator must be within the specific operating range.

(2) There are no sequencing requirements for $V_{DD(IO)}$ and V_{DDC}.

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

(4) Typical start-up time is 0.5 ms for a 12 MHz crystal.

Fig 13. Start-up sequence diagram

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.

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. Pin 26 (RTCK) is examined during an external reset (see Section 11.2 on page 58 and Section 12.4 on page 62). Pin PIO14 (see Section 25 “Flash memory system and programming” on page 217) examines the on-chip bootloader when this code is executed after every reset.

It is possible for a chip reset to occur during a Flash programming or erase operation. The Flash memory interrupts the ongoing operation and holds off the completion of reset to the CPU until internal Flash high voltages have settled.

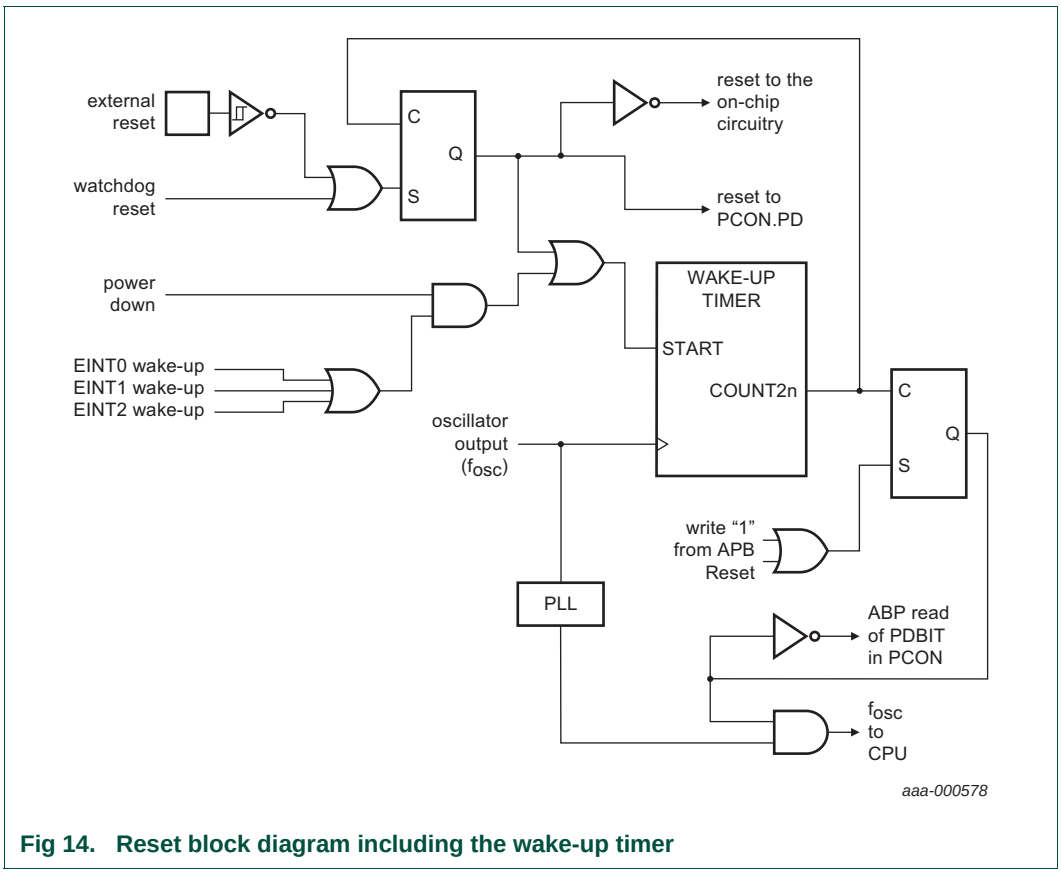


Fig 14. Reset block diagram including the wake-up timer

10.10.1 Reset source identification register (RSIR - 0xE01F C180)

This register contains 1 bit for each source of reset. Writing a logic 1 to any of these bits clears the corresponding read-side bit to logic 0. The interactions among the four sources are described below.

Table 56. Reset source identification register (RSIR - address 0xE01F C180) bit description

Bit	Symbol	Description	Reset value
0	POR	Power-On Reset (POR) event sets this bit, and clears all other bits in this register. If another reset signal (such as External Reset) remains asserted after the POR signal is negated, then its bit is set. This bit is not affected by any of the other reset sources.	see text
1	EXTR	assertion of the $\overline{\text{RESET}}$ signal sets this bit. This bit is cleared by POR, but is not affected by WDT reset.	see text
2	WDTR	this bit is set when the watchdog timer times out and bit WDTRESET in the watchdog mode register is logic 1. It is cleared by any of the other reset sources.	see text
7:3	-	reserved, user software must not write logic 1s to reserved bits; the value read from a reserved bit is not defined	n/a

10.11 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:

- provides peripherals with desired PCLK via APB bus so that they can operate at the speed chosen for the ARM processor. To achieve this, the APB bus can 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 as 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.
- allows 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 15 on page 56. Because the APB Divider is connected to the PLL output, the PLL remains active (if it was running) during Idle mode.

10.11.1 Register description

Only one register is used to control the APB Divider.

Table 57. APB Divider register map

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

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

10.11.2 APBDIV register (APBDIV - 0xE01F C100)

Register APB Divider contains 2 bits, allowing three divider values, as shown in Table 58.

Table 58. 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 register APBDIV it has no effect (previous setting is retained)	
7:2	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

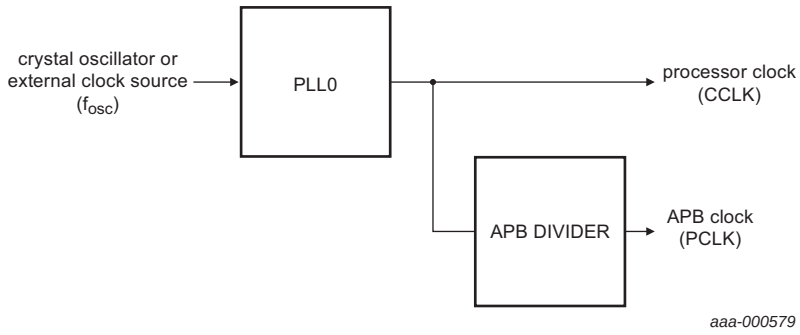


Fig 15. APB Divider connections

10.12 Wake-up timer

The purpose of the wake-up 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 previously mentioned functions are turned off for any reason. Since the oscillator and other functions are turned off during Power-down mode, any wake-up of the processor from Power-down mode uses the wake-up timer.

The wake-up timer monitors the crystal oscillator as the means of checking 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 (for example, capacitors), and the characteristics of the oscillator itself under the existing ambient conditions.

Once a clock is detected, the wake-up timer counts 4096 clocks, then enables the on-chip circuitry to initialize. When the on-board module's initialization is complete, the processor is released to execute instructions if the external reset is de-asserted. In the case where an external clock source is used in the system (as opposed to a crystal connected to the oscillator pins), the possibility that there can be little or no delay for oscillator start-up must be considered. The wake-up timer design then ensures that any other required chip functions are operational before the beginning of program execution.

Any of the various resets can bring the MPT612 out of Power-down mode, as can the external interrupts EINT2:0 and the RTC interrupt if the RTC is operating from its own oscillator on the RTCX1-2 pins. When one of these interrupts is enabled for wake-up and its selected event occurs, an oscillator wake-up cycle is started. The actual interrupt (if any) occurs after the wake-up timer expires, and is handled by the Vectored Interrupt Controller.

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 must reprogram the pin function to external interrupt, select the appropriate mode and polarity for the interrupt, and then select Power-down mode. At wake-up, software must restore the pin multiplexing to the peripheral function.

To summarize: on the MPT612, the wake-up timer enforces a minimum reset duration based on the crystal oscillator, and is activated whenever there is a wake-up from Power-down mode or any type of reset.

10.13 Code security vs. debugging

Applications in development typically need the debugging and tracing facilities in the MPT612. Later in the life cycle of an application, it can be more important to protect the application code from observation by hostile or competitive eyes. The Code Read Protection feature on the MPT612 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 25.8 “Code Read Protection (CRP)” on page 223.

11. Pin configuration

11.1 Pinout

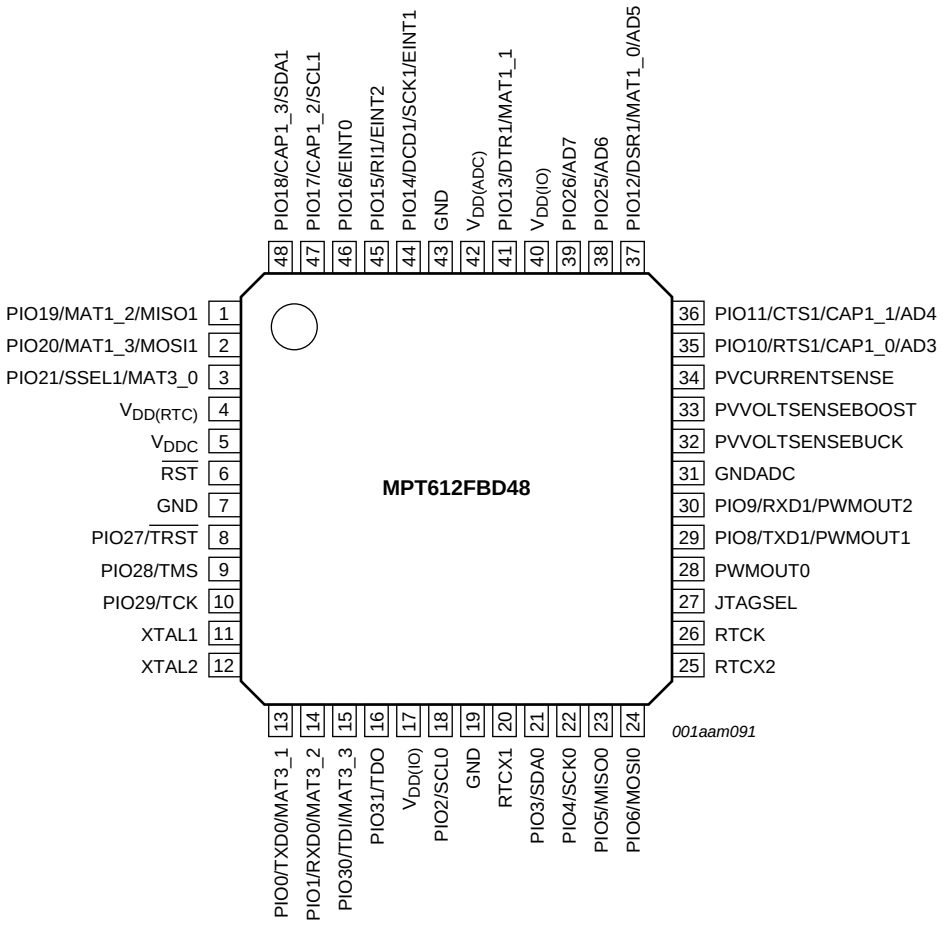


Fig 16. LQFP48 pin configuration

11.2 MPT612 pin description

The MPT612 pin functions are briefly described in Table 59.

Table 59. Pin description

Symbol	Pin	Type	Description
PIO0 to PIO31		I/O	**Port 0**: Port 0 is a 32-bit I/O port with individual direction controls for each bit. A total of 31 pins of Port 0 can be used as general purpose bidirectional digital I/Os while PIO31 is an output-only pin. Operation of port 0 pins depends upon pin function selected via pin connect block.
PIO0/TXD0/ MAT3_1	13[1]	I/O	**PIO0**: general purpose input/output digital pin
		O	**TXD0**: transmitter output for UART0
		O	**MAT3_1**: PWM output 1 for timer 3
PIO1/RXD0/ MAT3_2	14[1]	I/O	**PIO1**: general purpose input/output digital pin
		I	**RXD0**: receiver input for UART0
		O	**MAT3_2**: PWM output 2 for timer 3
PIO2/SCL0	18[2]	I/O	**PIO2**: general purpose input/output digital pin; output is open-drain
		I/O	**SCL0**: I^2C0 clock Input/output; open-drain output (for I^2C-bus compliance)
PIO3/SDA0	21[2]	I/O	**PIO3**: general purpose input/output digital pin; output is open-drain
		I/O	**SDA0**: I^2C0 data input/output; open-drain output (for I^2C-bus compliance)
PIO4/SCK0	22[1]	I/O	**PIO4**: general purpose input/output digital pin
		I/O	**SCK0**: serial clock for SPI0; SPI clock output from master or input to slave
PIO5/MISO0	23[1]	I/O	**PIO5**: general purpose input/output digital pin
		I/O	**MISO0**: master in slave out for SPI0. Data input to SPI master or data output from SPI slave
PIO6/MOSI0	24[1]	I/O	**PIO6**: general purpose input/output digital pin
		I/O	**MOSI0**: master out slave in for SPI0. Data output from SPI master or data input to SPI slave
PWMOUT0	28[1]	O	**PWMOUT0**: PWM output used for switching the MOSFET; do not use for any other purpose
PIO8/TXD1/ PWMOUT1	29[1]	I/O	**PIO8**: general purpose input/output digital pin
		O	**TXD1**: transmitter output for UART1
		O	**PWMOUT1**: PWM output; same frequency as PWMOUT0, however, the duty cycle can be changed
PIO9/RXD1/ PWMOUT2	30[1]	I/O	**PIO9**: general purpose input/output digital pin
		I	**RXDI**: receiver input for UART1
		O	**PWMOUT2**: PWM output; same frequency as PWMOUT0, however, the duty cycle can be changed
PIO10/RTS1/ CAP1_0/AD3	35[3]	I/O	**PIO10**: general purpose input/output digital pin
		O	**RTS1**: request to send output for UART1
		I	**CAP1_0**: capture input for timer 1, channel 0
		I	**AD3**: ADC 0, input 3
PIO11/CTS1/ CAP1_1/AD4	36[3]	I/O	**PIO11**: general purpose input/output digital pin
		I	**CTS1**: clear to send input for UART1
		I	**CAP1_1**: capture input for timer 1, channel 1
		I	**AD4**: ADC 0, input 4

Table 59. Pin description *...continued*

Symbol	Pin	Type	Description
PIO12/DSR1/ MAT1_0/AD5	37[3]	I/O	**PIO12**: general purpose input/output digital pin
		I	**DSR1**: data set ready input for UART1
		O	**MAT1_0**: PWM output for Timer 1, channel 0
		I	**AD5**: ADC 0, input 5
PIO13/DTR1/ MAT1_1	41[1]	I/O	**PIO13**: general purpose input/output digital pin
		O	**DTR1**: data terminal ready output for UART1
		O	**MAT1_1**: PWM output for timer 1, channel 1
PIO14/DCD1/ SCK1/EINT1	44[4][5]	I/O	**PIO14**: general purpose input/output digital pin
		I	**DCD1**: data carrier detect input for UART1
		I/O	**SCK1**: serial clock for SPI1. SPI clock output from master or input to slave
		I	**EINT1**: external interrupt 1 input
PIO15/RI1/ EINT2	45[4]	I/O	**PIO15**: general purpose input/output digital pin
		I	**RI1**: ring indicator input for UART1
		I	**EINT2**: external interrupt 2 input
PIO16/EINT0	46[4]	I/O	**PIO16**: general purpose input/output digital pin
		I	**EINT0**: external interrupt 0 input
PIO17/CAP1_2 / SCL1	47[6]	I/O	**PIO17**: general purpose input/output digital pin. The output is not open-drain.
		I	**CAP1_2**: capture input for timer 1, channel 2
		I/O	**SCL1**: I^2C1 clock input/output. This pin is an open-drain output if I^2C1 function is selected in the pin connect block.
PIO18/CAP1_3 / SDA1	48[6]	I/O	**PIO18**: general purpose input/output digital pin. The output is not open-drain.
		I	**CAP1_3**: capture input for timer 1, channel 3
		I/O	**SDA1**: I^2C1 data input/output. This pin is an open-drain output if I^2C1 function is selected in the pin connect block.
PIO19/MAT1_2 / MISO1	1[1]	I/O	**PIO19**: general purpose input/output digital pin
		O	**MAT1_2**: PWM output for timer 1, channel 2
		I/O	**MISO1**: master in slave out for SSP. Data input to SSP master or data output from SSP slave.
PIO20/MAT1_3 / MOSI1	2[1]	I/O	**PIO20**: general purpose input/output digital pin
		O	**MAT1_3**: PWM output for timer 1, channel 3
		I/O	**MOSI1**: master out slave In for SSP. Data output from SSP master or data input to SSP slave.
PIO21/SSEL1/ MAT3_0	3[1]	I/O	**PIO21**: general purpose input/output digital pin
		I	**SSEL1**: slave select for SPI1. Selects the SPI interface as a slave.
		O	**MAT3_0**: PWM output for timer 3, channel 0
PVVOLTSENS EBUCK	32[3]	I	**PVVOLTSENSEBUCK**: PV voltage sense for buck mode
PVVOLTSENS EBOOST	33[3]	I	**PVVOLTSENSEBOOST**: PV voltage sense for boost mode; this pin is not connected when only buck mode is used
PVCURRENTS ENSE	34[3]	I	**PVCURRENT SENSE**: PV current sense
PIO25/AD6	38[3]	I/O	**PIO25**: general purpose input/output digital pin
		I	**AD6**: ADC 0, input 6

Table 59. Pin description *...continued*

Symbol	Pin	Type	Description
PIO26/AD7	39[3]	I/O	**PIO26**: general purpose input/output digital pin
		I	**AD7**: ADC 0, input 7
PIO27/$\overline{\text{TRST}}$	8[1]	I/O	**PIO27**: general purpose input/output digital pin
		I	**$\overline{\text{TRST}}$**: test reset for JTAG interface. If JTAGSEL is HIGH, this pin is automatically configured for use with EmbeddedICE (Debug mode).
PIO28/TMS	9[1]	I/O	**PIO28**: general purpose input/output digital pin
		I	**TMS**: Test Mode Select for JTAG interface. If JTAGSEL is HIGH, this pin is automatically configured for use with EmbeddedICE (Debug mode).
PIO29/TCK	10[1]	I/O	**PIO29**: general purpose input/output digital pin
		I	**TCK**: test clock for JTAG interface. This clock must be slower than $\frac{1}{6}$ of the CPU clock (CCLK) for the JTAG interface to operate. If JTAGSEL is HIGH, this pin is automatically configured for use EmbeddedICE (Debug mode).
PIO30/TDI/ MAT3_3	15[1]	I/O	**PIO30**: general purpose input/output digital pin
		I	**TDI**: test data In for JTAG interface. If JTAGSEL is HIGH, this pin is automatically configured for use with EmbeddedICE (Debug mode).
		O	**MAT3_3**: PWM output 3 for timer 3
PIO31/TDO	16[1]	O	**PIO31**: general-purpose output only digital pin
		O	**TDO**: test data out for JTAG interface. If JTAGSEL is HIGH, this pin is automatically configured for use with EmbeddedICE (Debug mode).
RTCX1	20[7][8]	I	input to RTC oscillator circuit; input voltage must not exceed 1.8 V
RTCX2	25[7][8]	O	output from RTC oscillator circuit
RTCK	26[7]	I/O	**Returned test clock output:** extra signal added to the JTAG port; assists debugger synchronization when processor frequency varies; bidirectional pin with internal pull-up
XTAL1	11	I	input to oscillator circuit and internal clock generator circuits; input voltage must not exceed 1.8 V
XTAL2	12	O	output from the oscillator amplifier
JTAGSEL	27	I	**JTAG select:** when LOW, the part operates normally; when externally pulled HIGH at reset, PIO27 to PIO31 are configured as JTAG port, and the part is in Debug mode[9]; input with internal pull-down
$\overline{\text{RST}}$	6	I	**External reset input:** a LOW on this pin resets device changing I/O ports and peripherals to their default states and processor execution to start at address 0; TTL with hysteresis, 5 V tolerant
GND	7, 19, 43	I	**Ground:** 0 V reference
GNDADC	31	I	**analog ground:** 0 V reference; must be nominally same voltage as GND and isolated to minimize noise and error
$V_{DD(ADC)}$	42	I	**analog 3.3 V power supply:** must be nominally same voltage as $V_{DD(IO)}$ and isolated to minimize noise and error; level on this pin also provides voltage reference level for ADC
V_{DDC}	5	I	**1.8 V core power supply:** power supply voltage for internal circuitry and on-chip PLL
$V_{DD(IO)}$	17, 40	I	**3.3 V pad power supply:** power supply voltage for I/O ports
$V_{DD(RTC)}$	4	I	**RTC power supply:** 3.3 V on this pin supplies power to the RTC

[1] 5 V tolerant (if $V_{DD(IO)}$ and $V_{DD(ADC)} \geq 3.0$ V) pad providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control.

[2] Open-drain 5 V tolerant (if $V_{DD(IO)}$ and $V_{DD(ADC)} \geq 3.0$ V) digital I/O I^2C-bus 400 kHz specification compatible pad. It requires external pull-up to provide an output functionality. Open-drain configuration applies to ALL functions on that pin.

[3] 5 V tolerant (if $V_{DD(IO)}$ and $V_{DD(ADC)} \geq 3.0$ V) pad providing digital I/O (with TTL levels and hysteresis and 10 ns slew rate control) and analog input function. If configured for an input function, this pad utilizes a built-in glitch filter that blocks pulses shorter than 3 ns. When configured as an ADC input, digital section of the pad is disabled.

[4] 5 V tolerant (if $V_{DD(IO)}$ and $V_{DD(ADC)} \geq 3.0$ V) pad providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control. If configured for an input function, this pad utilizes a built-in glitch filter that blocks pulses shorter than 3 ns.

[5] A LOW level during reset on pin PIO14 is considered as an external hardware request to start the ISP command handler.

[6] Open-drain 5 V tolerant (if $V_{DD(IO)}$ and $V_{DD(ADC)} \geq 3.0$ V) digital I/O I^2C-bus 400 kHz specification compatible pad. It requires external pull-up to provide output functionality. Open-drain configuration applies only to I^2C function on that pin.

[7] Pad provides special analog functionality.

[8] For lowest power consumption, leave pins floating when the RTC is not used.

[9] See Section 26.8 “Debug mode” on page 241 for details.

12. Pin connect block

12.1 Features

The pin connect block allows individual pin configuration.

12.2 Applications

The purpose of the pin connect block is to configure the MPT612 pins to the desired functions.

12.3 Description

The pin connect block allows selected pins of the MPT612 to have more than one function. Configuration registers control the multiplexers to allow connection between the pin and the on-chip peripherals.

Connect peripherals to the appropriate pins before activation, and before any related interrupt(s) being enabled. Consider activity of any enabled peripheral function that is not mapped to a related pin as undefined.

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

The only partial exception from the above rule of exclusion is the case of inputs to the ADC. Regardless of the function that is currently selected for the port pin hosting the ADC input, this ADC input can be read at any time and variations of the voltage level on this pin is reflected in the ADC readings. However, valid analog reading(s) can be obtained only if the function of an analog input is selected. Only in this case the proper interface circuit is active between the physical pin and the ADC module. In all other cases, a part of digital logic necessary for the digital function to be performed is active and disrupts proper behavior of the ADC.

12.4 Register description

The pin control module contains 2 registers as shown in Table 60.

Table 60. 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 the content of reserved bits.

12.4.1 Pin function Select register 0 (PINSEL0 - 0xE002 C000)

Register PINSEL0 controls the functions of the pins as per the settings listed in Table 63 on page 66. The direction control bit in register IO0DIR is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically.

Table 61. Pin function select register 0 (PINSEL0 - address 0xE002 C000)

PINSEL0	Pin name	Value	Function	Value after reset
1:0	PIO0	00	GPIO pin 0	0
		01	TXD0 (UART0)	
		10	MAT3.1(Timer 3)	
		11	reserved	
3:2	PIO1	00	GPIO pin 1	0
		01	RXD0 (UART0)	
		10	MAT3.2 (Timer 3)	
		11	reserved	
5:4	PIO2	00	GPIO pin 2	0
		01	SCL0 (I^2C0)	
		11	reserved	
7:6	PIO3	00	GPIO pin 3	0
		01	SDA0 (I^2C0)	
		11	reserved	
9:8	PIO4	00	GPIO pin 4	0
		01	SCK0 (SPI0)	
		11	reserved	
11:10	PIO5	00	GPIO pin 5	0
		01	MISO0 (SPI0)	
		11	reserved	
13:12	PIO6	00	GPIO pin 6	0
		01	MOSI0 (SPI0)	
		11	reserved	
15:14	PIO7	00	GPIO pin 7	0
		01	SSEL0 (SPI0)	
		10	PWMOUT0	
		11	reserved	
17:16	PIO8	00	GPIO pin 8	0
		01	TXD1 (UART1)	
		10	PWMOUT1	
		11	reserved	
19:18	PIO9	00	GPIO pin 9	0
		01	RXD1 (UART1)	
		10	PWMOUT2	
		11	reserved	
21:20	PIO10	00	GPIO pin 10	0
		01	RTS1(UART1)	
		10	CAP1.0 (Timer 1)	
		11	AD3	

Table 61. Pin function select register 0 (PINSEL0 - address 0xE002 C000) *...continued*

PINSEL0	Pin name	Value	Function	Value after reset
23:22	PIO11	00	GPIO pin 11	0
		01	CTS1 (UART1)	
		10	CAP1.1 (Timer 1)	
		11	AD4	
25:24	PIO12	00	GPIO pin 12	0
		01	DSR1 (UART1)	
		10	MAT1.0 (Timer 1)	
		11	AD5	
27:26	PIO13	00	GPIO pin 13	0
		01	reserved	
		10	MAT1.1 (Timer 1)	
		11	DTR1 (UART1)	
29:28	PIO14	00	GPIO pin 14	0
		01	EINT1	
		10	SCK1 (SSP1)	
		11	DCD1 (UART1)	
31:30	PIO15	00	GPIO pin 15	0
		01	EINT2	
		10	reserved	
		11	RI1 (UART1)	

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

Register PINSEL1 controls the functions of the pins as per the settings listed in the following tables. The direction control bit in register IO0DIR is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically.

Table 62. Pin function select register 1 (PINSEL1 - address 0xE002 C004)

PINSEL1	Pin Name	Value	Function	Value after reset
1:0	PIO16	00	GPIO pin 16	0
		01	EINT0	
		11	reserved	
3:2	PIO17	00	GPIO pin 17	0
		01	SCL1 (I^2C1)	
		10	CAP1.2 (Timer 1)	
		11	reserved	
5:4	PIO18	00	GPIO pin 18	0
		01	SDA1 (I^2C1)	
		10	CAP1.3 (Timer 1)	
		11	reserved	

Table 62. Pin function select register 1 (PINSEL1 - address 0xE002 C004) *…continued*

PINSEL1	Pin Name	Value	Function	Value after reset
7:6	PIO19	00	GPIO pin 19	0
		01	MISO1 (SPI1)	
		10	MAT1.2 (Timer 1)	
		11	reserved	
9:8	PIO20	00	GPIO pin 20	0
		01	MOSI1 (SPI1)	
		10	MAT1.3 (Timer 1)	
		11	reserved	
11:10	PIO21	00	GPIO pin 21	0
		01	SSEL1 (SPI1)	
		10	MAT3.0 (Timer 3)	
		11	reserved	
13:12	PIO22	00	GPIO pin 22	0
		01	reserved	
		10	reserved	
		11	PVVOLTSENSEBUCK	
15:14	PIO23	00	GPIO pin 23	0
		01	reserved	
		10	reserved	
		11	PVVOLTSENSEBOOST	
17:16	PIO24	00	GPIO pin 24	0
		01	reserved	
		10	reserved	
		11	PVCURRENTSENSE	
19:18	PIO25	00	GPIO pin 25	0
		01	reserved	
		10	reserved	
		11	AD6	
21:20	PIO26	00	GPIO pin 26	0
		01	reserved	
		10	reserved	
		11	AD7	
23:22	PIO27	00	GPIO pin 27	0
		01	TRST (JTAG)	
		11	reserved	
25:24	PIO28	00	GPIO pin 28	0
		01	TMS (JTAG)	
		11	reserved	
27:26	PIO29	00	GPIO pin 29	0
		01	TCK (JTAG)	
		11	reserved	

Table 62. Pin function select register 1 (PINSEL1 - address 0xE002 C004) *...continued*

PINSEL1	Pin Name	Value	Function	Value after reset
29:28	PIO30	00	GPIO pin 30	0
		01	TDI (JTAG)	
		10	MAT3.3 (Timer 3)	
		11	reserved	
31:30	PIO31	00	GPIO pin 31	0
		01	TDO (JTAG)	
		10	reserved	
		11	reserved	

12.4.3 Pin function select register values

The PINSEL registers control the functions of device pins as shown Table 63. Pairs of bits in these registers correspond to specific device pins.

Table 63. 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 register IO0DIR 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 can be found in the appropriate data sheet.

13. General-Purpose Input/Output (GPIO) ports

13.1 Features

- Every physical GPIO pin is accessible through two independent sets of registers. One set provides enhanced features and higher speed GPIO pin access. The other register set provides slow speed GPIO pin access.
- Enhanced GPIO functions:
  - GPIO registers are relocated to the ARM local bus to achieve the fastest possible I/O timing
  - Mask registers allow sets of port bits to be treated as a group, leaving other bits unchanged
  - All registers are byte and half-word addressable
  - An entire port value can be written in one instruction
- Bit-level set and clear registers allow a single instruction set, or clear any number of bits

- Individual bits can be direction controlled
- All I/O pins default to inputs after reset

13.2 Applications

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

13.3 Pin description

Table 64. GPIO pin description

Pin	Type	Description
PIO0 to PIO31	input/ output	general-purpose input/output; the number of GPIOs available depends on the use of alternate functions

13.4 Register description

MPT612 has one 32-bit General-Purpose I/O port. A total of 32 input/output pins are available on PORT0 which is controlled by registers shown in Table 65 and Table 66.

Slow speed GPIO pin registers are shown in Table 65.

The registers in Table 66 represent the enhanced GPIO features available on the MPT612. 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 register System Control and Status flags (SCS) whether a GPIO will be accessed via registers that provide enhanced features or the set of slow speed GPIO registers (see Section 10.6.1 “System control and status flags register (SCS - 0xE01F C1A0)” on page 42). While both fast and slow speed GPIO registers of a port 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 is not observable via the corresponding slow speed GPIO register.

The following text refers to slow speed GPIO as slow GPIO, while the GPIO with enhanced features selected is referred to as the fast GPIO.

Slow GPIO registers are word accessible only. Fast GPIO registers are byte, half-word, and word accessible. In Table 65 and Table 66, bit 0 corresponds to PIO0, and bit 31 corresponds to PIO31.

Table 65. GPIO register map (slow speed GPIO APB accessible registers)

Generic name	Description	Access	Reset value[1]	PORT0 address and name
IOPIN	GPIO pin value register. Current state of GPIO configured pins can always be read from this register, regardless of pin direction.	R/W	n/a	0xE002 8000 IO0PIN
IOSET	GPIO output set register. Controls state of output pins with register IOCLR. Writing logic 1s produces HIGHs at the corresponding port pins. Writing logic 0s has no effect.	R/W	0x0000 0000	0xE002 8004 IO0SET
IODIR	GPIO direction control register. Individually controls the direction of each port pin.	R/W	0x0000 0000	0xE002 8008 IO0DIR
IOCLR	GPIO output clear register. Controls state of output pins. Writing logic 1s produces LOWs at corresponding pins and clears corresponding bits in register IOSET. Writing logic 0s has no effect.	WO	0x0000 0000	0xE002 800C IO0CLR

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

Table 66. GPIO register map (local bus accessible registers - enhanced GPIO features)

Generic name	Description	Access	Reset value[1]	PORT0 address and name
FIODIR	fast GPIO direction control register. Individually controls direction of each pin.	R/W	0x0000 0000	0x3FFF C000 FIO0DIR
FIOMASK	fast mask register for pins. Writes, sets, clears, and reads pins (via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN). Only bits enabled by logic 0s in this register are altered or returned. **Remark:** Bits in register FIOMASK are active LOW.	R/W	0x0000 0000	0x3FFF C010 FIO0MASK
FIOPIN	fast GPIO pin value register using FIOMASK. Current state of digital pins can be read from this register, regardless of pin direction or alternate function selection (when pins are not configured as an input to ADC). Value read is masked by ANDing with FIOMASK. Writing to this register puts corresponding values in all bits enabled by logic 0s in FIOMASK.	R/W	0x0000 0000	0x3FFF C014 FIO0PIN
FIOSET	fast GPIO output set register using FIOMASK. Controls state of output pins. Writing logic 1s produces HIGHs at corresponding pins. Writing logic 0s has no effect. Reading this register returns current content of port output register. Only bits enabled by logic 0s in FIOMASK can be altered.	R/W	0x0000 0000	0x3FFF C018 FIO0SET
FIOCLR	fast GPIO output clear register using FIOMASK. Controls state of output pins. Writing logic 1s produces LOWs at corresponding pins. Writing logic 0s has no effect. Only bits enabled by logic 0s 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 the content of reserved bits.

13.4.1 GPIO Direction register (IODIR, IO0DIR - 0xE002 8008; FIODIR, FIO0DIR - 0x3FFF C000)

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

IO0DIR is the slow speed GPIO register, while the enhanced GPIO functions are supported via the register FIO0DIR.

Table 67. GPIO 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 PIO0 ... bit 30 controls PIO30.	0x0000 0000
		0	controlled pin is input	
		1	controlled pin is output	

Table 68. Fast GPIO 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 PIO0 ... Bit 30 in register FIO0DIR controls PIO30.	0x0000 0000
		0	controlled pin is input	
		1	controlled pin is output	

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

Table 69. Fast GPIO Direction control byte and half-word accessible register description

Register name	Register length (bits) and access	Address	Description	Reset value
FIO0DIR0	8 (byte)	0x3FFF C000	fast GPIO direction control register 0. Bit 0 in register FIO0DIR0 corresponds to PIO0 ... bit 7 to PIO7.	0x00
FIO0DIR1	8 (byte)	0x3FFF C001	fast GPIO direction control register 1. Bit 0 in register FIO0DIR1 corresponds to PIO8 ... bit 7 to PIO15.	0x00
FIO0DIR2	8 (byte)	0x3FFF C002	fast GPIO direction control register 2. Bit 0 in register FIO0DIR2 corresponds to PIO16 ... bit 7 to PIO23.	0x00
FIO0DIR3	8 (byte)	0x3FFF C003	fast GPIO direction control register 3. Bit 0 in register FIO0DIR3 corresponds to PIO24 ... bit 7 to PIO31.	0x00
FIO0DIRL	16 (half-word)	0x3FFF C000	fast GPIO direction control lower half-word register. Bit 0 in register FIO0DIRL corresponds to PIO0 ... bit 15 to PIO15.	0x0000
FIO0DIRU	16 (half-word)	0x3FFF C002	fast GPIO direction control upper half-word register. Bit 0 in register FIO0DIRU corresponds to PIO16 ... bit 15 to PIO31.	0x0000

13.4.2 Fast GPIO mask register (FIOMASK, FIO0MASK - 0x3FFF C010)

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

A bit set to logic 0 in this register enables an access to the corresponding physical pin via a read or write access. If a bit in this register is logic 1, the corresponding pin is not changed with write access and if read, is not reflected in the updated register FIOPIN. For software examples, see Section 13.5 "GPIO usage notes" on page 73.

Table 70. Fast GPIO 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 registers FIOSET, FIOCLR, and FIOPIN. Current state of pin is observable in register FIOPIN.	
		1	physical pin is unaffected by writes to registers FIOSET, FIOCLR and FIOPIN. When register FIOPIN is read, this bit is not updated with state of physical pin	

Aside from the 32-bit long and word-only accessible register FIOMASK, every fast GPIO pin can also be controlled via several byte and half-word accessible registers listed in Table 71. Next to providing the same functions as register FIOMASK, these additional registers allow easier and faster access to the physical pins.

Table 71. Fast GPIO mask byte and half-word accessible register description

Register name	Register length (bits) and access	Address	Description	Reset value
FIO0MASK0	8 (byte)	0x3FFF C010	fast GPIO mask register 0. Bit 0 in register FIO0MASK0 corresponds to PIO0 ... bit 7 to PIO7.	0x00
FIO0MASK1	8 (byte)	0x3FFF C011	fast GPIO mask register 1. Bit 0 in register FIO0MASK1 corresponds to PIO8 ... bit 7 to PIO15.	0x00
FIO0MASK2	8 (byte)	0x3FFF C012	fast GPIO mask register 2. Bit 0 in register FIO0MASK2 corresponds to PIO16 ... bit 7 to PIO23.	0x00
FIO0MASK3	8 (byte)	0x3FFF C013	fast GPIO mask register 3. Bit 0 in register FIO0MASK3 corresponds to PIO24 ... bit 7 to PIO31.	0x00
FIO0MASKL	16 (half-word)	0x3FFF C001	fast GPIO mask lower half-word register. Bit 0 in register FIO0MASKL corresponds to PIO0 ... bit 15 to PIO15.	0x0000
FIO0MASKU	16 (half-word)	0x3FFF C012	fast GPIO mask upper half-word register. Bit 0 in register FIO0MASKU corresponds to PIO16 ... bit 15 to PIO31.	0x0000

13.4.3 GPIO Pin value register (IOPIN, IO0PIN - 0xE002 8000; FIOPIN, FIO0PIN - 0x3FFF C014)

This register provides the values of pins that are configured to perform only digital functions. The register gives 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 pin can have GPIO input or GPIO output, UART receive, and PWM output as selectable functions. Any configuration of that pin allows its current logic state to be read from register IOPIN.

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 register IOPIN is not valid.

Writing to register IOPIN stores the value in the pin output register, bypassing the need to use both registers IOSET and IOCLR to obtain the entire written value. Use this feature carefully in an application since it affects all pins.

The slow speed GPIO register is the IO0PIN, while the enhanced GPIOs are supported via register FIO0PIN. Access to pins via register FIOPIN is conditioned by the corresponding register FIOMASK (see Section 13.4.2 on page 69).

Only pins masked with logic 0s in the mask register (see Section 13.4.2 on page 69) are correlated to the current content of the Fast GPIO pin value register.

Table 72. GPIO 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 PIO0 ... Bit 31 in IO0PIN corresponds to PIO31.	n/a

Table 73. Fast GPIO 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 PIO0 ... Bit 31 in FIO0PIN corresponds to PIO31.	n/a

Aside from the 32-bit long and word-only accessible register FIOPIN, every fast GPIO can also be controlled via several byte and half-word accessible registers listed in Table 74. Next to providing the same functions as register FIOPIN, these additional registers allow easier and faster access to the physical pins.

Table 74. Fast GPIO pin value byte and half-word accessible register description

Register name	Register length (bits) and access	Address	Description	Reset value
FIO0PIN0	8 (byte)	0x3FFF C014	fast GPIO pin value register 0. Bit 0 in register FIO0PIN0 corresponds to PIO0 ... bit 7 to PIO7.	0x00
FIO0PIN1	8 (byte)	0x3FFF C015	fast GPIO pin value register 1. Bit 0 in register FIO0PIN1 corresponds to PIO8 ... bit 7 to PIO15.	0x00
FIO0PIN2	8 (byte)	0x3FFF C016	fast GPIO pin value register 2. Bit 0 in register FIO0PIN2 corresponds to PIO16 ... bit 7 to PIO23.	0x00
FIO0PIN3	8 (byte)	0x3FFF C017	fast GPIO pin value register 3. Bit 0 in register FIO0PIN3 corresponds to PIO24 ... bit 7 to PIO31.	0x00
FIO0PINL	16 (half-word)	0x3FFF C014	fast GPIO pin value lower half-word register. Bit 0 in register FIO0PINL corresponds to PIO0 ... bit 15 to PIO15.	0x0000
FIO0PINU	16 (half-word)	0x3FFF C016	fast GPIO pin value upper half-word register. Bit 0 in register FIO0PINU corresponds to PIO16 ... bit 15 to PIO31.	0x0000

13.4.4 GPIO output set register (IOSET, IO0SET - 0xE002 8004; FIOSET, FIO0SET - 0x3FFF C018)

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

Reading register IOSET 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 slow speed GPIO register while the enhanced GPIOs are supported via register FIO0SET. Access to pins via register FIOSET is conditioned by the corresponding register FIOMASK (see Section 13.4.2 on page 69).

Table 75. GPIO 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 PIO0 ... Bit 31 in IO0SET corresponds to PIO31.	0x0000 0000

Table 76. Fast GPIO 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 PIO0 ... Bit 31 in FIO0SET corresponds to PIO31.	0x0000 0000

Aside from the 32-bit long and word-only accessible register FIOSET, every fast GPIO can also be controlled via several byte and half-word accessible registers listed in Table 77. Next to providing the same functions as register FIOSET, these additional registers allow easier and faster access to the physical pins.

Table 77. Fast GPIO port 0 output set byte and half-word accessible register description

Register name	Register length (bits) and access	Address	Description	Reset value
FIO0SET0	8 (byte)	0x3FFF C018	fast GPIO output set register 0. Bit 0 in register FIO0SET0 corresponds to PIO0 ... bit 7 to PIO7.	0x00
FIO0SET1	8 (byte)	0x3FFF C019	fast GPIO output set register 1. Bit 0 in register FIO0SET1 corresponds to PIO8 ... bit 7 to PIO15.	0x00
FIO0SET2	8 (byte)	0x3FFF C01A	fast GPIO output set register 2. Bit 0 in register FIO0SET2 corresponds to PIO16 ... bit 7 to PIO23.	0x00
FIO0SET3	8 (byte)	0x3FFF C01B	fast GPIO output set register 3. Bit 0 in register FIO0SET3 corresponds to P0in24 ... bit 7 to PIO31.	0x00
FIO0SETL	16 (half-word)	0x3FFF C018	fast GPIO output set lower half-word register. Bit 0 in register FIO0SETL corresponds to PIO0 ... bit 15 to PIO15.	0x0000
FIO0SETU	16 (half-word)	0x3FFF C01A	fast GPIO output set upper half-word register. Bit 0 in register FIO0SETU corresponds to PIO16 ... bit 15 to PIO31.	0x0000

13.4.5 GPIO output clear register (IOCLR, IO0CLR - 0xE002 800C; FIOCLR, FIO0CLR - 0x3FFF C01C)

This register is used to produce a LOW level output at pins configured as GPIO in an OUTPUT mode. Writing logic 1 produces a LOW level at the corresponding pin and clears the corresponding bit in register IOSET. Writing logic 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 slow speed GPIO register while the enhanced GPIOs are supported via register FIO0CLR. Access to pins via register FIOCLR is conditioned by the corresponding register FIOMASK (see Section 13.4.2 on page 69).

Table 78. GPIO 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 PIO0 ... Bit 31 in IO0CLR corresponds to PIO31.	0x0000 0000

Table 79. Fast GPIO 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 PIO0 ... Bit 31 in FIO0CLR corresponds to PIO31.	0x0000 0000

Aside from the 32-bit long and word-only accessible register FIOCLR, every fast GPIO pin can also be controlled via several byte and half-word accessible registers listed in Table 80. Next to providing the same functions as register FIOCLR, these additional registers allow easier and faster access to the physical pins.

Table 80. Fast GPIO output clear byte and half-word accessible register description

Register name	Register length (bits) and access	Address	Description	Reset value
FIO0CLR0	8 (byte)	0x3FFF C01C	fast GPIO output clear register 0. Bit 0 in register FIO0CLR0 corresponds to PIO0 ... bit 7 to PIO7.	0x00
FIO0CLR1	8 (byte)	0x3FFF C01D	fast GPIO output clear register 1. Bit 0 in register FIO0CLR1 corresponds to PIO8 ... bit 7 to PIO15.	0x00
FIO0CLR2	8 (byte)	0x3FFF C01E	fast GPIO output clear register 2. Bit 0 in register FIO0CLR2 corresponds to PIO16 ... bit 7 to PIO23.	0x00
FIO0CLR3	8 (byte)	0x3FFF C01F	fast GPIO output clear register 3. Bit 0 in register FIO0CLR3 corresponds to PIO24 ... bit 7 to PIO31.	0x00
FIO0CLRL	16 (half-word)	0x3FFF C01C	fast GPIO output clear lower half-word register. Bit 0 in register FIO0CLRL corresponds to PIO0 ... bit 15 to PIO15.	0x0000
FIO0CLRU	16 (half-word)	0x3FFF C01E	fast GPIO output clear upper half-word register. Bit 0 in register FIO0SETU corresponds to PIO16 ... bit 15 to PIO31.	0x0000

13.5 GPIO usage notes

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

The state of the output configured GPIO pin is determined by writes into the pin's port IOSET and IOCLR registers. The last of these accesses to register IOSET/IOCLR determines the final output of a pin.

In the case of a code:

```
IO0DIR = 0x0000 0080 ;pin PIO7 configured as output
IO0CLR = 0x0000 0080 ;PIO7 goes LOW
IO0SET = 0x0000 0080 ;PIO7 goes HIGH
IO0CLR = 0x0000 0080 ;PIO7 goes LOW
```

Pin PIO7 is configured as an output (write to register IO0DIR). PIO7 output is then set LOW (first write to register IO0CLR). Short high pulse follows on PIO7 (write access to IO0SET), and the final write to register IO0CLR sets pin PIO7 back to LOW level.

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

A write access to pins IOSET followed by a write to register IOCLR results in pins outputting 0s slightly later than 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 pins register IOPIN.

The following code preserves existing output on pins PIO[31:16] and PIO[7:0] and at the same time sets PIO[15:8] to 0xA5, regardless of the previous value of pins PIO[15:8]:

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

The same outcome can be obtained using the fast pin 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;
```

13.5.3 Writing to IOSET/IOCLR vs. IOPIN

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

A write to register IOPIN enables instantaneous output of a desired content on the parallel GPIO. Binary data written into register IOPIN affects all output configured pins of that parallel port: 0s in IOPIN produces LOW level pin outputs, and 1s in IOPIN produces HIGH level pin outputs. In order to change the output of only a group of pins, the application must logically AND readout from the IOPIN with the 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 register IOPIN. Example 2 above illustrates the output of 0xA5 on pins 15 to 8 while preserving all other output pins as they were before.

13.5.4 Output signal frequency considerations when using slow speed GPIO and enhanced GPIO registers

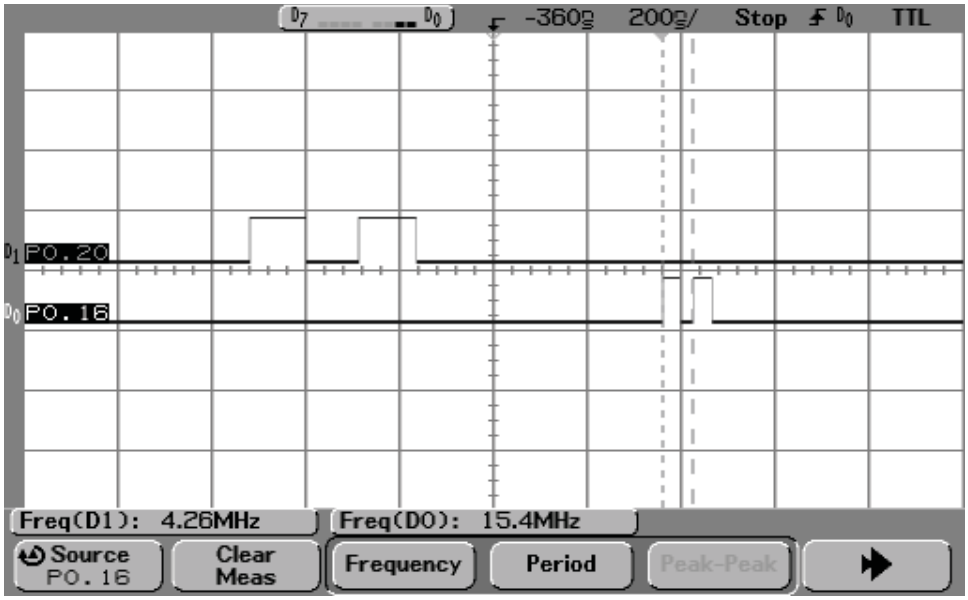
The enhanced features of the fast GPIO pins available on this MPT612 make GPIO pins more responsive to code that has the task of controlling them. In particular, software access to a GPIO pin is 3.5 times faster via the fast GPIO registers than it is when a set of slow speed registers is used. As a result of the increased access speed, the maximum output frequency of the digital pin is also increased 3.5 times. This large increase in output

frequency is not always obvious 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 pin output in assembly code and execute in ARM mode.

The following example shows code in which the pin control section is written in assembly language for ARM. First, port 0 is configured as a slow port, and the program generates two pulses on PIO20. Then port 0 is configured as a fast port, and two pulses are generated on PIO16. This illustrates the difference between the fast and slow GPIO port output capabilities. Once this code is compiled in ARM mode, its execution from the on-chip Flash yields the best results when the MAM module is configured as described in Section 8.8 “MAM usage notes” on page 18. Execution from the on-chip SRAM is independent of 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 PIO20*/
     ldr  r0,=0xe0028004  /*register address--IOSET*/
     ldr  r1,=0xe002800C  /*register address--IOCLR*/
          /*generate 2 pulses using slow GPIO on PIO0*/
     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,=0x00010000  /*select fast port 0.16 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 17 illustrates the above code executed by the MPT612 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).



aaa-000580

Fig 17. Illustration of the fast and slow GPIO access and output showing a 3.5× increase of the pin output frequency

14. Universal Asynchronous Receiver/Transmitter 0 (UART0)

14.1 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

14.2 Pin description

Table 81: UART0 pin description

Pin	Type	Description
RXD0	input	**serial input**: serial receive data
TXD0	output	**serial output**: serial transmit data

14.3 Register description

UART0 contains registers organized as shown in Table 82. The Divisor Latch Access Bit (DLAB) is contained in U0LCR[7] and enables access to the divisor latches.

Table 82: 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	n/a	0xE000 C000 (DLAB=0)
U0THR	transmit holding register	8-bit write data								WO	n/a	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	-	-	-	-	-	-	ABTO IntEn	ABEO IntEn	R/W	0x00	0xE000 C004 (DLAB=0)
		-	-	-	-	-	RX Line Status Interrupt Enable	THRE Interrupt Enable	RBR Interrupt Enable			
U0IIR	interrupt ID register	-	-	-	-	-	-	ABTOInt	ABEOInt	RO	0x01	0xE000 C008
		FIFO Enable		-	-	Interrupt Identification			Interrupt Pending			
U0FCR	FIFO control register	RX Trigger Level		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE000 C008
U0LCR	line control register	DLAB	Break Control	Parity Select		Parity Enable	Stop Bit Select	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	line status register	RXFE	TEMT	THRE	BI	FE	PE	OE	RDR	RO	0x60	0xE000 C014
U0SCR	scratch pad register	8-bit data								R/W	0x00	0xE000 C01C
U0ACR	auto-baud control register	-	-	-	-	-	-	ABTO IntClr	ABEO IntClr	R/W	0x00	0xE000 C020
		-	-	-	-	-	Auto Restart	Mode	Start			
U0FDR	fractional divider register	Reserved[31:8]									0x10	0xE000 C028
		MULVAL					DIVADDVAL					
U0TER	Tx enable register	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE000 C030

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

14.3.1 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.

In order to access the U0RBR, the Divisor Latch Access Bit (DLAB) in U0LCR must be logic 0. The U0RBR is always read only.

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

Table 83: UART0 Receiver buffer register (U0RBR - address 0xE000 C000, when DLAB = 0, read only) bit description

Bit	Symbol	Description	Reset value
7:0	RBR	UART0 receiver buffer register contains the oldest received byte in the UART0 Rx FIFO	undefined

14.3.2 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.

In order to access the U0THR, the Divisor Latch Access Bit (DLAB) in U0LCR must be logic 0. The U0THR is always write only.

Table 84: 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 UART0 transmit holding register stores data in UART0 transmit FIFO. Byte is sent when it reaches bottom of FIFO and when transmitter is available.	n/a

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

The UART0 Divisor Latch is part of the UART0 Fractional Baud Rate Generator and holds the value used to divide the clock supplied by the fractional prescaler in order to produce the baud rate clock, which must be 16x the desired baud rate (Equation 1 on page 79). Registers U0DLL and U0DLM 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. In order to access the UART0 divisor latches, the Divisor Latch Access Bit (DLAB) in U0LCR must be logic 1.

Details on how to select the right value for U0DLL and U0DLM can be found later in this user manual.

Table 85: UART0 Divisor latch LSB register (U0DLL - address 0xE000 C000, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLL	UART0 divisor latch LSB register and U0DLM register determine baud rate of UART0	0x01

Table 86: UART0 Divisor latch MSB register (U0DLM - address 0xE000 C004, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLM	UART0 divisor latch MSB register and U0DLL register determine baud rate of UART0	0x00

14.3.4 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.

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

Table 87: 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 logic 0, fractional baud rate generator will not affect UARTn baud rate.	0
7:4	MULVAL	1	baud rate pre-scaler multiplier value. Must be greater than or equal to 1 for UARTn to operate properly, regardless of whether fractional baud rate generator is used or not.	1
31:8	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	0

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

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

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

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

The value of MULVAL and DIVADDVAL must comply with the following conditions:

- $0 < MULVAL \leq 15$
- $0 \leq DIVADDVAL < 15$

- DIVADDVAL< MULVAL

The value of U0FDR must not be modified while transmitting/receiving data or data can be lost or corrupted.

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

14.3.4.1 Baud rate calculation

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

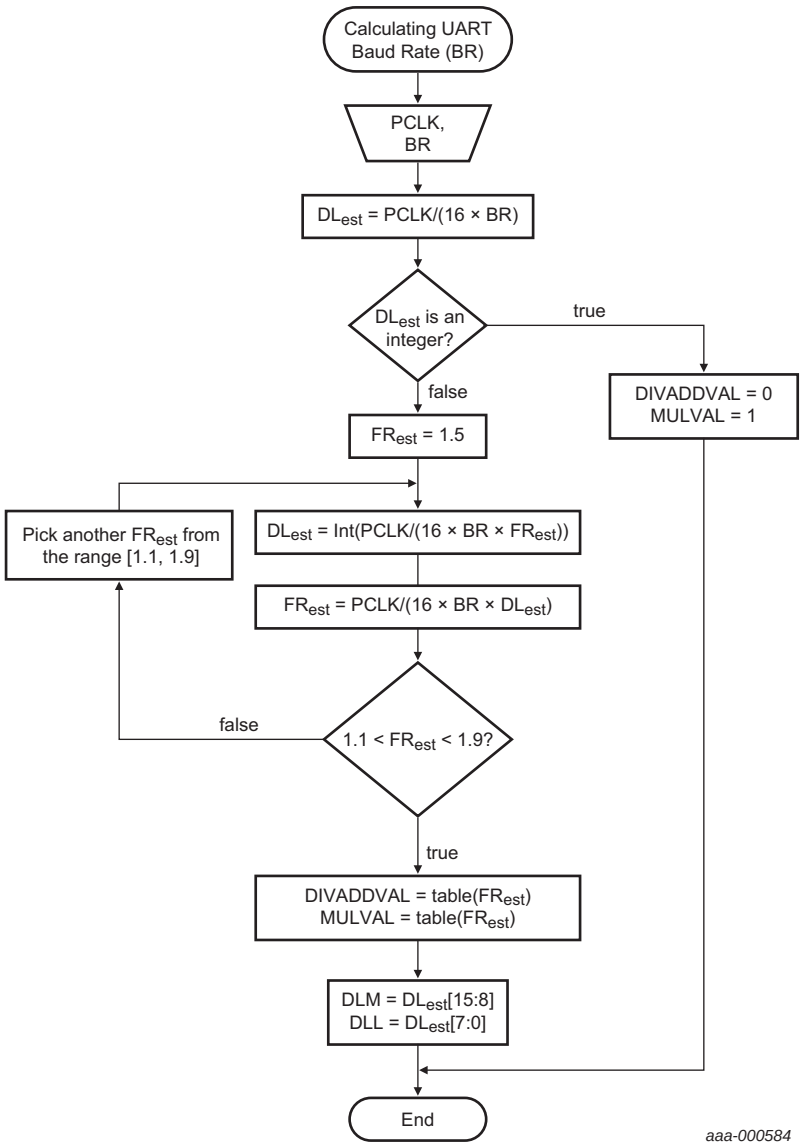


Fig 18. Algorithm for setting UART dividers

Table 88. 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

Table 88. Fractional divider setting look-up table *...continued*

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

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

Example 2: PCLK = 12 MHz, BR = 115200: according to the provided algorithm DL_{est} = PCLK / (16 × BR) = 12 MHz / (16 × 115200) = 6.51. This DL_{est} is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of FR_{est} = 1.5 a new DL_{est} = 4 is calculated and FR_{est} is recalculated as FR_{est} = 1.628. Since FR_{est} = 1.628 is within the specified range of 1.1 and 1.9, DIVADDVAL and MULVAL values can be obtained from the attached look-up table.

The closest value for FR_{est} = 1.628 in the look-up Table 88 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 2 on page 89, the UART baud rate is 115384 Bd. This rate has a relative error of 0.16 % from the originally specified value of 115200 Bd.

14.3.5 UART0 Interrupt enable register (U0IER - 0xE000 C004, when DLAB = 0)

The U0IER is used to enable UART0 interrupt sources.

Table 89. 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 receive data available interrupt for UART0. Also controls the character receive time-out interrupt.	0
		0	disables RDA interrupts	
		1	enables RDA interrupts	
1	THRE Interrupt Enable		U0IER[1] enables THRE interrupt for UART0. Status can be read from U0LSR[5].	0
		0	disables THRE interrupts	
		1	enables THRE interrupts	

Table 89. UART0 Interrupt enable register (U0IER - address 0xE000 C004, when DLAB = 0) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
2	RX Line Status Interrupt Enable		U0IER[2] enables the UART0 Rx line status interrupts. Status of interrupt can be read from U0LSR[4:1].	0
		0	disables Rx line status interrupts	
		1	enables Rx line status interrupts	
7:3	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
8	ABEOIntEn		enables end of auto-baud interrupt	0
		0	disables end of auto-baud interrupt	
		1	enables end of auto-baud interrupt	
9	ABTOIntEn		enables the auto-baud time-out interrupt.	0
		0	disables auto-baud time-out interrupt	
		1	enables auto-baud time-out interrupt	
31:10	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

14.3.6 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 90: 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. Pending interrupt can be determined by evaluating U0IIR[3:1].	1
		0	at least one interrupt is pending	
		1	no pending interrupts	
3:1	Interrupt Identification		U0IER[3:1] identifies an interrupt corresponding to 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7:6	FIFO Enable		bits are equivalent to U0FCR[0]	0

Table 90: UART0 Interrupt identification register (U0IIR - address 0xE000 C008, read only) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Interrupts are handled as described in Table 91. 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. In order to clear the interrupt before exiting the Interrupt Service Routine, the U0IIR must be read.

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 sets the interrupt can be observed via U0LSR[4:1]. The interrupt is cleared after 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) clears the interrupt. This interrupt is intended to flush the UART0 RBR after a message is 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 91: 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]

Table 91: UART0 interrupt handling *…continued*

U0IIR[3:0] value[1]	Priority	Interrupt type	Interrupt source	Interrupt reset
0100	second	Rx data available	Rx data available or trigger level reached in FIFO (U0FCR0 = 1)	U0RBR read[3] or UART0 FIFO drops below trigger level
1100	second	character time-out indication	minimum of one character in Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and trigger level setting (3.5 to 4.5 character times). exact time is: [(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 14.3.9 on page 87

[3] For details see Section 14.3.1 on page 78

[4] For details see Section 14.3.6 on page 83 and Section 14.3.2 on page 78

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. If the UART0 THR FIFO has held two or more characters at one time and currently, the U0THR is empty, a THRE interrupt is set immediately. 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).

14.3.7 UART0 FIFO Control register (U0FCR - 0xE000 C008)

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

Table 92. UART0 FIFO Control register (U0FCR - address 0xE000 C008) bit description

Bit	Symbol	Value	Description	Reset value
0	FIFO Enable	0	UART0 FIFOs disabled. Must not be used in application.	0
		1	active HIGH enable for both UART0 Rx and Tx FIFOs and U0FCR[7:1] access. Must be set for correct UART0 operation. Any transition on this bit automatically clears UART0 FIFOs.	
1	RX FIFO Reset	0	no impact on either UART0 FIFO	0
		1	writing a logic 1 to U0FCR[1] clears all bytes in UART0 Rx FIFO and resets pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	no impact on either UART0 FIFO	0
		1	writing a logic 1 to U0FCR[2] clears all bytes in UART0 Tx FIFO and resets pointer logic. This bit is self-clearing.	
5:3	-	0	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7:6	RX Trigger Level		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)	

14.3.8 UART0 Line control register (U0LCR - 0xE000 C00C)

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

Table 93: 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 transmitted character and attached parity bit is odd.	0
		01	even parity. Number of 1s in transmitted character and attached parity bit is 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	

14.3.9 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 94: UART0 Line status register (U0LSR - address 0xE000 C014, read only) bit description

Bit	Symbol	Value	Description	Reset value
0	Receiver Data Ready (RDR)		U0LSR0 set when U0RBR holds an unread character and is cleared when UART0 RBR FIFO is empty	0
		0	U0RBR is empty	
		1	U0RBR contains valid data	
1	Overrun Error (OE)		overrun error condition set when overrun occurs. An U0LSR read clears U0LSR1. U0LSR1 is set when UART0 RSR has a new character assembled and UART0 RBR FIFO is full. In this case, UART0 RBR FIFO is not overwritten and character in UART0 RSR is lost.	0
		0	overrun error status is inactive	
		1	overrun error status is active	
2	Parity Error (PE)		if parity bit of received character is wrong state, a parity error occurs. An U0LSR read clears U0LSR[2]. Time of parity error detection is dependent on U0FCR[0]. **Remark:** Parity error is associated with character at top of UART0 RBR FIFO.	0
		0	parity error status is inactive	
		1	parity error status is active	
3	Framing Error (FE)		if stop bit of received character is logic 0, a framing error occurs. An U0LSR read clears U0LSR[3]. Time of framing error detection is dependent on U0FCR0. Upon detection of a framing error, Rx attempts to resynchronize to data and assumes bad stop bit is early start bit. However, it cannot be assumed that next received byte is correct even if there is no framing error. **Remark:** Framing error is associated with character at top of UART0 RBR FIFO.	0
		0	framing error status is inactive	
		1	framing error status is active	
4	Break Interrupt (BI)		if RXD0 is held in spacing state (all 0s) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once break condition is detected, receiver goes idle until RXD0 goes to marking state (all 1s). A U0LSR read clears this status bit. Time of break detection is dependent on U0FCR[0]. **Remark:** Break interrupt is associated with character at top of 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	
6	Transmitter Empty (TEMT)		TEMT is set when both U0THR and U0TSR are empty; TEMT is cleared when either U0TSR or U0THR contain valid data	1
		0	U0THR and/or the U0TSR contains valid data	
		1	U0THR and the U0TSR are empty	

Table 94: UART0 Line status register (U0LSR - address 0xE000 C014, read only) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
7	Error in RX FIFO (RXFE)		U0LSR[7] is set when a character with Rx error such as framing error, parity error or break interrupt, is loaded into U0RBR. This bit is cleared when register U0LSR is read and there are no subsequent errors in UART0 FIFO.	0
		0	U0RBR contains no UART0 Rx errors or U0FCR[0] = 0	
		1	UART0 RBR contains at least one UART0 Rx error	

14.3.10 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 the 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 95: UART0 Scratch pad register (U0SCR - address 0xE000 C01C) bit description

Bit	Symbol	Description	Reset value
7:0	Pad	read/write byte	0x00

14.3.11 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 the user's discretion.

Table 96. Auto-baud control register (U0ACR - 0xE000 C020) bit description

Bit	Symbol	Value	Description	Reset value
0	Start		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. 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	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	0
8	ABEOIntClr		end of auto-baud interrupt clear bit (write only accessible). Writing a logic 1 clears corresponding interrupt in U0IIR. Writing a logic 0 has no impact.	0
9	ABTOIntClr		auto-baud time-out interrupt clear bit (write only accessible). Writing a logic 1 clears corresponding interrupt in U0IIR. Writing a logic 0 has no impact.	0
31:10	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	0

14.3.12 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 measures the bit time of the receive data stream and sets the divisor latch registers U0DLM and U0DLL accordingly.

Auto-baud is started by setting bit U0ACR Start. Auto-baud can be stopped by clearing bit U0ACR Start. Bit Start clears once auto-baud has finished, and reading the bit returns the status of auto-baud (pending/finished).

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

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

The auto-baud function can generate two interrupts:

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

The auto-baud interrupts must 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 pin UART0 Rx baud rate, but the value of register U0FDR is not going to be modified after rate measurement. Also, when auto-baud is used, any write to registers U0DLM and U0DLL must be done before register U0ACR write. The minimum and the maximum baud rates supported by UART0 are function of PCLK, number of data bits, stop-bits and parity bits.

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

14.3.13 UART0 Transmit enable register (U0TER - 0xE000 C030)

MPT612’s U0TER enables implementation of software flow control. When TXEn = 1, the UART0 transmitter sends data as long as it is available. When TXEn = 0, UART0 transmission stops.

Table 97 describes how to use bit TXEn to achieve software flow control.

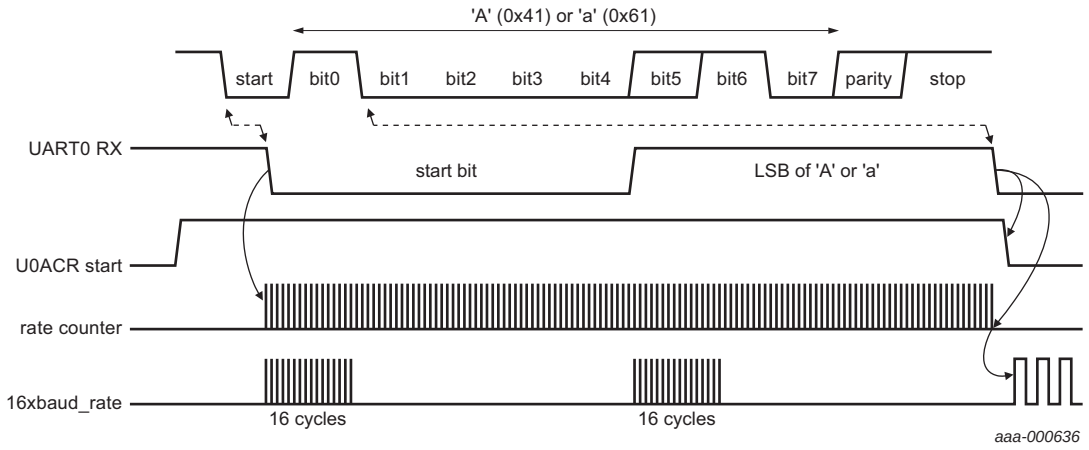
Table 97: UART0 Transmit enable register (U0TER - address 0xE000 C030) bit description

Bit	Symbol	Description	Reset value
6:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7	TXEN	if logic 1, as it is after a reset, data written to THR is output on pin TXD when any preceding data is sent. If cleared to logic 0 while a character is being sent, transmission of that character is completed, no further characters are sent until this bit is set again. In other words, if bit is logic 0, it blocks transfer of characters from THR or Tx FIFO into 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

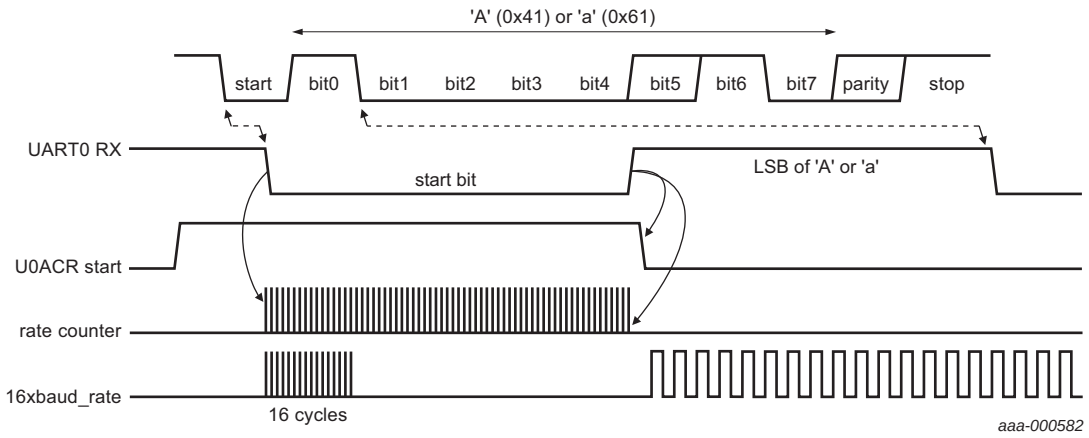
14.3.14 Auto-baud modes

When the software is expecting an AT command, it configures the UART0 with the expected character format and sets bit U0ACR Start. The initial values in the divisor latches U0DLM and U0DLM are set to do not care. Because of the ”A" or ”a" ASCII coding (”A" = 0x41, ”a" = 0x61), pin UART0 Rx sensed start bit and the LSB of the expected character are delimited by two falling edges. When bit U0ACR Start is set, the auto-baud protocol executes the following phases:

1. On setting bit U0ACR Start, the baud-rate measurement counter is reset and the UART0 U0RSR is reset. The U0RSR baud rate is switched to the highest rate.
2. A falling edge on pin UART0 Rx triggers the beginning of the start bit. The rate measuring counter starts counting PCLK cycles optionally pre-scaled by the fractional baud rate generator.
3. During receipt of the start bit, 16 pulses are generated on baud input RSR at the frequency of the (fractional baud rate pre-scaled) UART0 input clock, guaranteeing the start bit is stored in U0RSR.
4. During the receipt of the start bit (and the character LSB for mode = 0), the rate counter continues incrementing with the pre-scaled UART0 input clock (PCLK).
5. If Mode = 0, then the rate counter stops on the next falling edge of pin UART0 Rx. If Mode = 1, then the rate counter stops on the next rising edge of pin UART0 Rx.
6. The rate counter is loaded into U0DLM/U0DLL and the baud rate is switched to normal operation. After setting the U0DLM/U0DLL, the end of auto-baud interrupt U0IIR ABEOInt is set, if enabled. The U0RSR continues 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 19. Autobaud mode 0 and mode 1 waveforms

14.4 Architecture

The architecture of UART0 is shown in block diagram Figure 20.

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 registers U0DLL and U0DLM and is a 16× oversample clock, NBAUDOUT.

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

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

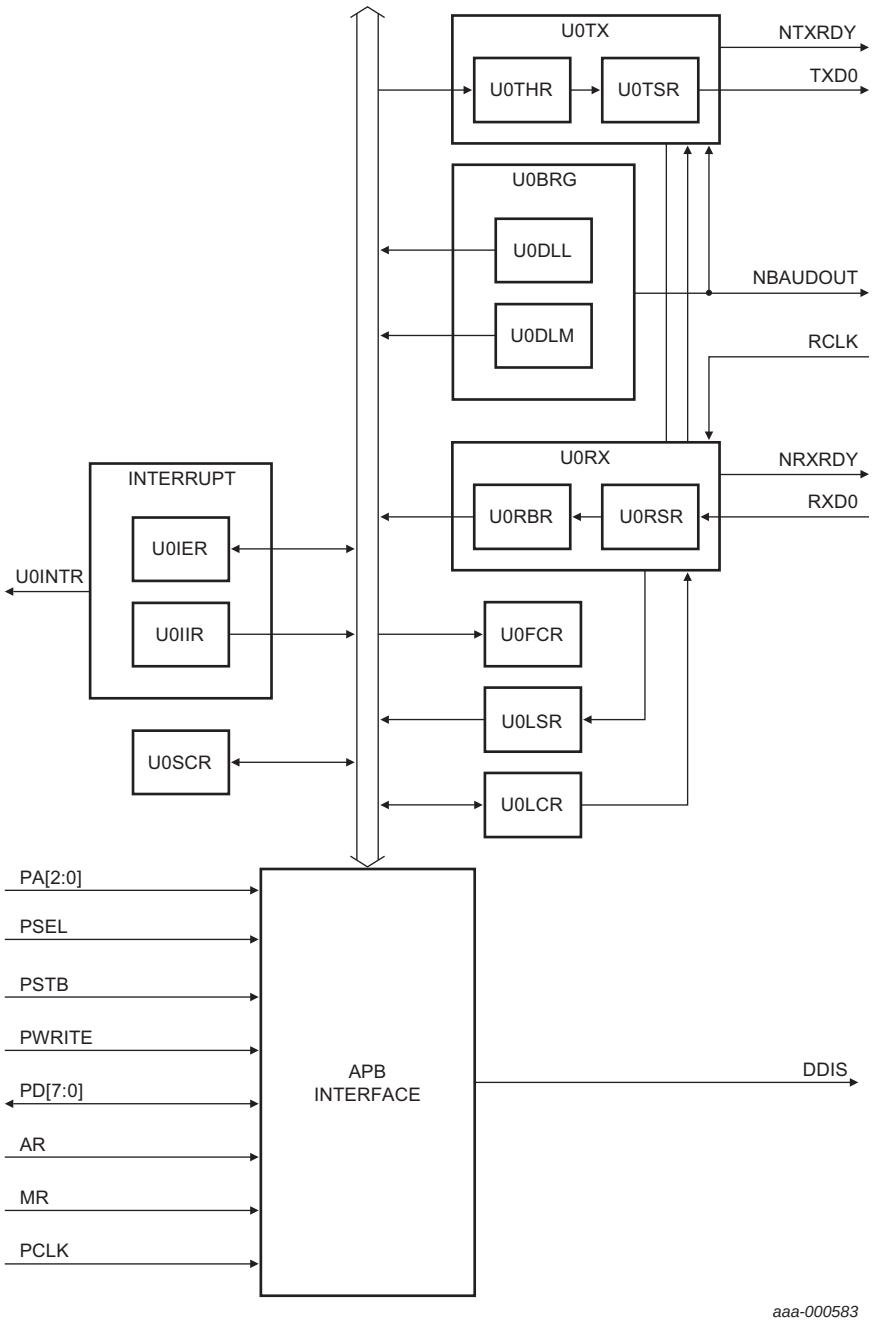


Fig 20. UART0 block diagram

15. Universal Asynchronous Receiver/Transmitter 1 (UART1)

15.1 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 auto-bauding 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

15.2 Pin description

Table 98. 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 external modem is ready to accept transmitted data via TXD1 from UART1. In normal operation of modem interface (U1MCR[4] = 0), complement value of this signal is stored in U1MSR[4]. If enabled (U1IER[3] = 1), state change information is stored in U1MSR[0] and is a source for a priority level 4 interrupt.
DCD1	input	data carrier detect. Active LOW signal indicates if external modem has established a communication link with UART1 and data can be exchanged. In normal operation of modem interface (U1MCR[4]=0), complement value of this signal is stored in U1MSR[7]. If enabled (U1IER[3] = 1), state change information is stored in U1MSR3 and is a source for a priority level 4 interrupt.
DSR1	input	data set ready. Active LOW signal indicates if external modem is ready to establish a communications link with UART1. In normal operation of modem interface (U1MCR[4] = 0), complement value of this signal is stored in U1MSR[5]. If enabled (U1IER[3] = 1), state change information is stored in U1MSR[1] and is a source for a priority level 4 interrupt.
DTR1	output	data terminal ready. Active LOW signal indicates that UART1 is ready to establish connection with external modem. Complement value of this signal is stored in U1MCR[0].
RI1	input	ring indicator. Active LOW signal indicates that telephone ringing signal is detected by modem. In normal operation of modem interface (U1MCR[4] = 0), complement value of this signal is stored in U1MSR[6]. If enabled (U1IER[3] = 1), state change information is stored in U1MSR[2] and is a source for a priority level 4 interrupt.
RTS1	output	request to send. Active LOW signal indicates that UART1 wants to transmit data to external modem. Complement value of this signal is stored in U1MCR[1].

15.3 Register description

UART1 contains registers organized as shown in Table 99. The Divisor Latch Access Bit (DLAB) is contained in U1LCR[7] and enables access to the divisor latches.

Table 99. 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	n/a	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding register	8-bit write data								WO	n/a	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	-	-	-	-	-	-	ABTO IntEn	ABEO Int	R/W	0x00	0xE001 0004 (DLAB=0)
		CTS Interrupt Enable	-	-	-	Modem Status Interrupt Enable	RX Line Interrupt Enable	THRE Interrupt Enable	RBR Interrupt Enable			
U1IIR	Interrupt Identification register	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE001 0008
		FIFO Enable		-	-	Interrupt Identification			Interrupt Pending			
U1FCR	FIFO Control register	RX Trigger Level		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control register	DLAB	Break Control	Parity Select		Parity Enable	Stop Bit Select	Word Length Select		R/W	0x00	0xE001 000C
U1MCR	Modem Control register	CTSen	RTSen	-	Loopback Mode Select	-	-	RTS Control	DTR Control	R/W	0x00	0xE001 0010
U1LSR	Line Status register	RXFE	TEMT	THRE	BI	FE	PE	OE	RDR	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 register	8-bit data								R/W	0x00	0xE001 001C
U1ACR	Auto-baud Control register	-	-	-	-	-	-	ABTO IntClr	ABEO IntClr	R/W	0x00	0xE001 0020
		-	-	-	-	-	Auto Restart	Mode	Start			

Table 99. UART1 register map *...continued*

Name	Description	Bit functions and addresses	Access	Reset value[1]	Address
		MSB LSB			
U1FDR	Fractional Divider register	reserved[31:8]	R/W	0x10	0xE001 0028
		MULVAL DIVADDVAL			
U1TER	TX Enable register	TXEN - - - - - - -	R/W	0x80	0xE001 0030

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

15.3.1 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 logic 0s.

The Divisor Latch Access Bit (DLAB) in U1LCR must be logic 0 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 (that is, the one that is read in the next read from the RBR), the right approach for fetching the valid pair of received bytes and status bits is first to read the content of register U1LSR, and then to read a byte from the U1RBR.

Table 100. UART1 Receiver buffer register (U1RBR - address 0xE001 0000, when DLAB = 0 read only) bit description

Bit	Symbol	Description	Reset value
7:0	RBR	UART1 receiver buffer register contains oldest received byte in UART1 Rx FIFO	undefined

15.3.2 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 logic 0 to access the U1THR. The U1THR is always write only.

Table 101. 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 UART1 transmit holding register stores data in UART1 transmit FIFO. Byte is sent when it reaches bottom of FIFO and when the transmitter is available.	n/a

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

The UART1 divisor latch is part of the UART1 fractional baud rate generator and holds the value used to divide the clock supplied by the fractional prescaler to produce the baud rate clock, which must be 16× the desired baud rate (Equation 4 on page 111). Registers U1DLL and U1DLM 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 logic 1 to access the UART1 divisor latches.

Details on how to select the right value for U1DLL and U1DLM can be found later on in this chapter.

Table 102. UART1 Divisor latch LSB register (U1DLL - address 0xE001 0000, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLLSB	UART1 divisor latch LSB register and U1DLM register determines baud rate of UART1	0x01

Table 103. UART1 Divisor latch MSB register (U1DLM - address 0xE001 0004, when DLAB = 1) bit description

Bit	Symbol	Description	Reset value
7:0	DLMSB	UART1 divisor latch MSB register and U1DLL register determines baud rate of UART1	0x00

15.3.4 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.

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

Table 104. 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 logic 0, fractional baud rate generator will not affect UARTn baud rate.	0
7:4	MULVAL	1	baud rate pre-scaler multiplier value. Must be greater than or equal to 1 for UARTn to operate properly, regardless of whether fractional baud rate generator is used or not.	1
31:8	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	0

This register controls the clock pre-scaler for 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 baud rate can be calculated as (n = 1):

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)} \quad (3)$$

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

The value of MULVAL and DIVADDVAL must comply with the following conditions:

- $0 < MULVAL \leq 15$
- $0 \leq DIVADDVAL < 15$
- DIVADDVAL < MULVAL

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

If register U1FDR 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 is not divided.

15.3.4.1 Baud rate calculation

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

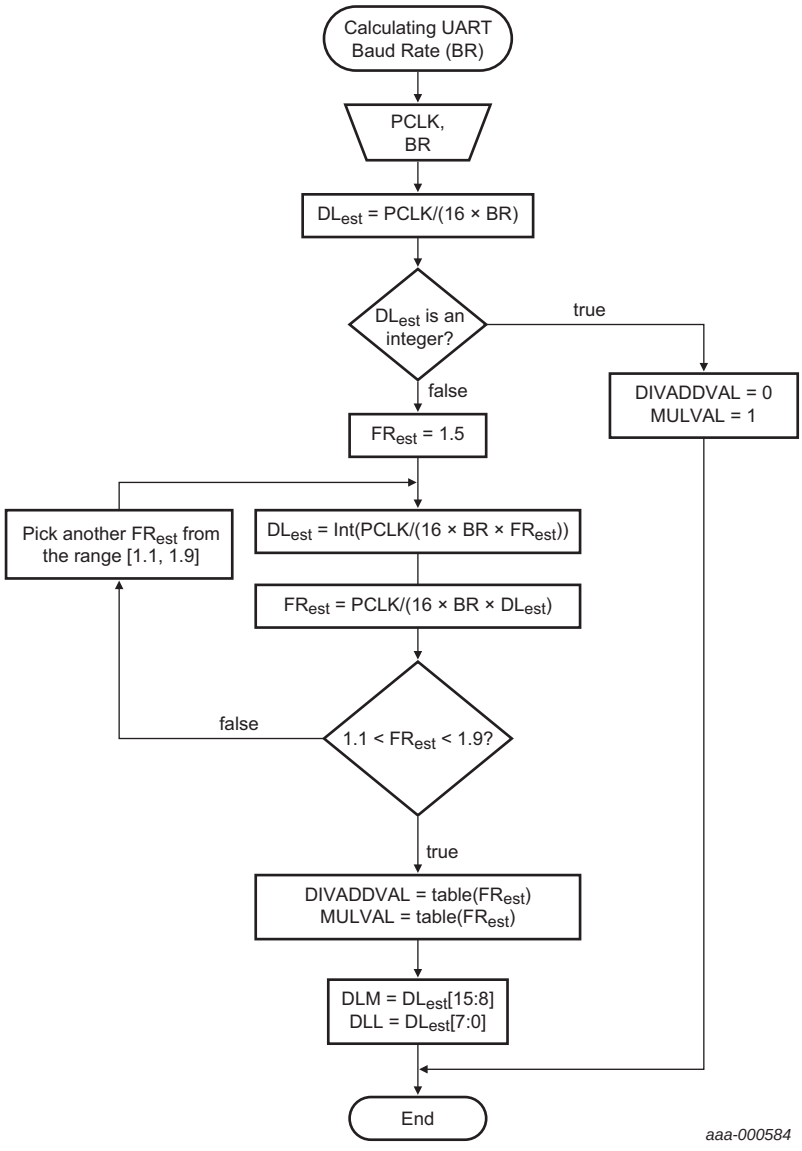


Fig 21. Algorithm for setting UART dividers

Table 105. 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

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

Example 2: PCLK = 12 MHz, BR = 115200: according to the provided algorithm DL_{est} = PCLK / (16 × BR) = 12 MHz / (16 × 115200) = 6.51. This DL_{est} is not an integer number and the next step is to estimate the FR parameter. Using an initial estimate of FR_{est} = 1.5, a new DL_{est} = 4 is calculated and FR_{est} is recalculated as FR_{est} = 1.628. Since FR_{est} = 1.628 is within the specified range of 1.1 and 1.9, DIVADDVAL and MULVAL values can be obtained from the attached look-up table.

The closest value for FR_{est} = 1.628 in the look-up Table 88 on page 81 is FR = 1.625. It is equivalent to DIVADDVAL = 5 and MULVAL = 8.

Based on these findings, the suggested UART setup is: DLM = 0, DLL = 4, DIVADDVAL = 5, and MULVAL = 8. According to Equation 3 on page 98 the UART's baud rate is 115384 Bd. This rate has a relative error of 0.16 % from the originally specified value of 115200 Bd.

15.3.5 UART1 Interrupt enable register (U1IER - 0xE001 0004, when DLAB = 0)

The U1IER is used to enable UART1 interrupt sources.

Table 106. 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 receive data available interrupt for UART1. It also controls character receive time-out interrupt.	0
		0	disables RDA interrupts	
		1	enables RDA interrupts	
1	THRE Interrupt Enable		U1IER[1] enables THRE interrupt for UART1. Status of this interrupt can be read from U1LSR[5].	0
		0	disables THRE interrupts	
		1	enables THRE interrupts	
2	RX Line Interrupt Enable		U1IER[2] enables UART1 Rx line status interrupts. Status of this interrupt can be read from U1LSR[4:1].	0
		0	disables = Rx line status interrupts	
		1	enables Rx line status interrupts	
3	Modem Status Interrupt Enable		U1IER[3] enables modem interrupt. Status of this interrupt can be read from U1MSR[3:0].	0
		0	disables modem interrupt	
		1	enables modem interrupt	
6:4	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
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 generates an interrupt if Modem Status Interrupt Enable (U1IER[3]) is set. In normal operation a CTS1 signal transition generates a Modem Status Interrupt unless the interrupt is disabled by clearing bit U1IER[3] in register U1IER. In auto-CTS mode, a transition on bit CTS1 triggers 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 must not write logic 1s to reserved bits; the value read from a reserved bit is not defined	n/a

15.3.6 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 107. 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. 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 interrupt corresponding to 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7:6	FIFO Enable		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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Interrupts are handled as described in Table 108. 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. In order to clear the interrupt before exiting the Interrupt Service Routine, the U1IIR must be read.

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 after 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 U1FCR[7:6] 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) clears the interrupt. This interrupt is intended to flush the UART1 RBR after a message is received that is not a multiple of the trigger level size. For example, if a peripheral wants to send a 105 character message and the trigger level is 10 characters, the CPU receives 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 108. 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 Rx FIFO and no character input or removed during time period depending on how many characters are in FIFO and what trigger level is (3.5 to 4.5 character times). exact time is: [(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 2), “0011”, “0101”, “0111”, “1000”, “1001”, “1010”, “1011”,”1101”,”1110”,”1111” are reserved.

[2] For details see Section 15.3.10 on page 107

[3] For details see Section 15.3.1 on page 97

[4] For details see Section 15.3.6 on page 102 and Section 15.3.2 on page 97

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. If the UART1 THR FIFO has held two or more characters at one time and currently, the U1THR is empty, a THRE interrupt is set immediately. 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 MPT612. 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 generates a modem interrupt. The source of the modem interrupt can be determined by examining U1MSR[3:0]. A U1MSR read clears the modem interrupt.

15.3.7 UART1 FIFO control register (U1FCR - 0xE001 0008)

The U1FCR controls the operation of the UART1 Rx and Tx FIFOs.

Table 109. 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 application.	0
		1	active HIGH enable for both UART1 Rx and Tx FIFOs and U1FCR[7:1] access. Must be set for correct UART1 operation. Any transition on this bit automatically clears UART1 FIFOs.	
1	RX FIFO Reset	0	no impact on either UART1 FIFO	0
		1	writing a logic 1 to U1FCR[1] clears all bytes in UART1 Rx FIFO and resets pointer logic. This bit is self-clearing.	
2	TX FIFO Reset	0	no impact on either UART1 FIFO	0
		1	writing a logic 1 to U1FCR[2] clears all bytes in UART1 Tx FIFO and resets pointer logic. This bit is self-clearing.	
5:3	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7:6	RX Trigger Level		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)	

15.3.8 UART1 Line control register (U1LCR - 0xE001 000C)

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

Table 110. 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)	
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 transmitted character and attached parity bit is odd.	0
		01	even parity. Number of 1s in transmitted character and attached parity bit is even.	
		10	forced 1 stick parity	
		11	forced 0 stick parity	

Table 110. UART1 Line control register (U1LCR - address 0xE001 000C) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
6	Break Control	0	disable break transmission	0
		1	enable break transmission. Output pin UART1 TXD 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	

15.3.9 UART1 Modem control register (U1MCR - 0xE001 0010)

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

Table 111. 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. Reads as logic 0 when modem loopback mode is active.	0
1	RTS Control	-	source for modem output pin RTS. Reads as logic 0 when modem loopback mode is active.	0
3:2	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
4	Loopback Mode Select		modem loopback mode provides a mechanism to perform diagnostic loopback testing. Serial data from transmitter is connected internally to serial input of receiver. Input pin RXD1 has no effect on loopback, and output pin TXD1 is held in marking state. Modem inputs (CTS, DSR, RI and DCD) are disconnected externally. Externally, modem outputs (RTS, DTR) are set inactive. Internally, four modem outputs are connected to four modem inputs. Therefore, upper 4 bits of U1MSR are driven by the lower 4 bits of U1MCR rather than the 4 modem inputs in normal mode. Permits modem status interrupts to be generated in loopback mode by writing lower 4 bits of U1MCR.	0
		0	disable modem loopback mode	
		1	enable modem loopback mode	
5:3	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
6	RTSen		auto-RTS control bit	0
		0	disable auto-RTS flow control	
		1	enable auto-RTS flow control	
7	CTSen		auto-CTS control bit	0
		0	disable auto-CTS flow control	
		1	enable auto-CTS flow control	

15.3.9.1 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 only starts transmitting if the CTS1 input signal is asserted.

Auto-RTS: The auto-RTS function is enabled by setting bit RTSen. 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 reaching the trigger level (assuming sending UART has another byte to send) because it may 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, bit RTSen controls the RTS1 output of the UART1. If Auto-RTS mode is enabled, hardware controls the RTS1 output, and the actual value of RTS1 is copied in bit RTS Control of the UART1. As long as Auto-RTS is enabled, the value of bit RTS Control is read-only for software.

Example: Suppose UART1 operating in type 550 has the trigger level in U1FCR set to 0x2, if Auto-RTS is enabled, UART1 deasserts the RTS1 output when the receive FIFO contains 8 bytes (Table 109 on page 104). The RTS1 output is reasserted when the receive FIFO reaches the previous trigger level: 4 bytes.

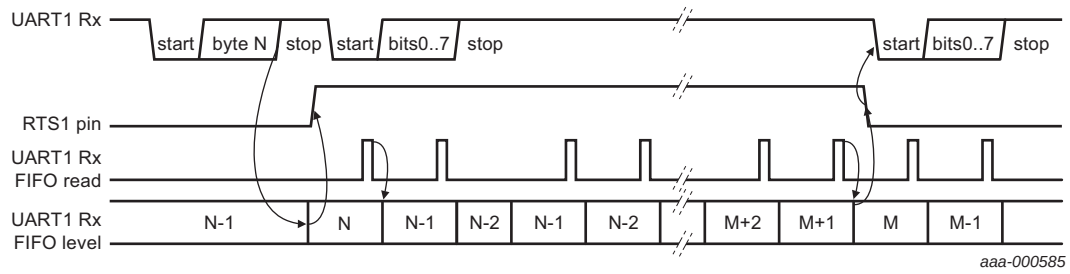


Fig 22. Auto-RTS functional timing

Auto-CTS: The auto-CTS function is enabled by setting bit CTSen. 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 bit CTS Interrupt Enable is set, however bit Delta CTS in U1MSR is set. Table 112 lists the conditions for generating a Modem Status interrupt.

Table 112. 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

Table 112. Modem status interrupt generation *…continued*

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
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 23 illustrates the auto-CTS functional timing.

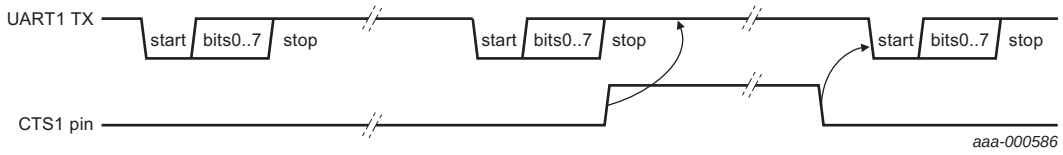


Fig 23. Auto-CTS functional timing

The CTS1 signal is asserted while starting transmission of the initial character. Transmission stalls when the pending transmission has completed. The UART continues transmitting a logic 1 bit as long as CTS1 is deasserted (HIGH). When CTS1 is deasserted, transmission resumes and a start bit is sent followed by the data bits of the next character.

15.3.10 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 113. 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 U1RBR holds an unread character and is cleared when UART1 RBR FIFO is empty	0
		0	U1RBR is empty	
		1	U1RBR contains valid data	
1	Overrun Error (OE)		overrun error condition is set if overrun occurs. An U1LSR read clears U1LSR[1]. U1LSR[1] is set when UART1 RSR has a new character assembled and UART1 RBR FIFO is full. In this case, UART1 RBR FIFO is not overwritten and character in UART1 RSR is lost.	0
		0	overrun error status is inactive	
		1	overrun error status is active	

Table 113. UART1 Line status register (U1LSR - address 0xE001 0014, read only) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
2	Parity Error (PE)		when parity bit of received character is the wrong state, a parity error occurs. An U1LSR read clears U1LSR[2]. Time of parity error detection is dependent on U1FCR[0]. **Remark:** Parity error is associated with character at top of UART1 RBR FIFO	0
		0	parity error status is inactive	
		1	parity error status is active	
3	Framing Error (FE)		when stop bit of received character is logic 0, a framing error occurs. An U1LSR read clears U1LSR[3]. Time of framing error detection is dependent on U1FCR0. On detection of framing error, Rx attempts to resynchronize to data and assumes that bad stop bit is an early start bit. However, it cannot be assumed that next received byte is correct even if there is no framing error. **Remark:** Framing error is associated with character at top of 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 spacing state (all 0s) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once break condition is detected, receiver is idle until RXD1 enters marking state (all 1s). An U1LSR read clears this status bit. Time of break detection is dependent on U1FCR[0]. **Remark:** Break interrupt is associated with character at top of 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 on 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 U1TSR or U1THR contain valid data	1
		0	U1THR and/or U1TSR contains valid data	
		1	U1THR and U1TSR are empty	
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 U1RBR. Cleared when register U1LSR is read and there are no subsequent errors in UART1 FIFO.	0
		0	U1RBR contains no UART1 Rx errors or U1FCR[0] = 0	
		1	UART1 RBR contains at least one UART1 Rx error	

15.3.11 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.

Remark: Modem signals have no direct effect on UART1 operation, they facilitate software implementation of modem signal operations.

Table 114. UART1 Modem status register (U1MSR - address 0xE001 0018) bit description

Bit	Symbol	Value	Description	Reset value
0	Delta CTS		set on state change of input CTS. Cleared on 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 U1MSR read.	0
		0	no change detected on modem input, DSR	
		1	state change detected on modem input, DSR	
2	Trailing Edge RI		set on LOW-to-HIGH transition of input RI. Cleared on U1MSR read.	0
		0	no change detected on modem input, RI	
		1	LOW-to-HIGH transition detected on RI	
3	Delta DCD		set on 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. Connected to U1MCR[1] in modem loopback mode.	0
5	DSR		data set ready state. Complement of input signal DSR. Connected to U1MCR[0] in modem loopback mode.	0
6	RI		ring indicator state. Complement of input RI. Connected to U1MCR[2] in modem loopback mode.	0
7	DCD		data carrier detect state. Complement of input DCD. Connected to U1MCR[3] in modem loopback mode.	0

15.3.12 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 the user's discretion. There is no provision in the interrupt interface that indicates to the host that a read or write of the U1SCR has occurred.

Table 115. UART1 Scratch pad register (U1SCR - address 0xE001 0014) bit description

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

15.3.13 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 baud rate generation and can be read and written at the user's discretion.

Table 116. Auto-baud control register (U1ACR - address 0xE001 0020) bit description

Bit	Symbol	Value	Description	Reset value
0	Start		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. Automatically cleared after auto-baud completion.	
1	Mode		auto-baud mode select bit	0
		0	mode 0	
		1	mode 1	

Table 116. Auto-baud control register (U1ACR - address 0xE001 0020) bit description

Bit	Symbol	Value	Description	Reset value
2	AutoRestart	0	no restart	0
		1	restart in case of time-out (counter restarts at next UART1 Rx falling edge)	
7:3	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	0
8	ABEOIntClr		end of auto-baud interrupt clear bit (write only accessible). Writing a logic 1 clears the corresponding interrupt in the U1IIR. Writing a logic 0 has no impact.	0
9	ABTOIntClr		auto-baud time-out interrupt clear bit (write only accessible). Writing a logic 1 clears the corresponding interrupt in the U1IIR. Writing a logic 0 has no impact.	0
31:10	-	n/a	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined.	0

15.3.14 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 measures the bit time of the receive data stream and sets the divisor latch registers U1DLM and U1DLL accordingly.

Auto-baud is started by setting bit U1ACR Start. Auto-baud can be stopped by clearing bit U1ACR Start. The start bit clears once auto-baud has finished, and reading the bit returns the status of auto-baud (pending/finished).

Two auto-baud measuring modes are available which can be selected by bit U1ACR Mode. In mode 0 the baud rate is measured on two subsequent falling edges of pin UART1 Rx (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 pin UART1 Rx (the length of the start bit).

If a time-out occurs (the rate measurement counter overflows), bit U1ACR AutoRestart can be used to restart baud rate measurement automatically. If this bit is set, the rate measurement restarts at the next falling edge of pin UART1 Rx.

The auto-baud function can generate two interrupts:

- The U1IIR ABTOInt interrupt is set if the interrupt is enabled (U1IER ABToIntEn is set and the auto-baud rate measurement counter overflows)
- The U1IIR ABEOInt interrupt is 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 bits U1ACR ABTOIntClr and ABEOIntEn.

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 will affect the measuring of pin UART1 Rx baud rate, but the value of register U1FDR is not going to be modified after rate measurement. Also, when auto-baud is used, any write

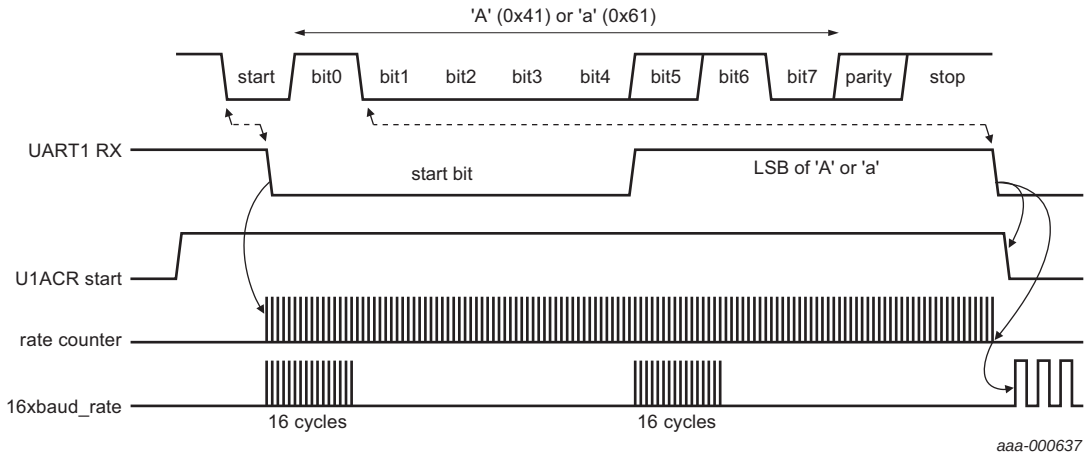
to registers U1DLM and U1DLL must be done before register U1ACR write. The minimum and the maximum baud rates supported by UART1 are function of PCLK, number of data bits, stop-bits and parity bits.

$$ratemin = \frac{2 \times PCLK}{16 \times 2^{15}} \leq UART1_{baudrate} \leq \frac{PCLK}{16 \times (2 + databits + paritybits + stopbits)} = ratemax \quad (4)$$

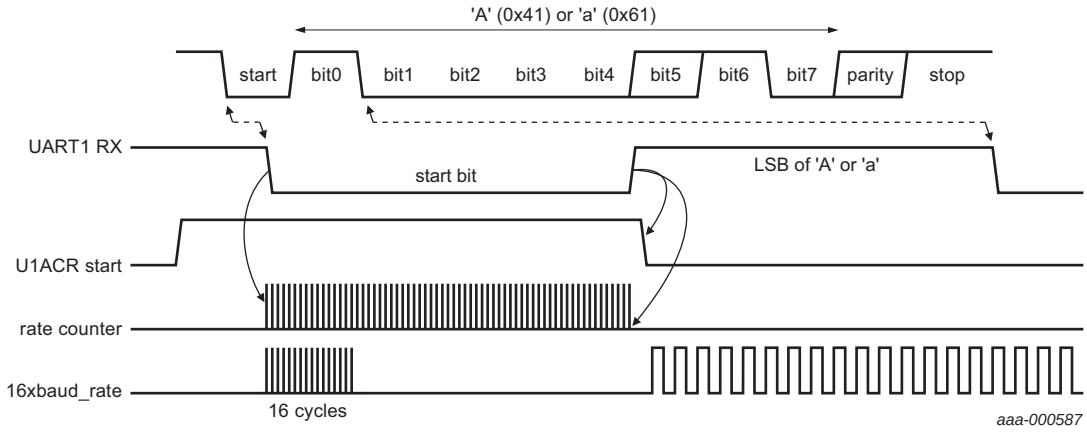
15.3.15 Auto-baud modes

When the software is expecting an AT command, it configures UART1 with the expected character format and sets bit U1ACR Start. The initial values in the divisor latches U1DLM and U1DLM are do not care. Because of the ”A" or ”a" ASCII coding (”A" = 0x41, ”a" = 0x61), pin UART1 Rx sensed start bit and the LSB of the expected character are delimited by two falling edges. When bit U1ACR Start is set, the auto-baud protocol executes the following phases:

1. On setting bit U1ACR Start, the baud rate measurement counter is reset and the UART1 U1RSR is reset. The U1RSR baud rate is switched to the highest rate.
2. A falling edge on pin UART1 Rx triggers the beginning of the start bit. The rate measuring counter starts counting PCLK cycles optionally pre-scaled by the fractional baud rate generator.
3. During receipt of the start bit, 16 pulses are generated on the RSR baud input at the frequency of the (fractional baud rate pre-scaled) UART1 input clock, guaranteeing the start bit is stored in U1RSR.
4. During the receipt of the start bit (and the character LSB for mode = 0), the rate counter continues incrementing with the pre-scaled UART1 input clock (PCLK).
5. If Mode = 0, the rate counter stops on the next falling edge of pin UART1 Rx. If Mode = 1, the rate counter stops on the next rising edge of pin UART1 Rx.
6. The rate counter is loaded into U1DLM/U1DLL and the baud rate is switched to normal operation. After setting U1DLM/U1DLL, the end of auto-baud interrupt U1IIR ABEOInt is set, if enabled. The U1RSR now continues to receive 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 24. Auto-baud mode 0 and mode 1 waveforms

15.3.16 UART1 Transmit enable register (U1TER - 0xE001 0030)

MPT612's U1TER enables implementation of software and hardware flow control. When TXEn = 1, UART1 transmitter sends data as long as it is available. When bit TXEn is logic 0, UART1 transmission stops.

Table 117 describes how to use bit TXEn to achieve software flow control.

Table 117. UART1 Transmit enable register (U1TER - address 0xE001 0030) bit description

Bit	Symbol	Description	Reset value
6:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
7	TXEN	if logic 1, as it is after a reset, data written to THR is output on pin TXD after any preceding data is sent. If cleared to logic 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, if a bit is set to logic 0, it blocks the transfer of characters from THR or Tx FIFO into the transmit shift register. Software can clear this bit when it detects that the -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 is true, or when it receives an XON (DC1) character.	1

15.4 Architecture

The architecture of the UART1 is shown in block diagram Figure 25.

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 registers U1DLL and U1DLM and is a 16× 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.

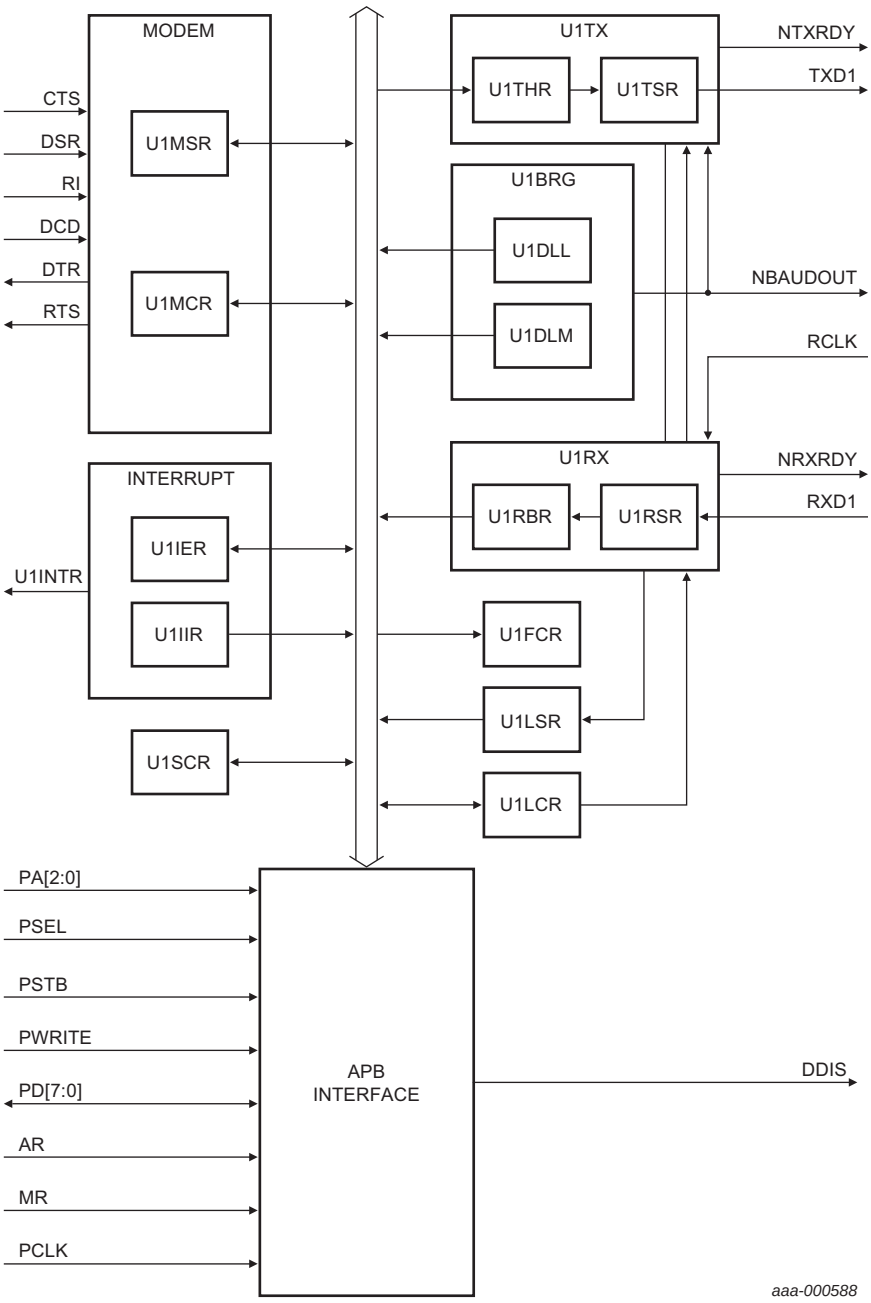


Fig 25. UART1 block diagram

16. I²C interfaces I²C0 and I²C1

16.1 Features

- Standard I²C-compliant bus interfaces can be configured as Master, Slave, or Master/Slave

- Arbitration is handled between simultaneously transmitting masters without corruption of serial data on the bus
- Programmable clock allows adjustment of I^2C transfer rates
- Data transfer is bidirectional between masters and slaves
- Serial clock synchronization allows devices with different bit rates to communicate via one serial bus
- Serial clock synchronization is used as a handshake mechanism to suspend and resume serial transfer
- I^2C-bus can be used for test and diagnostic purposes

16.2 Applications

Interfaces to external I^2C standard parts, such as serial RAMs, LCDs and tone generators.

16.3 Description

A typical I^2C-bus configuration is shown in Figure 26. Depending on the state of the direction bit (R/W), two types of data transfers are possible on the I^2C-bus:

- Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte.
- Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. Next follows the data bytes transmitted by the slave to the master. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a “not acknowledge” is returned. The master device generates all of the serial clock pulses and the Start and Stop conditions. A transfer is ended with a Stop condition or with a repeated Start condition. Since a repeated Start condition is also the beginning of the next serial transfer, the I^2C-bus is not released.

The MPT612 I^2C interfaces are byte oriented and have four operating modes: master transmitter mode, master receiver mode, slave transmitter mode and slave receiver mode.

The I^2C interfaces comply with the entire I^2C specification, supporting the ability to turn off power to the MPT612 without interfering with other devices on the same I^2C-bus.

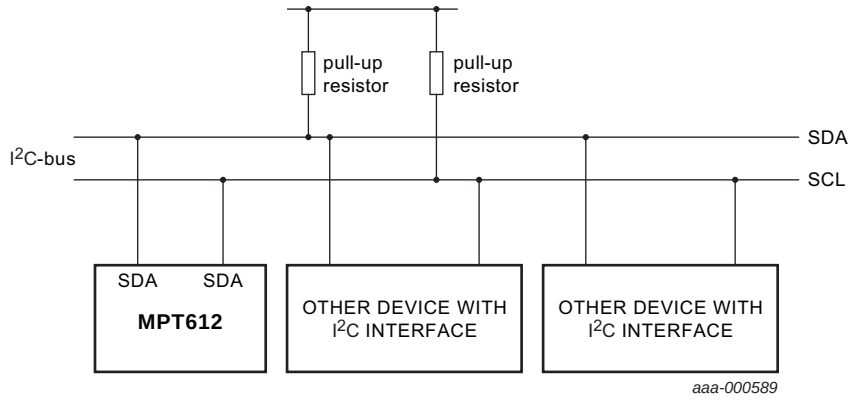


Fig 26. I²C-bus configuration

16.4 Pin description

Table 118. I²C pin description

Pin	Type	Description
SDA0,1	input/output	I²C serial data
SCL0,1	input/output	I²C serial clock

16.5 I²C operating modes

In a given application, the I²C block can 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.

16.5.1 Master transmitter mode

In this mode, data is transmitted from master to slave. Before the master transmitter mode can be entered, register I2CONSET must be initialized as shown in Table 119. To enable the I²C function, I2EN must be set to logic 1. If bit AA is logic 0, the I²C interface will not acknowledge any address when another device is master of the bus, so it cannot enter slave mode. The STA, STO and SI bits must be logic 0. Bit SI is cleared by writing logic 1 to bit SIC in register I2CONCLR.

Table 119. I2C0CONSET and I2C1CONSET used to configure Master mode

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

The first byte transmitted contains the slave address of the receiving device (7 bits) and the data direction bit. In this mode, the data direction bit (R/W) must be logic 0 which means write. The first byte transmitted contains the slave address and write bit. Data is

transmitted 8 bits at a time. After each byte is transmitted, an acknowledge bit is received. Start and Stop conditions are output to indicate the beginning and the end of a serial transfer.

The I^2C interface enters master transmitter mode when software sets bit STA. The I^2C logic sends the Start condition when the bus is free. After the Start condition is transmitted, bit SI is set, and the status code in register I2STAT is 0x08. This status code is used to vector to a state service routine which loads the slave address and write bit to register I2DAT, and then clears bit SI. SI is cleared by writing a logic 1 to bit SIC bit in register I2CONCLR.

When the slave address and R/W bit are transmitted and an acknowledgment bit is received, bit SI 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 logic 1). The appropriate actions to be taken for each of these status codes are shown in Table 134 on page 135 to Table 137 on page 139.

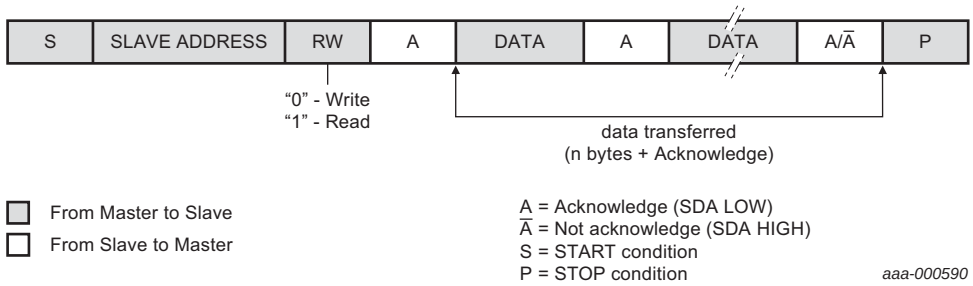


Fig 27. Format in the master transmitter mode

16.5.2 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 is transmitted, the interrupt service routine must load the slave address and the data direction bit to register I^2C Data (I2DAT), and then clear bit SI. In this case, the data direction bit (R/W) must be logic 1 to indicate a read.

When the slave address and data direction bit have been transmitted and an acknowledge bit is received, bit SI is set, and the status register shows the status code. For master mode, the possible status codes are 0x40, 0x48, or 0x38. For slave mode, the possible status codes are 0x68, 0x78, or 0xB0. For details, see Table 135 on page 136.

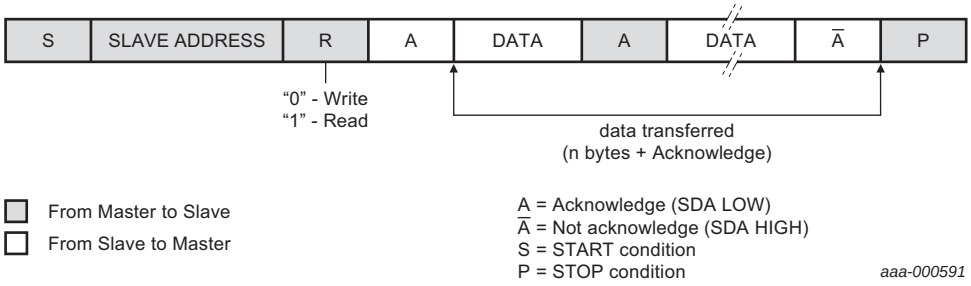


Fig 28. Format of master receiver mode

After a repeated Start condition, I²C can switch to the master transmitter mode.

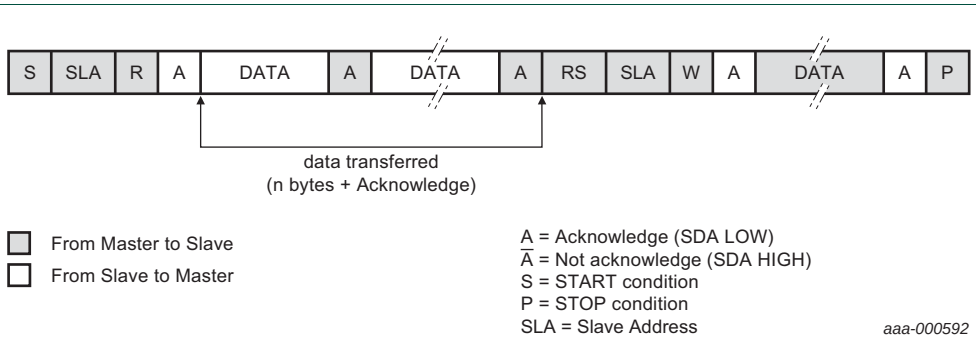


Fig 29. A master receiver switches to master transmitter after sending repeated Start

16.5.3 Slave receiver mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, the user writes to the Slave Address register (I2ADR) and writes to register I²C Control Set (I2CONSET) as shown in Table 120.

Table 120. I2C0CONSET and I2C1CONSET used to configure Slave mode

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

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

After I2ADR and I2CONSET are initialized, the I²C interface waits until addressed by its own address or general address followed by the data direction bit. If the direction bit is logic 0 (W), it enters slave receiver mode. If the direction bit is logic 1 (R), it enters slave transmitter mode. After the address and direction bit have been received, bit SI is set and a valid status code can be read from the status register (I2STAT). Refer to Table 136 on page 137 for the status codes and actions.

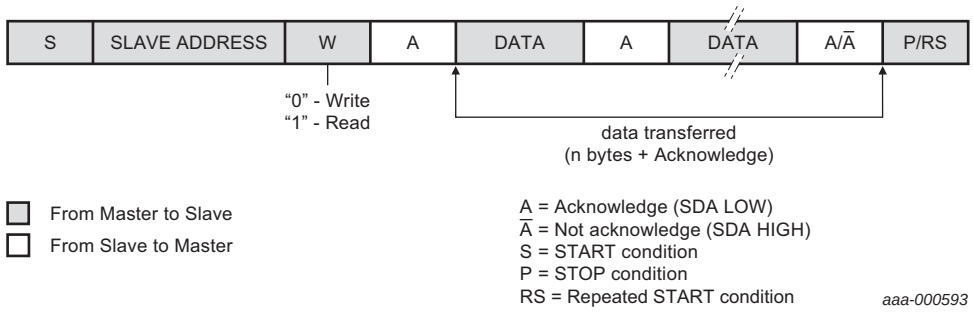


Fig 30. Format of slave receiver mode

16.5.4 Slave transmitter mode

The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit is logic 1, indicating a read operation. Serial data is transmitted via SDA while the serial clock is input through SCL. Start and Stop conditions are recognized as the beginning and end of a serial transfer. In a given application, I^2C can operate as a master and as a slave. In the slave mode, the I^2C hardware looks for its own slave address and the general call address. If one of these addresses is detected, an interrupt is requested. When the MPT612 wants to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave action is not interrupted. If bus arbitration is lost in the master mode, the I^2C interface switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

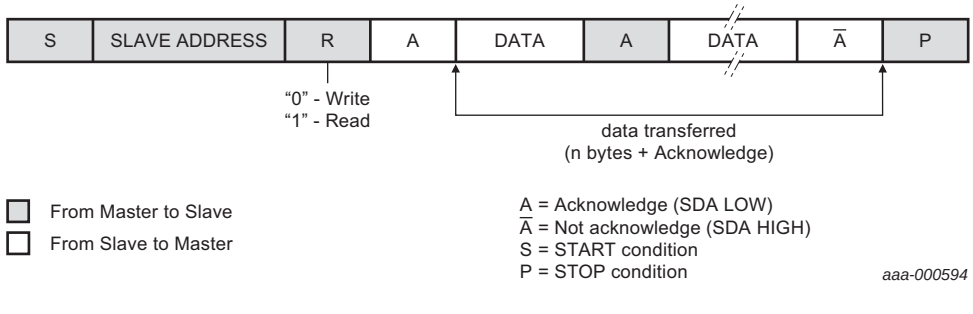


Fig 31. Format of slave transmitter mode

16.6 I^2C implementation and operation

Figure 32 shows how the on-chip I2C-bus interface is implemented, and the following text describes the individual blocks.

16.6.1 Input filters and output stages

Input signals are synchronized with the internal clock, and spikes shorter than three clocks are filtered out.

The output for I^2C is a special pad designed to conform to the I^2C specification.

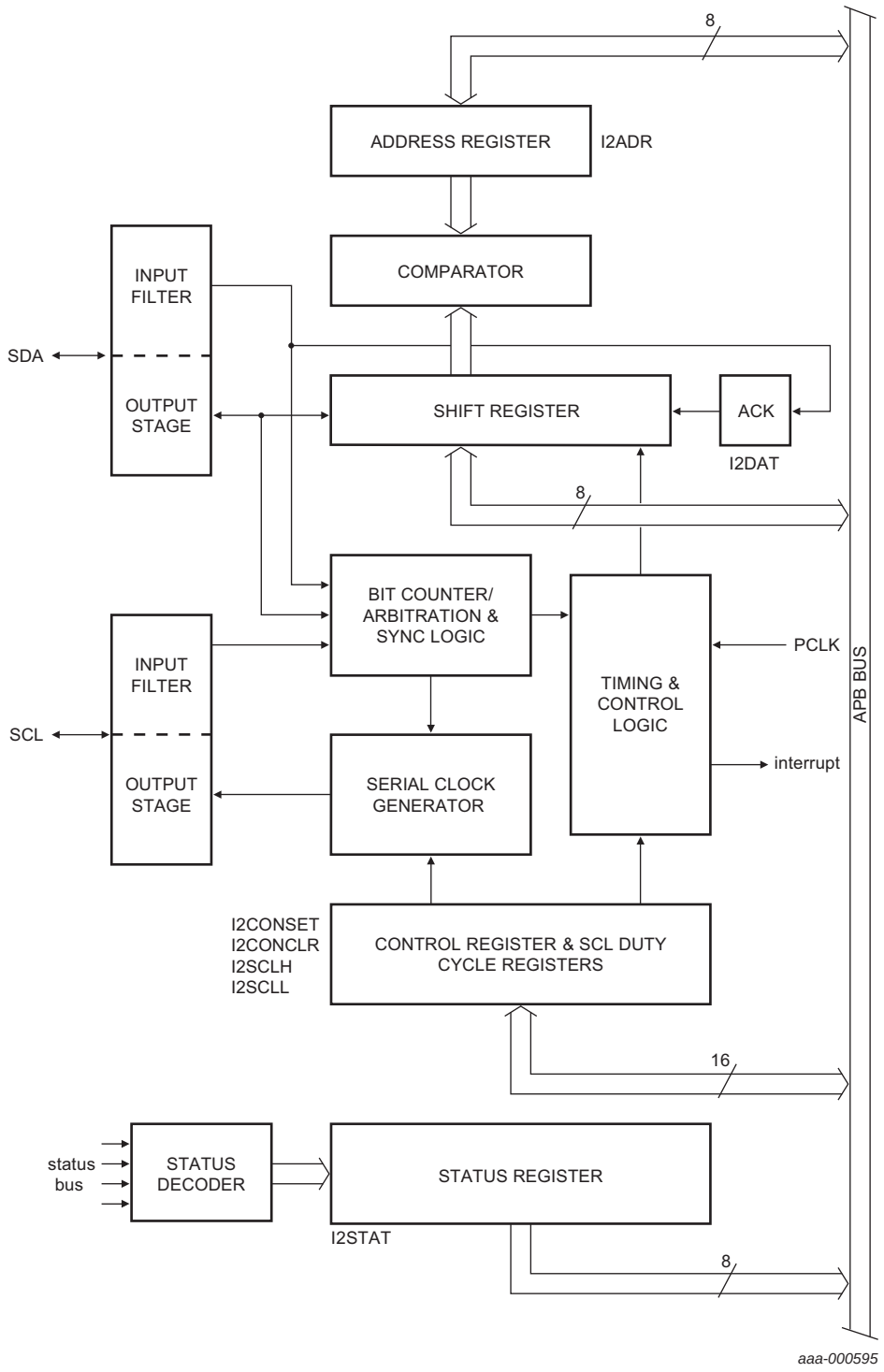


Fig 32. I²C serial interface block diagram

16.6.2 Address register, I2ADDR

This register can be loaded with the 7-bit slave address (7 most significant bits) to which the I^2C block responds when programmed as a slave transmitter or receiver. The LSB (GC) is used to enable general call address (0x00) recognition.

16.6.3 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.

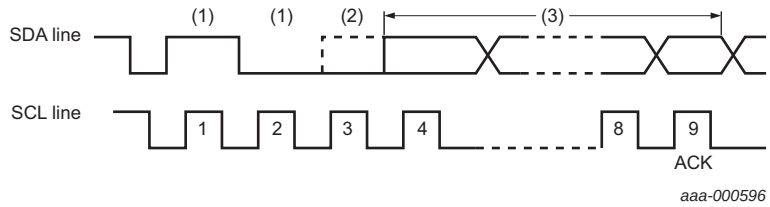
16.6.4 Shift register, I2DAT

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 transmitted is the MSB (bit 7) and, after a byte is received, the first bit of received data is 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.

16.6.5 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 I^2C-bus. If another device on the bus overrules a logic 1 and pulls the SDA line LOW, arbitration is lost, and the I^2C block immediately changes from master transmitter to slave receiver. The I^2C block continues to output clock pulses (on SCL) until transmission of the current serial byte is complete.

Arbitration can also be lost in the master receiver mode. Loss of arbitration in this mode can only occur while the I^2C block is returning a “not acknowledge: (logic 1) to the bus. Arbitration is lost when another device on the bus pulls this signal LOW. Since this can occur only at the end of a serial byte, the I^2C block generates no further clock pulses. Figure 33 shows the arbitration procedure.



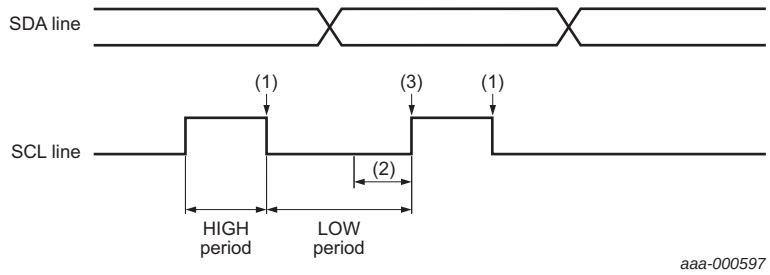
(1) Another device transmits serial data.

(2) Another device overrules a logic 1 (dotted line) transmitted by this I^2C master by pulling the SDA line LOW. Arbitration is lost, and this I^2C enters slave receiver mode.

(3) This I^2C is in slave receiver mode but still generates clock pulses until the current byte is transmitted. This I^2C will not generate clock pulses for the next byte. Data on SDA originates from the new master once it has won arbitration.

Fig 33. Arbitration procedure

The synchronization logic synchronizes the serial clock generator with the clock pulses on the SCL line from another device. If two or more master devices generate clock pulses, the “mark” duration is determined by the device that generates the shortest “marks,” and the “space” duration is determined by the device that generates the longest “spaces”. Figure 34 shows the synchronization procedure.



(1) Another device pulls the SCL line LOW before this I²C has timed a complete HIGH time. The other device effectively determines the (shorter) HIGH period.

(2) Another device continues to pull the SCL line LOW after this I²C has timed a complete LOW time and released SCL. The I²C clock generator is forced to wait until SCL goes HIGH. The other device effectively determines the (longer) LOW period.

(3) The SCL line is released, and the clock generator begins timing the HIGH time.

Fig 34. Serial clock synchronization

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

16.6.6 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 slave mode. The I²C output clock frequency and duty cycle is programmable via the I²C Clock Control registers. See the description of registers I2CSCLL and I2CSCLH 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.

16.6.7 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.

16.6.8 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 can be read as I2CONSET. Writing to I2CONSET sets bits in the I2C control register that correspond to logic 1s in the value written. Conversely, writing to I2CONCLR clears bits in I2C control register that correspond to 1s in the value written.

16.6.9 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 can 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 logic 0. 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 are sufficient for most of the service routines (see the software example in this section).

16.7 Register description

Each I2C interface contains 7 registers as shown in Table 121.

Table 121. I2C register map

Name	Description	Access	Reset value[1]	I2C0 Address and name	I2C1 Address and name
I2CONSET	**I2C control set register.** If a logic 1 is written to a bit of this register, the corresponding bit in I2C control register is set. Writing a logic 0 has no effect on the corresponding bit in I2C control register.	R/W	0x00	0xE001 C000 I2C0CONSET	0xE005 C000 I2C1CONSET
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 I2C0STAT	0xE005 C004 I2C1STAT
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 is received can be read from this register.	R/W	0x00	0xE001 C008 I2C0DAT	0xE005 C008 I2C1DAT
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 I2C0ADR	0xE005 C00C I2C1ADR
I2SCLH	**SCH duty cycle register high halfword.** Determines the HIGH time of the I2C clock.	R/W	0x04	0xE001 C010 I2C0SCLH	0xE005 C010 I2C1SCLH
I2SCLL	**SCL duty cycle register low halfword.** Determines the LOW time of the I2C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I2C master and certain times used in Slave mode.	R/W	0x04	0xE001 C014 I2C0SCLL	0xE005 C014 I2C1SCLL
I2CONCLR	**I2C control clear register.** When a logic 1 is written to a bit of this register, the corresponding bit in I2C control register is cleared. Writing a logic 0 has no effect on the corresponding bit in I2C control register.	WO	NA	0xE001 C018 I2C0CONCLR	0xE005 C018 I2C1CONCLR

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

16.7.1 I²C Control set register (I2CONSET: I2C0, I2C0CONSET - 0xE001 C000 and I2C1, I2C1CONSET - 0xE005 C000)

The I2CONSET registers control setting of bits in register I2CON that controls operation of the I²C interface. Writing a logic 1 to a bit in this register causes the corresponding bit in register I²C control to be set. Writing a logic 0 has no effect.

Table 122. I²C Control set register (I2CONSET: I2C0, I2C0CONSET - address 0xE001 C000 and I2C1, I2C1CONSET - address 0xE005 C000) bit description

Bit	Symbol	Description	Reset value
1:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
2	AA	assert acknowledge flag; see text below	
3	SI	I²C interrupt flag	0
4	STO	stop flag; see text below	0
5	STA	start flag; see text below	0
6	I2EN	I²C interface enable; see text below	0
7	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

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

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

I2EN must not be used to release the I²C-bus temporarily, since when I2EN is reset, the I²C-bus status is lost. The AA flag must 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 logic 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 frees 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 is transmitted or received, it transmits a repeated Start condition. STA can be set at any time, including when the I²C interface is in an addressed slave mode.

STA can be cleared by writing logic 1 to bit STAC in register I2CONCLR. When STA is logic 0, no Start condition or repeated Start condition is generated.

If STA and STO are both set, then a Stop condition is transmitted on the I²C-bus if 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 I^2C interface to transmit a Stop condition in master mode, or recover from an error condition in slave mode. When STO is logic 1 in master mode, a Stop condition is transmitted on the I^2C-bus. When the bus detects the Stop condition, STO is cleared automatically.

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

SI is the I^2C Interrupt flag. This bit is set when the I^2C state changes. However, entering state F8 does not set SI since there is nothing for an interrupt service routine to do in that case.

While SI is set, the LOW period of the serial clock on the SCL line is stretched, and the serial transfer is suspended. When SCL is HIGH, it is unaffected by the state of the SI flag. SI must be reset by software, by writing a logic 1 to bit SIC in register I2CONCLR.

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

- The address in the slave address register is received
- The general call address is received while the general call bit (GC) in I2ADR is set
- A data byte is received while the I^2C is in the master receiver mode
- A data byte is received while the I^2C is in the addressed slave receiver mode

Bit AA can be cleared by writing logic 1 to bit AAC in register I2CONCLR. When AA is logic 0, a not acknowledge (HIGH level to SDA) is returned during the acknowledge clock pulse on the SCL line in the following situations:

- A data byte is received while the I^2C is in the master receiver mode
- A data byte is received while the I^2C is in the addressed slave receiver mode

16.7.2 I^2C Control clear register (I2CONCLR: I2C0, I2C0CONCLR - 0xE001 C018 and I2C1, I2C1CONCLR - 0xE005 C018)

The I2CONCLR registers control clearing of bits in register I2CON that controls operation of the I^2C interface. Writing a logic 1 to a bit in this register causes the corresponding bit in the I^2C control register to be cleared. Writing a logic 0 has no effect.

Table 123. I^2C Control set register (I2CONCLR: I2C0, I2C0CONCLR - address 0xE001 C018 and I2C1, I2C1CONCLR - address 0xE005 C018) bit description

Bit	Symbol	Description	Reset value
1:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
2	AAC	assert acknowledge clear bit	
3	SIC	I^2C interrupt clear bit	0
4	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Table 123. I²C Control set register (I2CONCLR: I2C0, I2C0CONCLR - address 0xE001 C018 and I2C1, I2C1CONCLR - address 0xE005 C018) bit description *...continued*

Bit	Symbol	Description	Reset value
5	STAC	start flag clear bit	0
6	I2ENC	I²C interface disable bit	0
7	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

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

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

STAC is the Start flag Clear bit. Writing a logic 1 to this bit clears bit STA in register I2CONSET. Writing logic 0 has no effect.

I2ENC is the I²C interface disable bit. Writing a logic 1 to this bit clears bit I2EN in register I2CONSET. Writing logic 0 has no effect.

16.7.3 I²C Status register (I2STAT: I2C0, I2C0STAT - 0xE001 C004 and I2C1, I2C1STAT - 0xE005 C004)

Each I²C Status register reflects the condition of the corresponding I²C interface. The I²C Status register is read-only.

Table 124. I²C Status register (I2STAT: I2C0, I2C0STAT - address 0xE001 C004 and I2C1, I2C1STAT - address 0xE005 C004) bit description

Bit	Symbol	Description	Reset value
2:0	-	unused bits and always 0	0
7:3	Status	actual status information of I²C interface	0x1F

The three least significant bits are always logic 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 bit SI is not set. All other 25 status codes correspond to defined I²C states. When any of these states are entered, bit SI is set. For a complete list of status codes, refer to Table 134 on page 135 to Table 137 on page 139 inclusive.

16.7.4 I²C Data register (I2DAT: I2C0, I2C0DAT - 0xE001 C008 and I2C1, I2C1DAT - 0xE005 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 bit SI is set. Data in I2DAT remains stable when bit SI is set. Data in I2DAT is always shifted from right to left: the first bit transmitted is the MSB (bit 7), and after a byte is received, the first bit of received data is at the MSB of I2DAT.

Table 125. I²C Data register (I2DAT: I2C0, I2C0DAT - address 0xE001 C008 and I2C1, I2C1DAT - address 0xE005 C008) bit description

Bit	Symbol	Description	Reset value
7:0	Data	holds received data values, or data to be transmitted	0

16.7.5 I²C Slave address register (I2ADR: I2C0, I2C0ADR - 0xE001 C00C and I2C1, I2C1ADR - address 0xE005 C00C)

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

Table 126. I²C Slave address register (I2ADR: I2C0, I2C0ADR - address 0xE001 C00C and I2C1, I2C1ADR - address 0xE005 C00C) bit description

Bit	Symbol	Description	Reset value
0	GC	general call enable bit	0
7:1	Address	I²C device address for slave mode	0x00

16.7.6 I²C SCL high duty cycle register (I2SCLH: I2C0, I2C0SCLH - 0xE001 C010 and I2C1, I2C1SCLH - 0xE0015 C010)

Table 127. I²C SCL high duty cycle register (I2SCLH: I2C0, I2C0SCLH - address 0xE001 C010 and I2C1, I2C1SCLH - address 0xE005 C010) bit description

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

16.7.7 I²C SCL low duty cycle register (I2SCLL: I2C0 - I2C0SCLL: 0xE001 C014; I2C1 - I2C1SCLL: 0xE0015 C014)

Table 128. I²C SCL low duty cycle register (I2SCLL: I2C0, I2C0SCLL - address 0xE001 C014 and I2C1, I2C1SCLL - address 0xE005 C014) bit description

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

16.7.8 Selecting the appropriate I²C data rate and duty cycle

Software must set values for 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):

$$I^2C_{bitfrequency} = \frac{PCLK}{I2CSCLH + I2CSCLL} \quad (5)$$

The values for I2SCLL and I2SCLH must 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 kHz to 400 kHz. Each register value must be greater than or equal to 4. Table 129 gives some examples of I²C-bus rates based on PCLK frequency and I2SCLL and I2SCLH values.

Table 129. Example of 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

16.8 Details of I²C operating modes

The four operating modes are:

- Master transmitter
- Master receiver
- Slave receiver
- Slave transmitter

Data transfers in each mode of operation are shown in Figure 35 on page 131 to Figure 39 on page 142. Table 130 lists abbreviations used in these figures when describing the I²C operating modes.

Table 130. 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 Figure 35 to Figure 39, circles are used to indicate when the serial interrupt flag is set. The numbers in the circles show the status code held in register I2STAT. 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 Table 134 on page 135 to Table 138 on page 141 inclusive.

16.8.1 Master transmitter mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 35 on page 131). Before the master transmitter mode can be entered, I2CON must be initialized as follows:

Table 131. I2CONSET used to initialize master transmitter mode

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

The I^2C rate must also be configured in registers I2SCLL and I2SCLH. To enable the I^2C block, I2EN must be set to logic 1. If bit AA is reset, the I^2C block will not acknowledge its own slave address or the general call address in the event of another device becoming master of the bus. In other words, if AA is reset, the I^2C interface cannot enter a slave mode. STA, STO, and SI must be reset.

The master transmitter mode can now be entered by setting bit STA. The I^2C logic now tests the I^2C-bus and generates a Start condition when 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) is 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). Bit SI 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 is 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 take for each of these status codes is detailed in Table 134 on page 135. After a repeated Start condition (state 0x10). The I^2C block can switch to the master receiver mode by loading I2DAT with SLA+R).

16.8.2 Master receiver mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (see Figure 36 on page 132). The transfer is initialized as in the master transmitter mode. When the Start condition is transmitted, the interrupt service routine must load I2DAT with the 7-bit slave address and the data direction bit (SLA+R). Before the serial transfer can continue, bit SI in I2CON must be cleared.

When the slave address and the data direction bit have been transmitted and an acknowledgment bit is 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 take for each of these status codes is detailed in Table 135 on page 136. After a repeated Start condition (state 0x10), the I^2C block can switch to the master transmitter mode by loading I2DAT with SLA+W.

16.8.3 Slave receiver mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 37 on page 133). To initiate the slave receiver mode, I2ADR and I2CON must be loaded as follows:

Table 132. I2C0ADR and I2C1ADR usage in slave receiver mode

Bit:	7	6	5	4	3	2	1	0
Symbol:	own slave 7-bit address							GC

The upper 7 bits are the address to which the I^2C block responds when addressed by a master. If the LSB (GC) is set, the I^2C block responds to the general call address (0x00); otherwise it ignores the general call address.

Table 133. I2C0CONSET and I2C1CONSET used to initialize slave receiver mode

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

The I^2C-bus rate settings do not affect the I^2C block in the slave mode. To enable the I^2C block, 2EN must be set to logic 1. To enable the I^2C block to acknowledge its own slave address or the general call address, bit AA must be set. STA, STO, and SI must be reset.

When I2ADR and I2CON are initialized, the I^2C block waits until addressed by its own slave address followed by the data direction bit which must be logic 0 (W) for the I^2C block to operate in the slave receiver mode. After its own slave address and bit W are 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 136 on page 137. The slave receiver mode can also be entered if arbitration is lost while the I^2C block is in the master mode (see status 0x68 and 0x78).

If bit AA is reset during a transfer, the I^2C block returns a not acknowledge (logic 1) to SDA after the next received data byte. While AA is reset, the I^2C block does not respond to its own slave address or a general call address. However, the I^2C-bus is still monitored and address recognition can be resumed at any time by setting AA. This means that bit AA can be used to isolate the I^2C block from the I^2C-bus temporarily.

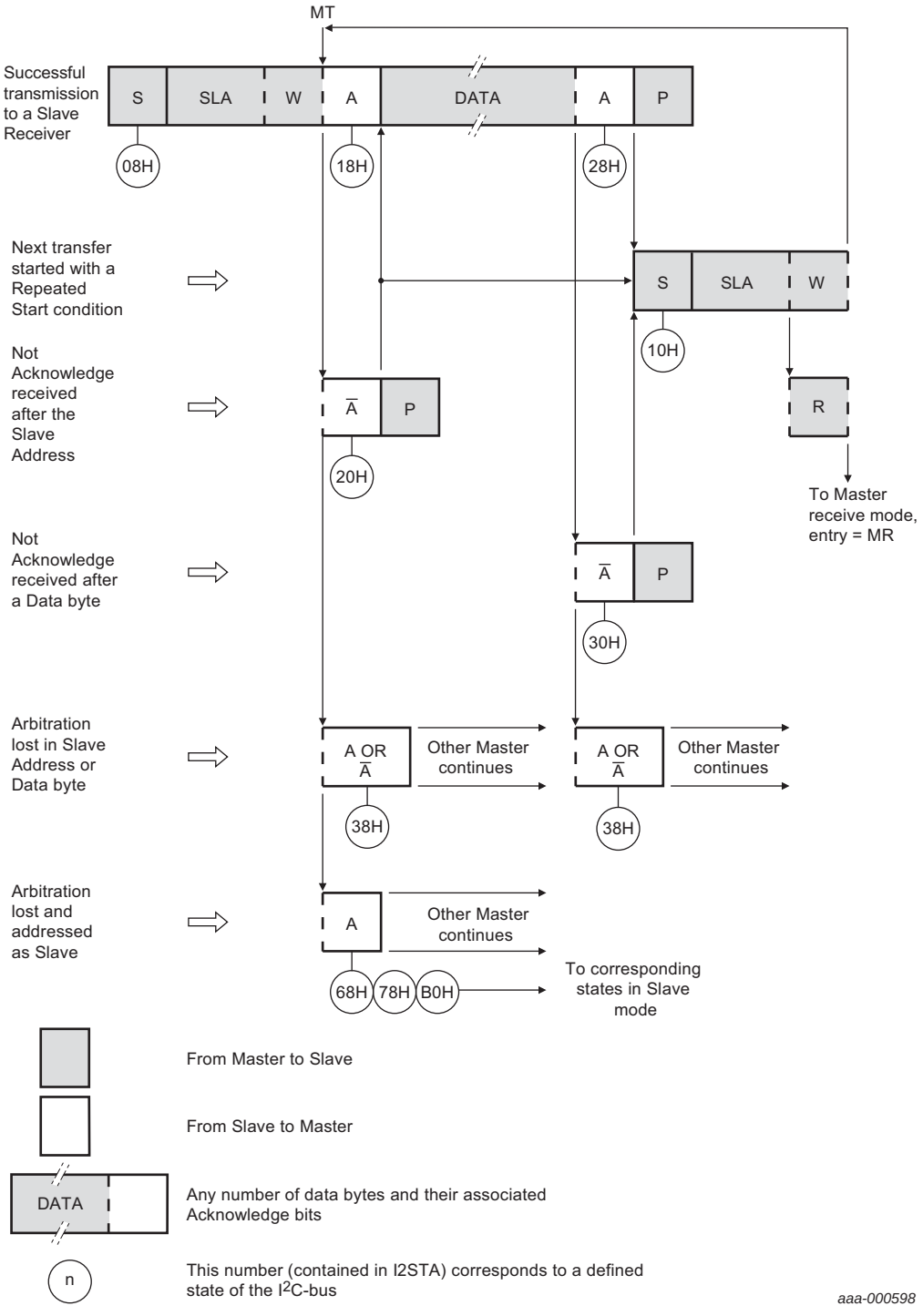


Fig 35. Format and states in the master transmitter mode

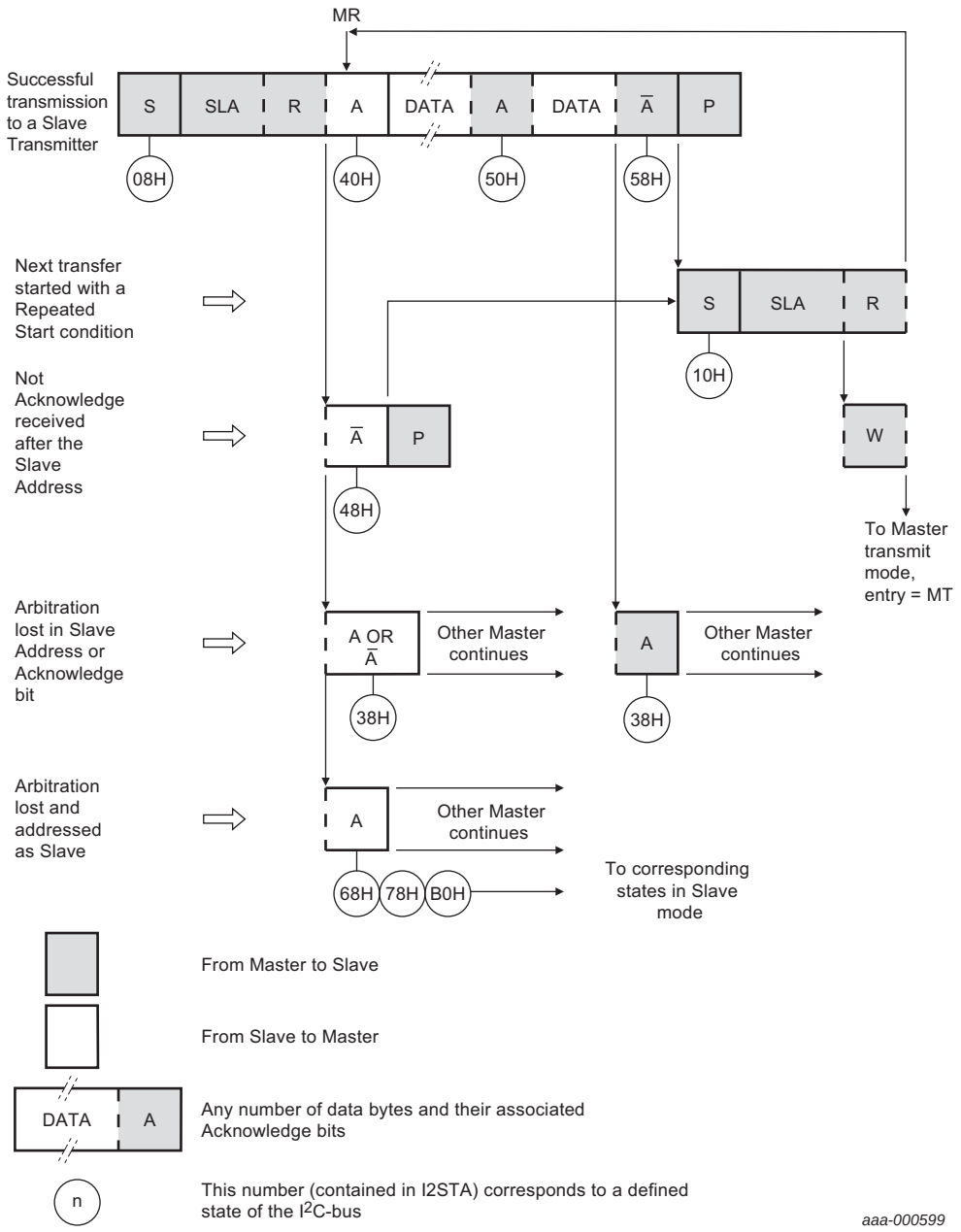


Fig 36. Format and states in the master receiver mode

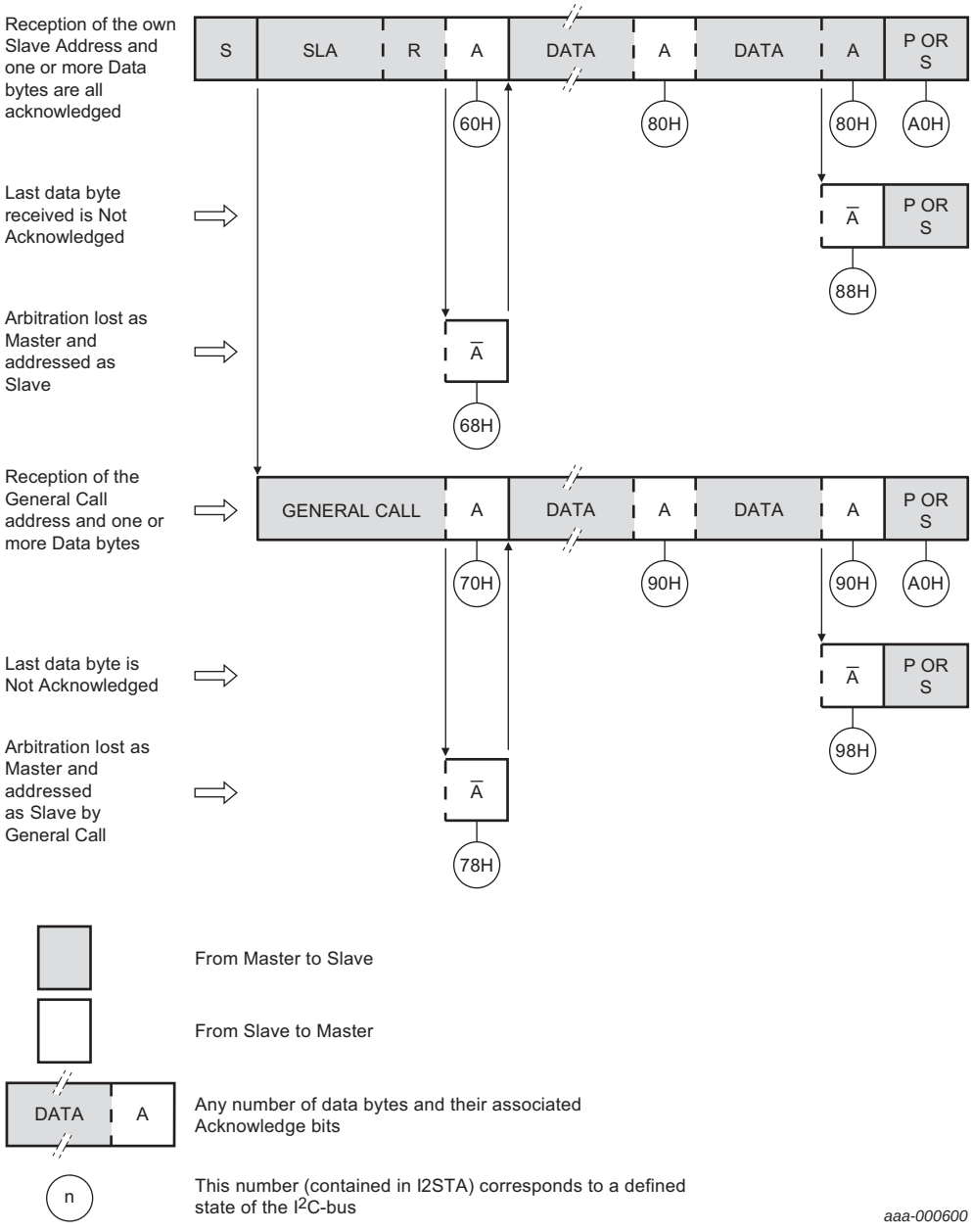


Fig 37. Format and states in the slave receiver mode

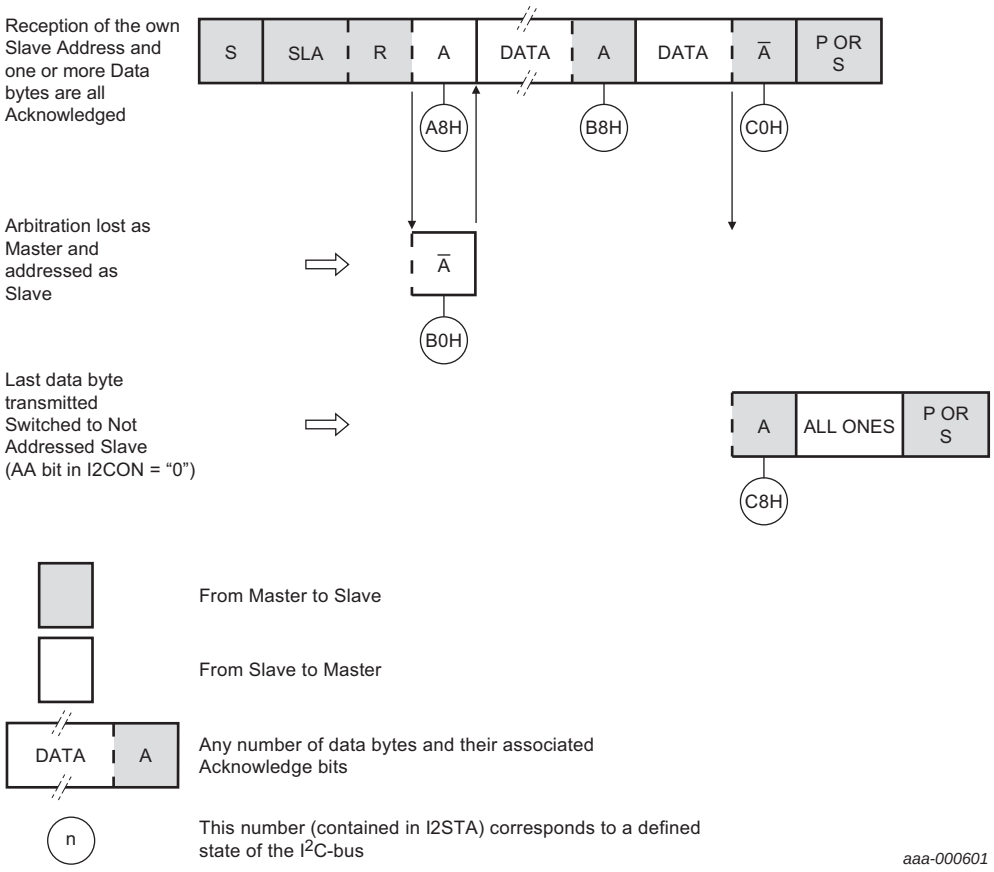


Fig 38. Format and states in the slave transmitter mode

16.8.4 Slave transmitter mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 38). Data transfer is initialized as in the slave receiver mode. When I2ADR and I2CON have been initialized, the I^2C block waits until addressed by its own slave address followed by the data direction bit which must be logic 1 (R) for the I^2C block to operate in the slave transmitter mode. After its own slave address and bit R are 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 take for each of these status codes is detailed in Table 137 on page 139. The slave transmitter mode can also be entered if arbitration is lost while the I^2C block is in the master mode (see state 0xB0).

If bit AA is reset during a transfer, the I^2C block transmits the last byte of the transfer and enters state 0xC0 or 0xC8. The I^2C block is switched to the not-addressed slave mode and ignores the master receiver if it continues the transfer. Thus the master receiver receives all logic 1s as serial data. While AA is reset, the I^2C block does not respond to its own slave address or a general call address. However, the I^2C-bus is still monitored, and address recognition can be resumed at any time by setting AA. This means that bit AA can be used to isolate the I^2C block from the I^2C-bus temporarily.

Table 134. Master transmitter mode

Status code (I2CSTAT)	Status of 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 is transmitted	load SLA+W	X	0	0	X	SLA+W transmitted; ACK bit received
0x10	a repeated start condition is transmitted	load SLA+W or	X	0	0	X	as above
		load SLA+R	X	0	0	X	SLA+W transmitted; the I²C block is switched to MST/REC mode
0x18	SLA+W transmitted; ACK received	load data byte or	0	0	0	X	data byte transmitted; ACK bit received
		no I2DAT action or	1	0	0	X	repeated start transmitted
		no I2DAT action or	0	1	0	X	stop condition transmitted; STO flag is reset
		no I2DAT action	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset
0x20	SLA+W transmitted; NOT ACK received	load data byte or	0	0	0	X	data byte transmitted; ACK bit received
		no I2DAT action or	1	0	0	X	repeated start transmitted
		no I2DAT action or	0	1	0	X	stop condition transmitted; STO flag reset
		no I2DAT action	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset
0x28	data byte in I2DAT transmitted; ACK received	load data byte or	0	0	0	X	data byte transmitted; ACKbit received
		no I2DAT action or	1	0	0	X	repeated start transmitted
		no I2DAT action or	0	1	0	X	stop condition transmitted; STO flag reset
		no I2DAT action	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset
0x30	data byte in I2DAT transmitted; NOT ACK received	load data byte or	0	0	0	X	data byte transmitted; ACKbit received
		no I2DAT action or	1	0	0	X	repeated start transmitted
		no I2DAT action or	0	1	0	X	stop condition transmitted; STO flag reset
		no I2DAT action	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset
0x38	arbitration lost in SLA+R/W or data bytes	no I2DAT action or	0	0	0	X	I²C-bus released; not addressed slave entered
		no I2DAT action	1	0	0	X	a start condition is transmitted when bus is free

Table 135. Master receiver mode

Status code (I2CSTAT)	Status of I^2C-bus and hardware	Application software response					Next action taken by I^2C hardware
		To/from I2DAT	To I2CON				
			STA	STO	SI	AA	
0x08	a start condition is transmitted	load SLA+R	X	0	0	X	SLA+R transmitted; ACK bit received
0x10	a repeated start condition is transmitted	load SLA+R or	X	0	0	X	as above
		load SLA+W	X	0	0	X	SLA+W transmitted; the I^2C block switched to MST/TRX mode
0x38	arbitration lost in NOT ACK bit	no I2DAT action or	0	0	0	X	I^2C-bus released; the I^2C block enters a slave mode
		no I2DAT action	1	0	0	X	a start condition is transmitted when the bus is free
0x40	SLA+R transmitted; ACK received	no I2DAT action or	0	0	0	0	data byte received; NOT ACK bit returned
		no I2DAT action	0	0	0	1	data byte received; ACK bit returned
0x48	SLA+R transmitted; NOT ACK received	no I2DAT action or	1	0	0	X	repeated start condition transmitted
		no I2DAT action or	0	1	0	X	stop condition transmitted; STO flag reset
		no I2DAT action	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset
0x50	data byte received; ACK returned	read data byte or	0	0	0	0	data byte received; NOT ACK bit returned
		read data byte	0	0	0	1	data byte received; ACK bit returned
0x58	data byte received; NOT ACK returned	read data byte or	1	0	0	X	repeated start condition transmitted
		read data byte or	0	1	0	X	stop condition transmitted; STO flag reset
		read data byte	1	1	0	X	stop condition followed by a start condition transmitted; STO flag reset

Table 136. Slave receiver mode

Status code (I2CSTAT)	Status of 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 received; ACK returned	no I2DAT action or	X	0	0	0	data byte received and NOT ACK returned
		no I2DAT action	X	0	0	1	data byte received and ACK returned
0x68	arbitration lost in SLA+R/W as master; own SLA+W received, ACK returned	no I2DAT action or	X	0	0	0	data byte received and NOT ACK returned
		no I2DAT action	X	0	0	1	data byte received and ACK returned
0x70	general call address (0x00) received; ACK returned	no I2DAT action or	X	0	0	0	data byte received and NOT ACK returned
		no I2DAT action	X	0	0	1	data byte received and ACK returned
0x78	arbitration lost in SLA+R/W as master; general call address received, ACK returned	no I2DAT action or	X	0	0	0	data byte received and NOT ACK returned
		no I2DAT action	X	0	0	1	data byte received and ACK returned
0x80	previously addressed with own SLV address; data received; ACK returned	read data byte or	X	0	0	0	data byte received and NOT ACK returned
		read data byte	X	0	0	1	data byte received and ACK returned
0x88	previously addressed with own SLA; data byte received; NOT ACK 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 recognized; general call address 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 is transmitted when the bus is free.
		read data byte	1	0	0	1	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR[0] = logic 1. A start condition is transmitted when the bus is free.
0x90	previously addressed with general call; data byte received; ACK returned	read data byte or	X	0	0	0	data byte received and NOT ACK returned
		read data byte	X	0	0	1	data byte received and ACK returned

Table 136. Slave receiver mode *…continued*

Status code (I2CSTAT)	Status of I^2C-bus and hardware	Application software response					Next action taken by I^2C hardware
		To/from I2DAT	To I2CON				
			STA	STO	SI	AA	
0x98	previously addressed with general call; data byte received; NOT ACK 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 recognized; general call address 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 is transmitted when the bus is free.
		read data byte	1	0	0	1	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR[0] = logic 1. A start condition is transmitted when the bus is free.
0xA0	a stop condition or repeated start condition 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 recognized; general call address 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 is transmitted when the bus is free.
		no STDAT action	1	0	0	1	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR[0] = logic 1. A start condition is transmitted when the bus is free.

Table 137. Slave transmitter mode

Status code (I2CSTAT)	Status of 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 received; ACK returned	load data byte	X	0	0	0	last data byte transmitted and ACK bit received
		load data byte	X	0	0	1	data byte transmitted; ACK received
0xB0	arbitration lost in SLA+R/W as master; own SLA+R received, ACK returned	load data byte	X	0	0	0	last data byte transmitted and ACK bit received
		load data byte	X	0	0	1	data byte transmitted; ACK bit received
0xB8	data byte in I2DAT transmitted; ACK received	load data byte	X	0	0	0	last data byte transmitted and ACK bit received
		load data byte	X	0	0	1	data byte transmitted; ACK bit received
0xC0	data byte in I2DAT transmitted; NOT ACK received	no I2DAT action	0	0	0	0	switched to not-addressed SLV mode; no recognition of own SLA or general call address
		no I2DAT action	0	0	0	1	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR[0] = logic 1
		no I2DAT action	1	0	0	0	switched to not-addressed SLV mode; no recognition of own SLA or general call address. A start condition is transmitted when the bus is free.
		no I2DAT action	1	0	0	1	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR[0] = logic 1. A start condition is transmitted when the bus is free.
0xC8	last data byte in I2DAT is transmitted (AA = 0); ACK received	no I2DAT action	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 recognized; general call address recognized if I2ADR[0] = logic 1
		No I2DAT action	1	0	0	0	switched to not-addressed SLV mode; no recognition of own SLA or general call address. A start condition is transmitted when the bus is free.
		no I2DAT action	1	0	0	01	switched to not-addressed SLV mode; own SLA recognized; general call address recognized if I2ADR.0 = logic 1. A start condition is transmitted when the bus is free.

16.8.5 Miscellaneous states

There are two I2STAT codes that do not correspond to a defined I²C hardware state (see Table 138 on page 141) and are discussed below.

16.8.6 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.

16.8.7 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 can 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 138. Miscellaneous states

Status code (I2CSTAT)	Status of 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 with 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 internal hardware is affected in the MST or addressed SLV modes. In all cases, bus is released and I²C block is switched to the not-addressed SLV mode. STO is reset.

16.8.8 Some special cases

The I²C hardware has facilities to handle the following special cases that can occur during a serial transfer:

16.8.9 Simultaneous repeated Start conditions from two masters

A repeated Start condition can 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 39 on page 142). 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 releases the bus, and no interrupt request is generated. If another master frees the bus by generating a Stop condition, the I²C block transmits a normal Start condition (state 0x08), and a retry of the total serial data transfer can commence.

16.8.10 Data transfer after loss of arbitration

Arbitration can be lost in the master transmitter and master receiver modes (see Figure 33 on page 121). Loss of arbitration is indicated by the following states in I2STAT; 0x38, 0x68, 0x78, and 0xB0 (see Figure 35 on page 131 and Figure 36 on page 132).

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.

16.8.11 Forced access to the I²C-bus

In some applications, it can 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 40).

16.8.12 I²C-bus obstructed by a LOW level on SCL or SDA

If SDA or SCL is pulled LOW by an uncontrolled source, an I²C-bus hang-up occurs. 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. If 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 (for example, a slave device out of bit synchronization), the problem can be solved by transmitting additional clock pulses on the SCL line (see Figure 41). 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.

16.8.13 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. If 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 can 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 138 on page 141.

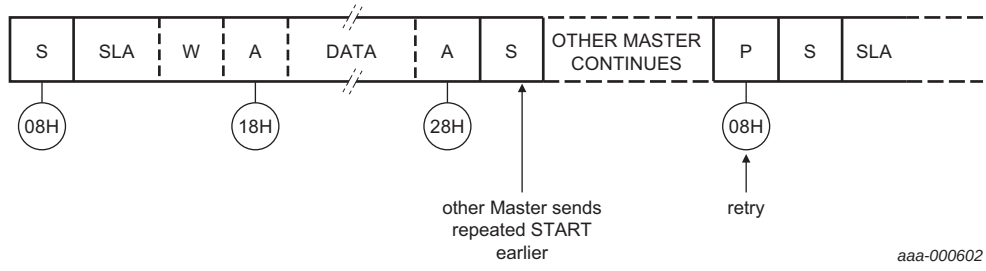


Fig 39. Simultaneous repeated Start conditions from two masters

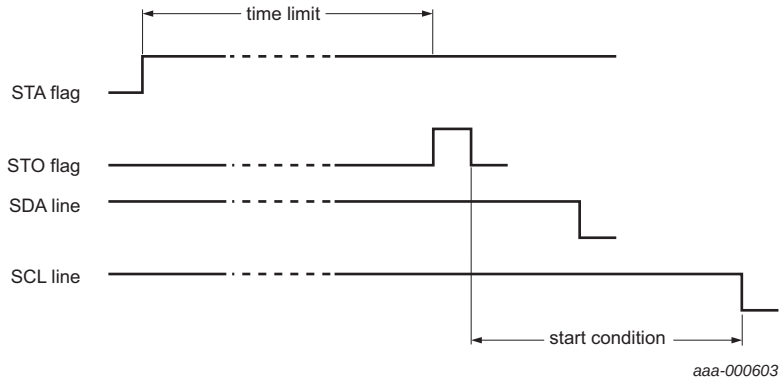
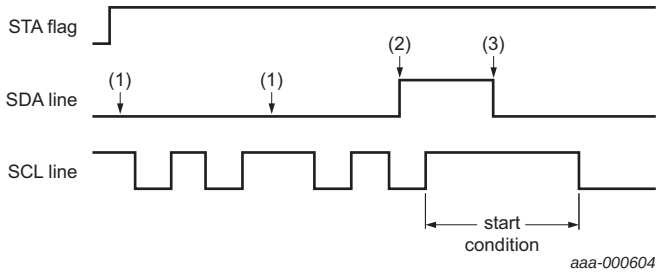


Fig 40. Forced access to a busy I²C-bus



(1) Unsuccessful attempt to send a Start condition.
(2) SDA line is released.
(3) Successful attempt to send a Start condition. State 08H is entered.

Fig 41. Recovering from a bus obstruction caused by a LOW level on SDA

16.8.14 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

16.8.15 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
- 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.

16.8.16 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.

16.8.17 The state service routines

Each state routine is part of the I²C interrupt routine and handles one of the 26 states.

16.8.18 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 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.

16.9 Software example

16.9.1 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.

16.9.2 Start master transmit function

Begin a master transmit operation by setting up the buffer, pointer, and data count, then initiate 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 bit STA.
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.

16.9.3 Start master receive function

Begin a master receive operation by setting up the buffer, pointer, and data count, then initiate 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 bit STA.
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.

16.9.4 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.

16.9.5 Non mode-specific states

16.9.5.1 State: 0x00

Bus error. Enter not-addressed slave mode and release bus.

1. Write 0x14 to I2CONSET to set the STO and AA bits.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.5.2 Master states

State 08 and state 10 are for both master transmit and master receive modes. Bit R/W decides whether the next state is within master transmit mode or master receive mode.

16.9.5.3 State: 0x08

A Start condition is transmitted. The slave address + bit R/W is transmitted, an ACK bit is received.

1. Write slave address with bit R/W to I2DAT.
2. Write 0x04 to I2CONSET to set bit AA.
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.

16.9.5.4 State: 0x10

A repeated Start condition is transmitted. The slave address + bit R/W is transmitted, an ACK bit is received.

1. Write slave address with bit R/W to I2DAT.
2. Write 0x04 to I2CONSET to set bit AA.
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.

16.9.6 Master transmitter states

16.9.6.1 State: 0x18

Previous state was state 8 or state 10, slave address + write is transmitted, ACK is received. The first data byte is transmitted, an ACK bit is received.

1. Load I2DAT with first data byte from master transmit buffer.
2. Write 0x04 to I2CONSET to set bit AA.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment master transmit buffer pointer.
5. Exit.

16.9.6.2 State: 0x20

Slave address + write is transmitted, NOT ACK is received. A Stop condition is 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.

16.9.6.3 State: 0x28

Data is transmitted, ACK is 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 bit AA.
7. Write 0x08 to I2CONCLR to clear the SI flag.
8. Increment master transmit buffer pointer
9. Exit.

16.9.6.4 State: 0x30

Data is transmitted, NOT ACK received. A Stop condition is 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.

16.9.6.5 State: 0x38

Arbitration is lost during slave address + write or data. The bus is released and not-addressed slave mode is entered. A new Start condition is 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.

16.9.7 Master receive states

16.9.7.1 State: 0x40

Previous state was state 08 or state 10. Slave address + read is transmitted, ACK is received. Data is received and ACK returned.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.7.2 State: 0x48

Slave address + read is transmitted, NOT ACK is received. A Stop condition is 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.

16.9.7.3 State: 0x50

Data is received, ACK is returned. Data is read from I2DAT. Additional data is received. If this is the last data byte, then NOT ACK is returned, otherwise ACK is 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 bit AA.
4. Exit.
5. Write 0x04 to I2CONSET to set bit AA.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment master receive buffer pointer.
8. Exit.

16.9.7.4 State: 0x58

Data is received, NOT ACK is returned. Data is read from I2DAT. A Stop condition is 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.

16.9.8 Slave receiver states

16.9.8.1 State: 0x60

Own slave address + write is received, ACK is returned. Data is received and ACK returned.

1. Write 0x04 to I2CONSET to set bit AA.
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.

16.9.8.2 State: 0x68

Arbitration is lost in slave address and bit R/W as bus master. Own slave address + write is received, ACK is returned. Data is received and ACK is 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.

16.9.8.3 State: 0x70

General call is received, ACK is returned. Data is received and ACK returned.

1. Write 0x04 to I2CONSET to set bit AA.
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.

16.9.8.4 State: 0x78

Arbitration is lost in slave address + bit R/W as bus master. General call is received and ACK is returned. Data is 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.

16.9.8.5 State: 0x80

Previously addressed with own slave address. Data is received and ACK is returned. Additional data is 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 bit AA.
4. Exit.
5. Write 0x04 to I2CONSET to set bit AA.
6. Write 0x08 to I2CONCLR to clear the SI flag.
7. Increment slave receive buffer pointer.
8. Exit.

16.9.8.6 State: 0x88

Previously addressed with own slave address. Data is received and NOT ACK is returned. Received data is not saved. Not-addressed slave mode is entered.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.8.7 State: 0x90

Previously addressed with general call. Data is received, ACK is returned. Received data is saved. Only the first data byte is received with ACK. Additional data is 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 bit AA.
3. Exit.

16.9.8.8 State: 0x98

Previously addressed with general call. Data is received, NOT ACK is returned. Received data is not saved. Not addressed slave mode is entered.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.8.9 State: 0xA0

A Stop condition or repeated Start is received, while still addressed as a slave. Data is not saved. Not-addressed Slave mode is entered.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.9 Slave transmitter states

16.9.9.1 State: 0xA8

Own slave address + read is received, ACK is returned. Data is transmitted, ACK bit is received.

1. Load I2DAT from slave transmit buffer with first data byte.
2. Write 0x04 to I2CONSET to set bit AA.
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.

16.9.9.2 State: 0xB0

Arbitration lost in slave address and bit R/W as bus master. Own slave address + read is received, ACK is returned. Data is transmitted, ACK bit is 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.

16.9.9.3 State: 0xB8

Data is transmitted, ACK is received. Data is transmitted, ACK bit is received.

1. Load I2DAT from slave transmit buffer with data byte.
2. Write 0x04 to I2CONSET to set bit AA.
3. Write 0x08 to I2CONCLR to clear the SI flag.
4. Increment slave transmit buffer pointer.
5. Exit.

16.9.9.4 State: 0xC0

Data is transmitted, NOT ACK is received. Not-addressed slave mode is entered.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

16.9.9.5 State: 0xC8

The last data byte is transmitted, ACK is received. Not-addressed slave mode is entered.

1. Write 0x04 to I2CONSET to set bit AA.
2. Write 0x08 to I2CONCLR to clear the SI flag.
3. Exit.

17. SPI Interface SPI0

17.1 Features

- Single complete and independent SPI controller
- Compliant with Serial Peripheral Interface (SPI) specification
- Synchronous, serial, full duplex communication
- SPI master only
- Maximum data bit rate of one eighth of the input clock rate
- 8 to 16 bits per transfer

17.2 Description

17.2.1 SPI overview

SPI is a full duplex serial interface. It can handle multiple slaves connected to a given bus. Only a single master and a single slave can communicate on the interface during a given data transfer. During a data transfer, the master always sends 8 to 16 bits of data to the slave, and the slave always sends a byte of data to the master.

17.2.2 SPI data transfers

Figure 42 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 timing diagram 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 the following two points:

- the SPI is illustrated with CPOL set to both 0 and 1
- activation and de-activation of the SSEL signal: when CPHA = 1, the SSEL signal is always inactive between data transfers; this is not guaranteed when CPHA = 0 (the signal can remain active)

.

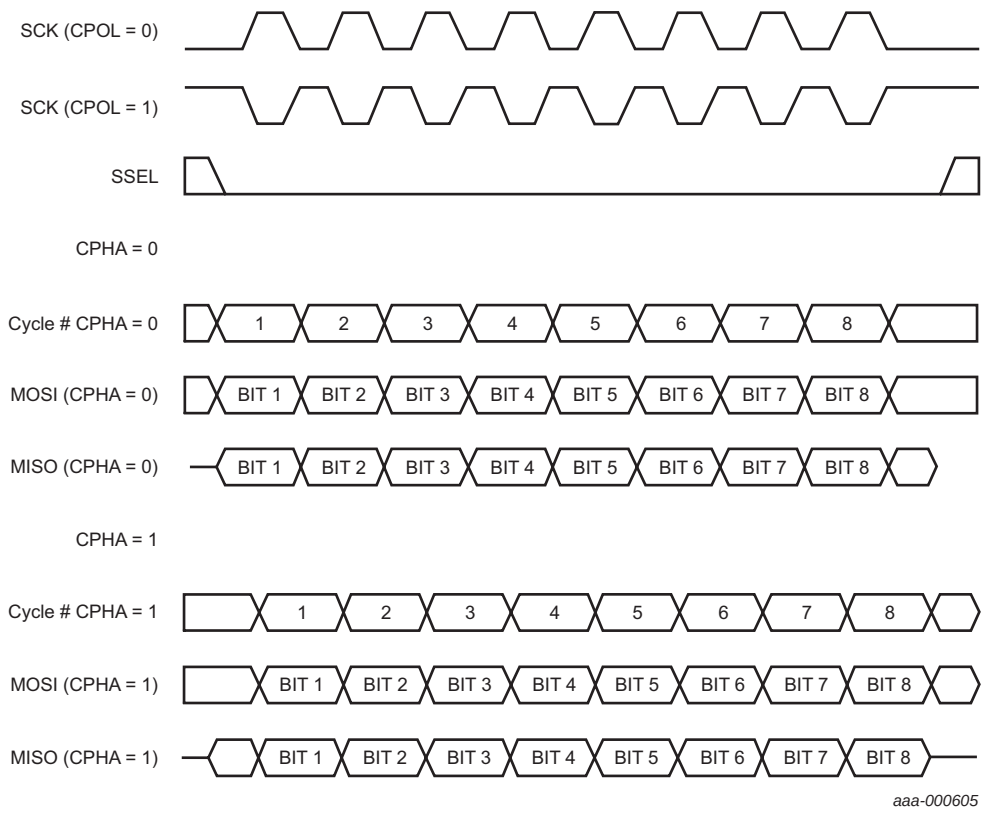


Fig 42. SPI data transfer format (CPHA = 0 and CPHA = 1)

The data and clock phase relationships are summarized in Table 139. 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 139. SPI data to clock phase relationship

CPOL and CPHA settings	First data driven	Other data driven	Data sampled
CPOL = 0, CPHA = 0	before 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	before 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 logic 0, the transfer starts when the SSEL signal is active, and ends when SSEL is inactive. When a device is a slave, and CPHA is set to logic 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.

17.2.3 General information

There are four registers that control the SPI peripheral. They are described in detail in Section 17.4 “Register description” on page 155.

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 before 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 which is indicated by bit SPIF. The remaining bits in the register are exception condition indicators. These exceptions are 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 must 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 must be set before a transfer takes 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 selected by the SSEL signal being active.

17.2.4 Master operation

The following sequence describes how one must process a data transfer with the SPI block when it is set up to be 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 be transmitted to the SPI data register. This write starts the SPI data transfer.
4. Wait for bit SPIF in the SPI status register to be set to logic 1. Bit SPIF 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 be transmitted.

Remark: In order to clear the SPIF status bit, a read or write of the SPI data register is required. 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.

17.2.5 Exception conditions

17.2.6 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 bit SPIF in the status register being active. When a transfer completes, the SPI block must move the received data to the read buffer. If bit SPIF is active (the read buffer is full), the new receive data is lost, and the read overrun (ROVR) bit in the status register is activated.

17.2.7 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 been read when the SPIF status is active. If the SPI data register is written in this time frame, the write data is lost, and the write collision (WCOL) bit in the status register is activated.

17.2.8 Mode fault

The SSEL signal must always be inactive when the SPI block is a master. If the SSEL signal is active, when the SPI block is a master, it 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 is activated, the SPI signal drivers are de-activated, and the SPI mode is changed to be a slave.

17.2.9 Slave abort

If the SSEL signal is inactive before the transfer is complete, a slave transfer is considered to be aborted. If a slave aborts, the transmit and receive data for the transfer that was in progress are lost, and the slave abort (ABRT) bit in the status register is activated.

17.3 Pin description

Table 140. SPI pin description

Pin Name	Type	Pin Description
SCK0	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 3-stated.
SSEL0	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 can be driven by a simple general-purpose I/O under software control. On pin MPT612, SSEL0 can be used for a different function when the SPI0 interface is only used in Master mode. For example, the pin hosting the SSEL0 function can be configured as an output digital GPIO pin or used to select one of the match outputs.
MISO0	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.
MOSI0	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.

17.4 Register description

The SPI contains 5 registers as shown in Table 141. All registers are byte, halfword and word accessible.

Table 141. SPI register map

Name	Description	Access	Reset value[1]	Address
S0SPCR	SPI control register. Controls SPI operation.	R/W	0x00	0xE002 0000
S0SPSR	SPI status register. Shows status of the SPI.	RO	0x00	0xE002 0004
S0SPDR	SPI data register. Bi-directional register provides SPI transmit and receive data. Transmit data is provided to the SPI0 by writing to this register. Data received by the SPI0 can be read from this register.	R/W	0x00	0xE002 0008
S0SPCCR	SPI clock counter register. Controls the frequency of a master's SCK0.	R/W	0x00	0xE002 000C
S0SPINT	SP interrupt flag. Contains the interrupt flag for the SPI interface.	R/W	0x00	0xE002 001C

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

17.4.1 SPI Control register (S0SPCR - 0xE002 0000)

Register S0SPCR controls the operation of the SPI0 as per the configuration bits setting.

Table 142: SPI Control register (S0SPCR - address 0xE002 0000) bit description

Bit	Symbol	Value	Description	Reset value
1:0	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
2	BitEnable	0	SPI controller sends and receives 8 bits of data per transfer	0
3	CPHA		clock phase control determines relationship between data and clock on SPI transfers, and controls when a slave transfer is defined as starting and ending.	0
		0	data is sampled on the first clock edge of SCK. A transfer starts and ends with activation and deactivation of the SSEL signal.	
		1	data is sampled on SCK second clock edge. A transfer starts with the first clock edge, and ends with the last sampling edge when SSEL signal is active.	
4	CPOL		clock polarity control	0
		0	SCK is active HIGH	
		1	SCK is active LOW	
5	MSTR		Master mode select	0
		0	SPI operates in Slave mode	
		1	SPI operates in Master mode	
6	LSBF		LSB first. Controls which direction each byte is shifted when transferred	0
		0	SPI data is transferred MSB (bit 7) first	
		1	SPI data is transferred LSB (bit 0) first	
7	SPIE		serial peripheral interrupt enable	0
		0	SPI interrupts are inhibited	
		1	a hardware interrupt is generated each time the SPIF or MODF bits are activated	
11:8	BITS		when bit 2 of this register is logic 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

17.4.2 SPI Status register (S0SPSR - 0xE002 0004)

Register S0SPSR controls the operation of the SPI0 as per the configuration bits setting.

Table 143: SPI Status register (S0SPSR - address 0xE002 0004) bit description

Bit	Symbol	Description	Reset value
2:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
3	ABRT	slave abort. If logic 1, indicates that a slave abort has occurred. Cleared by reading this register.	0
4	MODF	Mode fault. If logic 1, indicates that a mode fault error has occurred. Cleared by reading this register, then writing the SPI0 control register.	0
5	ROVR	read overrun. If logic 1, indicates that a read overrun has occurred. Cleared by reading this register.	0
6	WCOL	write collision. If logic 1, indicates that a write collision has occurred. Cleared by reading this register, then accessing the SPI data register.	0
7	SPIF	SPI transfer complete flag. If logic 1, indicates when a SPI data transfer is complete. When a master, set at end of last cycle of the transfer. When a slave, set on the last data sampling edge of SCK. Cleared by first reading this register, then accessing the SPI data register. **Remark:** this is not the SPI interrupt flag. This flag is found in register SPINT.	0

17.4.3 SPI Data register (S0SPDR - 0xE002 0008)

This bidirectional 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 starts a SPI data transfer. Writes to this register are blocked from when a data transfer starts to when the SPIF status bit is set, and the status register has not been read.

Table 144: SPI Data register (S0SPDR - address 0xE002 0008) bit description

Bit	Symbol	Description	Reset value
7:0	DataLow	SPI bidirectional data port	0x00
15:8	DataHigh	if bit 2 of SPCR is logic 1 and bits 11:8 are other than 1000, some or all of these bits contain the additional transmit and receive bits. When less than 16 bits are selected, the more significant among these bits read as logic 0s.	0x00

17.4.4 SPI Clock counter register (S0SPCCR - 0xE002 000C)

This register controls the frequency of a master's SCK. The register indicates the number of PCLK cycles that make up an SPI clock. The value of this register must always be an even number. As a result, bit 0 must always be logic 0. The value of the register must also always be greater than or equal to 8. Violations can result in unpredictable behavior.

Table 145: SPI Clock counter register (S0SPCCR - address 0xE002 000C) bit description

Bit	Symbol	Description	Reset value
7:0	Counter	SPI0 clock counter setting	0x00

The SPI0 rate can be calculated as: PCLK / SPCCR0 value. The PCLK rate is CCLK / APB divider rate as determined by register APBDIV contents.

17.4.5 SPI Interrupt register (S0SPINT - 0xE002 001C)

This register contains the interrupt flag for the SPI0 interface.

Table 146: SPI Interrupt register (S0SPINT - address 0xE002 001C) bit description

Bit	Symbol	Description	Reset value
0	SPI Interrupt Flag	SPI interrupt flag. Set by SPI interface to generate an interrupt. Cleared by writing a logic 1 to this bit. **Remark:** set once when SPIE = 1 and at least one SPIF or bit WCOL is logic 1. However, only when SPI interrupt bit is set and SPI0 interrupt is enabled in VIC can SPI-based interrupt be processed by interrupt handling software.	0
7:1	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

17.5 Architecture

The block diagram of the SPI solution implemented in SPI0 interface is shown in Figure 43.

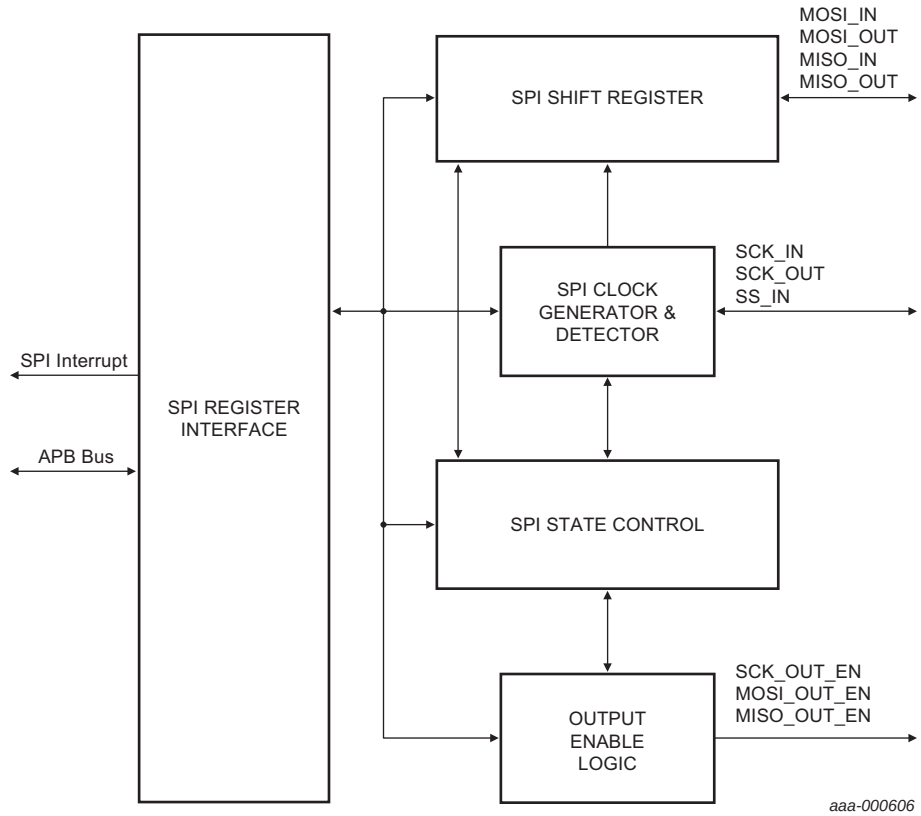


Fig 43. SPI block diagram

18. SPI/SSP interface SPI1

18.1 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

18.2 Description

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

Table 147. SSP pin descriptions

Pin Name	Type	Interface pin name/function			Pin description
		SPI	SSI	Microwire	
SCK1	I/O	SCK	CLK	SK	**serial clock.** SCK/CLK/SK is a clock signal used to synchronize 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. SCK1 only switches during a data transfer. Any other time, SSP either holds it in its inactive state, or does not drive it (leaves it in HIGH impedance state).

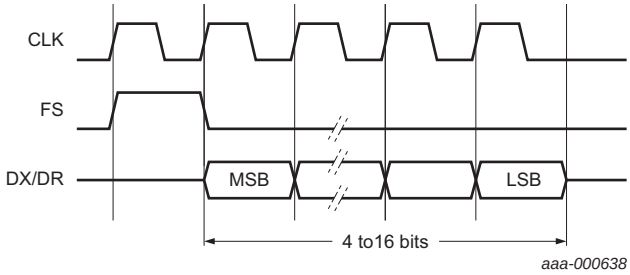
Table 147. SSP pin descriptions *…continued*

Pin Name	Type	Interface pin name/function			Pin description
		SPI	SSI	Microwire	
SSEL1	I/O	SSEL	FS	CS	**Slave Select/Frame Sync/Chip Select.** If the SSP is a bus master, it drives this signal 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. If 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.
MISO1	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. If the SSP is a slave and not selected by SSEL, it does not drive this signal (leaves it in HIGH impedance state).
MOSI1	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.

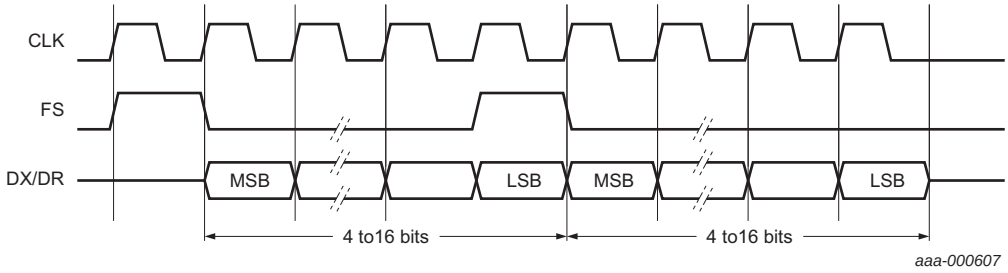
18.3 Bus description

18.3.1 Texas Instruments Synchronous Serial (SSI) frame format

Figure 44 shows the 4-wire Texas Instruments synchronous serial frame format supported by the SSP module.



a. Single frame transfer



b. Continuous/back-to-back two frames transfer

Fig 44. Texas Instruments synchronous serial frame format

For a device configured as a master in this mode, CLK and FS are forced LOW, and the transmit data line DX is tri-stated 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 pin DX. Likewise, the MSB of the received data is shifted onto pin DR 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.

18.3.2 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 bits CPOL and CPHA within the SSPCR0 control register.

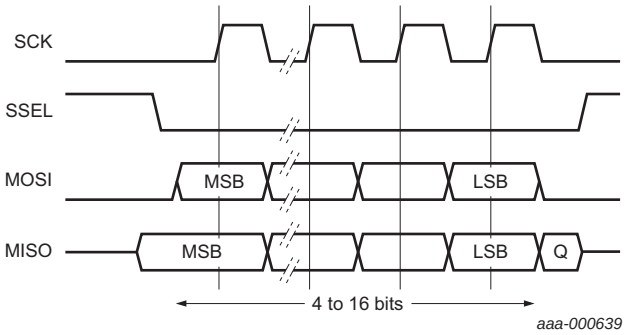
18.3.3 Clock Polarity (CPOL) and Clock Phase (CPHA) control

When the CPOL clock polarity control bit is LOW, it produces a steady state LOW value on pin SCK. If the CPOL clock polarity control bit is HIGH, a steady state HIGH value is placed on pin CLK when data is not being transferred.

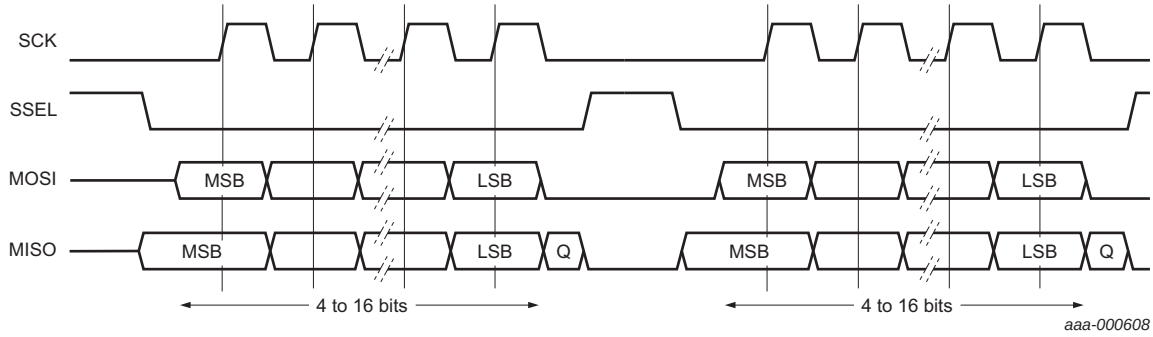
The CPHA control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the CPHA phase control bit is LOW, data is captured on the first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

18.3.4 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 45.



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



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

Fig 45. Motorola SPI frame formats with CPOL = 0 and CPHA = 0

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 enables slave data onto the MISO input line of the master. Master's MOSI is enabled.

One half SCK period later, valid master data is transferred to pin MOSI. 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.

If 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 is 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 0. Therefore the master device must raise the level on pin SSEL of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, pin SSEL is returned to its idle state one SCK period after the last bit is captured.

18.3.5 SPI format with CPOL = 0, CPHA = 1

The transfer signal sequence for SPI format with CPOL = 0, CPHA = 1 is shown in Figure 46, which covers both single and continuous transfers.

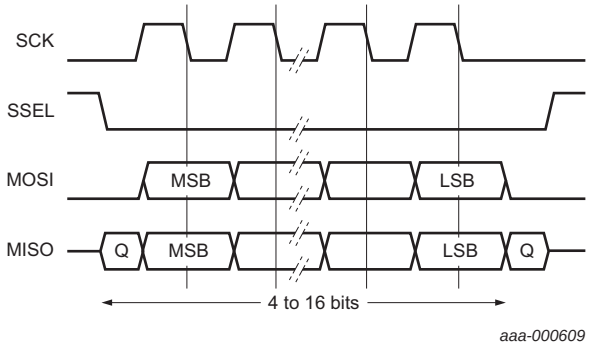


Fig 46. Motorola SPI frame format (single transfer) with CPOL = 0 and CPHA = 1

In this configuration, during idle periods:

- 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 pin MOSI 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.

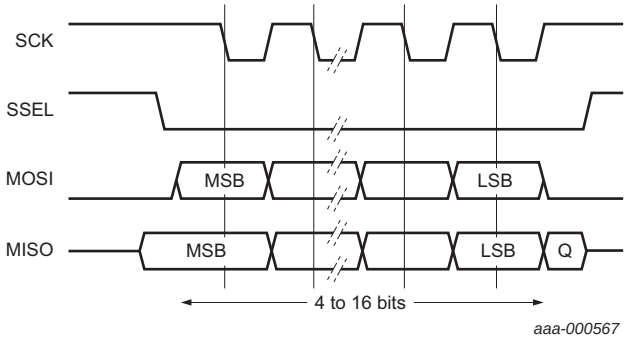
Data is then captured on the falling edges and propagated on the rising edges of the SCK signal.

If 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.

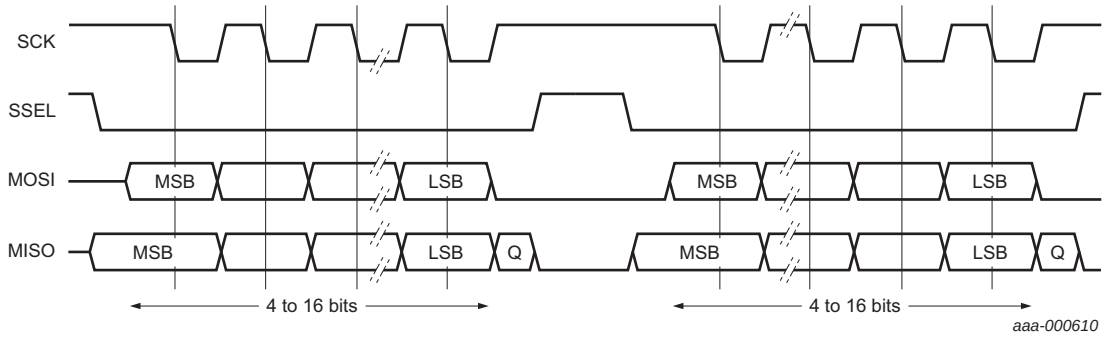
For continuous back-to-back transfers, pin SSEL is held LOW between successive data words and termination is the same as that of the single word transfer.

18.3.6 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 47.



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



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

Fig 47. SPI frame format with CPOL = 1 and CPHA = 0

In this configuration, during idle periods:

- 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 pin MOSI 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 goes LOW after one further half SCK period. This means that data is captured on the falling edges and is propagated on the rising edges of the SCK signal.

If 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 is 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 bit CPHA is logic 0. Therefore the master device must raise the level on pin SSEL of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, pin SSEL is returned to its idle state one SCK period after the last bit is captured.

18.3.7 SPI format with CPOL = 1, CPHA = 1

The transfer signal sequence for SPI format with CPOL = 1, CPHA = 1 is shown in Figure 48, which covers both single and continuous transfers.

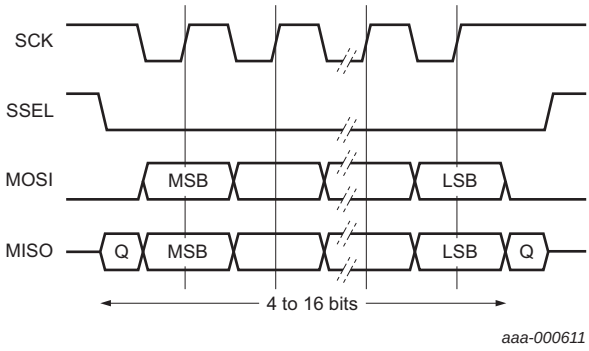


Fig 48. Motorola SPI frame format with CPOL = 1 and CPHA = 1 (single transfer)

In this configuration, during idle periods:

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

18.3.8 Semiconductor Microwire frame format

Figure 49 shows the Microwire frame format for a single frame. Figure 50 shows the same format when back-to-back frames are transmitted.

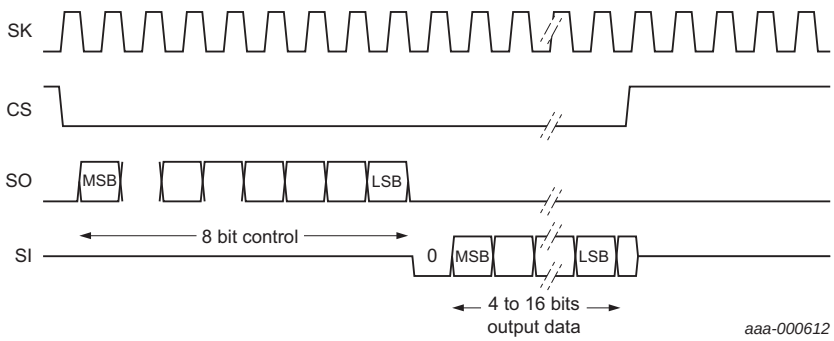


Fig 49. Microwire frame format (single transfer)

Microwire format is 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 is 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:

- 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 pin SO. CS remains LOW for the duration of the frame transmission. Pin SI remains tri-stated 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 line SI 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 is latched into the receive serial shifter, which causes the data to be transferred to the receive FIFO.

Remark: The off-chip slave device can tri-state the receive line either on the falling edge of SK after the LSB is latched by the receive shifter, or when pin CS 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 of SK after the LSB of the frame has been latched into the SSP.

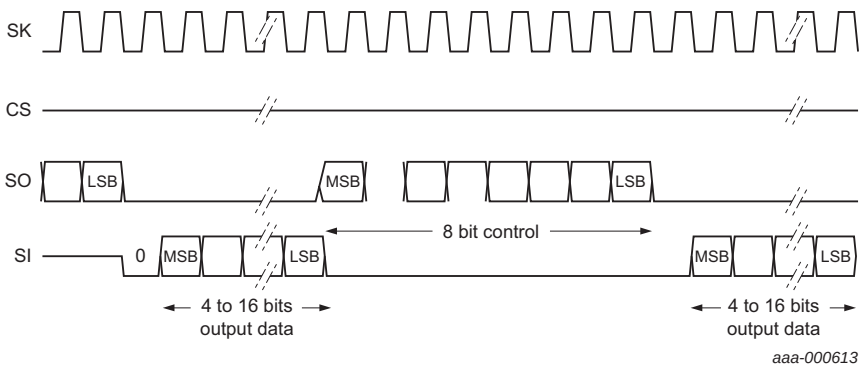


Fig 50. Microwire frame format (continuous transfers)

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

In Microwire mode, the SSP slave samples the first bit of received 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 51 illustrates the 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 twice 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.

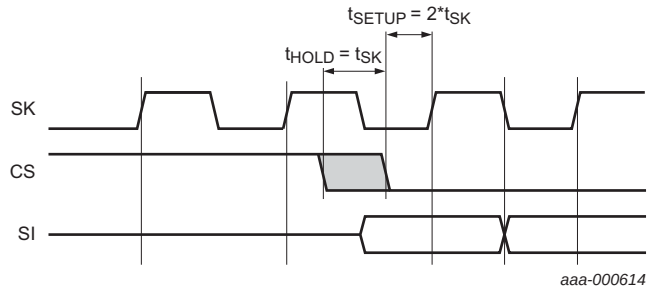


Fig 51. Microwire frame format (continuous transfers) - detailed

18.4 Register description

The SSP contains 9 registers as shown in Table 148. All registers are byte, halfword and word accessible.

Table 148. SSP register map

Name	Description	Access	Reset value[1]	Address
SSPCR0	control register 0. Selects serial clock rate, bus type, and data size.	R/W	0x0000	0xE006 8000
SSPCR1	control register 1. Selects Master/Slave and other modes.	R/W	0x00	0xE006 8004
SSPDR	data register. Writes fill the transmit FIFO, and reads empty the receive FIFO.	R/W	0x0000	0xE006 8008

Table 148. SSP register map *…continued*

Name	Description	Access	Reset value[1]	Address
SSPSR	status register	RO	0x03	0xE006 800C
SSPCPSR	clock prescale register	R/W	0x00	0xE006 8010
SSPIMSC	interrupt mask set and clear register	R/W	0x00	0xE006 8014
SSPRIS	raw interrupt status register	R/W	0x04	0xE006 8018
SSPMIS	masked interrupt status register	RO	0x00	0xE006 801C
SSPICR	SSPICR interrupt clear register	WO	NA	0xE006 8020

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

18.4.1 SSP Control register 0 (SSPCR0 - 0xE006 8000)

This register controls the basic operation of the SSP controller.

Table 149. SSP Control register 0 (SSPCR0 - address 0xE006 8000) bit description

Bit	Symbol	Value	Description	Reset value
3:0	DSS		data size select. Controls number of bits transferred in each frame. Values 0000-0010 are not supported and must 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	combination is not supported and must not be used	
6	CPOL		clock out polarity. Only used in SPI mode	
		0	SSP controller maintains bus clock LOW between frames	
		1	SSP controller maintains bus clock HIGH between frames	

Table 149. SSP Control register 0 (SSPCR0 - address 0xE006 8000) bit description *…continued*

Bit	Symbol	Value	Description	Reset value
7	CPHA		clock out phase. Only used in SPI mode.	0
		0	SSP controller captures serial data on the first clock transition of frame, that is, the transition **away from** the inter-frame state of the clock line.	
		1	SSP controller captures serial data on second clock transition of frame, that is, the transition **back to** the inter-frame state of the clock line.	
15:8	SCR		serial clock rate. Number of prescaler-output clocks per bit on bus, minus one. Given that CPSDVR is the prescale divider, and APB clock PCLK clocks the prescaler, the bit frequency is PCLK / (CPSDVSR × [SCR+1]).	0x00

18.4.2 SSP Control register 1 (SSPCR1 - 0xE006 8004)

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

Table 150: SSP Control register 1 (SSPCR1 - address 0xE006 8004) bit description

Bit	Symbol	Value	Description	Reset value
0	LBM		Loop back Mode	0
		0	during normal operation	
		1	serial input is taken from serial output (MOSI or MISO) rather than serial input pin (MISO or MOSI respectively)	
1	SSE		SSP enable	0
		0	SSP controller is disabled	
		1	SSP controller interacts with other devices on serial bus. Software must write appropriate control information to other SSP registers and interrupt controller registers before setting this bit.	
2	MS		Master/Slave mode. Can only be written when bit SSE is logic 0.	0
		0	SSP controller acts as a master on the bus, driving SCLK, MOSI, and SSEL lines and receiving MISO line\	
		1	SSP controller acts as a slave on the bus, driving MISO line and receiving SCLK, MOSI, and SSEL lines	
3	SOD		slave output disable. Relevant only in slave mode (MS = 1). If it is logic 1, this blocks this SSP controller from driving transmit data line MISO.	0
7:4	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

18.4.3 SSP Data register (SSPDR - 0xE006 8008)

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

Table 151: SSP Data register (SSPDR - address 0xE006 8008) bit description

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

18.4.4 SSP Status register (SSPSR - 0xE006 800C)

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

Table 152: SSP Status register (SSPDR - address 0xE006 800C) bit description

Bit	Symbol	Description	Reset value
0	TFE	transmit FIFO empty. Logic 1 if transmit FIFO is empty, logic 0 if not.	1
1	TNF	transmit FIFO not full. Logic 0 if Tx FIFO is full, logic 1 if not.	1
2	RNE	receive FIFO not empty. Logic 0 if receive FIFO is empty, logic 1 if not.	0
3	RFF	receive FIFO full. Logic 1 if receive FIFO is full, logic 0 if not.	0
4	BSY	busy. Logic 0 if SSP controller is idle, or logic 1 if currently sending/receiving a frame and/or Tx FIFO is not empty.	0
7:5	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

18.4.5 SSP Clock prescale register (SSPCPSR - 0xE006 8010)

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 153: SSP Clock prescale register (SSPCPSR - address 0xE006 8010) bit description

Bit	Symbol	Description	Reset value
7:0	CPSDVSR	even value between 2 and 254 by which PCLK is divided to yield the prescaler output clock. Bit 0 always reads 0.	0

Remark: the SSPCPSR value must be properly initialized or the SSP controller is not able to transmit data correctly. If SSP is operating in master mode, $CPSDVSR_{min} = 2$, while in slave mode, $CPSDVSR_{min} = 12$.

18.4.6 SSP Interrupt mask set/clear register (SSPIMSC - 0xE006 8014)

This register controls whether either 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” means “disabled”. ARM uses the word “masked” to mean “enabled”. To avoid confusion, the word “masked” is not used.

Table 154: SSP Interrupt mask set/clear register (SSPIMSC - address 0xE006 8014) bit description

Bit	Symbol	Description	Reset value
0	RORIM	**receive overrun interrupt**. Software must set this bit to enable interrupt when a receive overrun occurs, that is, when Rx FIFO is full and another frame is received. The ARM specification implies that preceding frame data is overwritten by new frame data when this occurs.	0
1	RTIM	**receive time-out interrupt**. Software must set this bit to enable interrupt when a receive time-out condition occurs. A receive time-out occurs when Rx FIFO is not empty, and no new data is received, nor has data been read from FIFO, for 32 bit times.	0
2	RXIM	**Rx interrupt**. Software must set this bit to enable interrupt when RX FIFO is at least half full.	0
3	TXIM	**Tx interrupt**. Software must set this bit to enable interrupt when Tx FIFO is at least half empty.	0
7:4	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

18.4.7 SSP Raw interrupt status register (SSPRIS - 0xE006 8018)

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

Table 155: SSP Raw interrupt status register (SSPRIS - address 0xE006 8018) bit description

Bit	Symbol	Description	Reset value
0	RORRIS	logic 1 if another frame was received while RX FIFO was full. ARM specification implies that preceding frame data is overwritten by new frame data when this occurs.	0
1	RTRIS	logic 1 if there is a receive time-out condition. Note that a receive time-out can be negated if further data is received.	0
2	RXRIS	logic 1 if Rx FIFO is at least half full	0
3	TXRIS	logic 1 if Tx FIFO is at least half empty	1
7:4	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

18.4.8 SSP Masked interrupt register (SSPMIS - 0xE006 801C)

This read-only register contains a logic 1 for each interrupt condition that is asserted and enabled in the SSPIMSC. When an SSP interrupt occurs, the interrupt service routine must read this register to determine the cause(s) of the interrupt.

Table 156: SSP Masked interrupt status register (SSPMIS - address 0xE006 801C) bit description

Bit	Symbol	Description	Reset value
0	RORMIS	logic 1 if another frame was received while Rx FIFO was full, and this interrupt is enabled	0
1	RTMIS	logic 1 when there is a receive time-out condition and this interrupt is enabled. Note that a receive time-out can be negated if further data is received.	0
2	RXMIS	logic 1 if Rx FIFO is at least half full, and this interrupt is enabled	0
3	TXMIS	logic 1 if Tx FIFO is at least half empty, and this interrupt is enabled	0
7:5	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

18.4.9 SSP Interrupt clear register (SSPICR - 0xE006 8020)

Software can write several logic 1s 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 157: SSP Interrupt clear register (SSPICR - address 0xE006 8020) bit description

Bit	Symbol	Description	Reset value
0	RORIC	writing a logic 1 to this bit clears the “frame was received when Rx FIFO was full” interrupt	n/a
1	RTIC	writing a logic 1 to this bit clears the receive time-out interrupt	n/a
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

19. Analog-to-Digital Converter (ADC)

19.1 Features

- 10-bit successive approximation analog-to-digital converter
- Power-down mode
- Measurement range 0 V to $V_{DD(IO)}$ (typically 3 V; not to exceed $V_{DD(ADC)}$ voltage level)
- 10-bit conversion time $\geq$ 2.44 μs
- Burst conversion mode for single or multiple inputs
- Optional conversion on transition on input pin or timer match signal
- Dedicated result register for every analog input to reduce interrupt overhead

19.2 Description

The APB clock provides basic clocking for the ADCs. A programmable divider is included in each converter to scale this clock to the 4.5 MHz (maximum) clock needed by the successive approximation process. A fully accurate conversion requires 11 of these clocks.

19.3 Pin description

Table 158 gives a brief summary of each of ADC-related pin.

Table 158. ADC pin description

Pin	Type	Description
AD7:0	input	**analog inputs.** The ADC cell can measure the voltage on any of these input signals. Note that these analog inputs are always connected to their pins, even if the pin function select register assigns them to port pins. A simple self-test of the ADC can be done by driving these pins as port outputs. If the ADC is used, signal levels on analog input pins must not be above the level of V_{3A} at any time. Otherwise, ADC readings are invalid. If the ADC is not used in an application, the pins associated with the A-to-D inputs can be used as 5 V tolerant digital IO pins. **Remark:** while the ADC pins are specified as 5 V tolerant (see Table 59 on page 58), analog multiplexing in ADC block is not. More than 3.3 V ($V_{DD(ADC)}$) must not be applied to any pin that is selected as an ADC input, or the ADC reading will be incorrect. If for example AD0 and AD1 are used as the ADC0 inputs and voltage on AD0 = 4.5 V while AD1 = 2.5 V, an excessive voltage on AD0 can cause an incorrect reading of AD1, although the AD1 input voltage is within the right range.
$V_{DD(ADC)}$, GND_{ADC}	power	**analog power and ground.** Must nominally be the same voltages as V_{DD} and GND but must be isolated to minimize noise and error. $V_{DD(ADC)}$ also provides the ADC voltage reference level (V_{Ref}). **Remark:** If ADC is not used, $V_{DD(ADC)}$ must still be tied to $V_{DD(IO)}$, and GND_{ADC} must be grounded. These pins must not be left floating.

19.4 Register description

The ADC registers are shown in Table 159.

Table 159. ADC registers

Generic name	Description	Access	Reset value[1]	AD0 Address and name
ADCR	A/D control register. ADCR must be written to select operating mode before A/D conversion can occur.	R/W	0x0000 0001	0xE003 4000 AD0CR
ADGDR	A/D global data register. Contains ADC's DONE bit and result of the most recent A/D conversion.	R/W	n/a	0xE003 4004 AD0GDR
ADSTAT	A/D status register. Contains DONE and OVERRUN flags for all A/D channels and A/D interrupt flag.	RO	0x0000 0000	0xE003 4030 AD0STAT
ADINTEN	A/D interrupt enable register. Contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to generation of an A/D interrupt.	R/W	0x0000 0100	0xE003 400C AD0INTEN
ADDR3	A/D channel 3 data register. Contains the result of the most recent conversion completed on channel 3.	RO	n/a	0xE003 401C AD0DR3
ADDR4	A/D channel 4 data register. Contains the result of the most recent conversion completed on channel 4.	RO	n/a	0xE003 4020 AD0DR4
ADDR5	A/D channel 5 data register. Contains the result of the most recent conversion completed on channel 5.	RO	n/a	0xE003 4024 AD0DR5
ADDR6	A/D channel 6 data register. Contains the result of the most recent conversion completed on channel 6.	RO	n/a	0xE003 4028 AD0DR6
ADDR7	A/D channel 7 data register. Contains the result of the most recent conversion completed on channel 7.	RO	n/a	0xE003 402C AD0DR7

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

19.4.1 A/D Control register (AD0CR - 0xE003 4000)

Table 160: A/D Control register (AD0CR - address 0xE003 4000) bit description

Bit	Symbol	Value	Description	Reset value
7:0	SEL	-	selects which of the AD7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects pin AD0, and bit 7 selects pin AD7. In software-controlled mode, only one of these bits must be logic 1. In hardware scan mode, any value containing 1 to 8 1s. All 0s is equivalent to 0x01.	0x01
15:8	CLKDIV	-	APB clock (PCLK) is divided by (this value plus one) to produce the A/D converter clock, which must be less than or equal to 4.5 MHz. Typically, software must program the smallest value in this field that yields a clock of 4.5 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	0
16	BURST	1	ADC repeats conversions at the rate selected by CLKS field, scanning (if necessary) through the pins selected by 1s in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1 bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion in progress when this bit is cleared is completed. **Remark:** START bits must be 000 when BURST = logic 1 or conversions will not start.	0
		0	conversions are software controlled and require 11 clocks	
19:17	CLKS		this field selects the number of clocks used for each conversion in Burst mode, and the number of accuracy bits of the result in the RESULT bits of ADDR, between 11 clocks (10 bits) and 4 clocks (3 bits)	000
		000	11 clocks/10 bits	
		001	10 clocks/9bits	
		010	9 clocks/8 bits	
		011	8 clocks/7 bits	
		100	7 clocks/6 bits	
		101	6 clocks/5 bits	
		110	5 clocks/4 bits	
		111	4 clocks/3 bits	
20	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
21	PDN	1	ADC is operational	0
		0	ADC is in power-down mode	
23:22	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Table 160: A/D Control register (AD0CR - address 0xE003 4000) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
26:24	START		if bit BURST is logic 0, these bits control whether and when an A/D conversion is started:	0
		000	no start (this value must be used when clearing PDN to logic 0)	
		001	start conversion now	
		010	start conversion when the edge selected by bit 27 occurs on pin PIO16/EINT0	
		011	start conversion when the edge selected by bit 27 occurs on PIO22	
		100	reserved	
		101	reserved	
		110	start conversion when the edge selected by bit 27 occurs on MAT1.0	
		111	start conversion when the edge selected by bit 27 occurs on MAT1.1	
27	EDGE		this bit is significant only when the START field contains 010 to 111. In these cases:	0
		1	start conversion on a falling edge on the selected CAP/MAT signal	
		0	start conversion on a rising edge on the selected CAP/MAT signal	
31:28	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

19.4.2 A/D Global data register (AD0GDR - 0xE003 4004)

Table 161: A/D Global data register (AD0GDR - address 0xE003 4004) bit description

Bit	Symbol	Description	Reset value
5:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
15:6	RESULT	if bit DONE is logic 1, this field contains a binary fraction representing voltage on pin Ain selected by SEL field, divided by voltage on $V_{DD(ADC)}$ pin (V/V_{REF}). Logic 0 in the field indicates that the voltage on pin Ain was less than, equal to, or close to that on GND_{ADC}, while 0x3FF indicates that voltage on pin Ain was close to, equal to, or greater than that on V_{REF}.	n/a
23:16	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
26:24	CHN	contains the channel from which the RESULT bits were converted (for example, 000 identifies channel 0, 001 channel 1...).	n/a
29:27	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
30	OVERUN	logic 1 in burst mode if the results of one or more conversions are lost and overwritten before the conversion that produced the result in the RESULT bits. Cleared by reading this register.	0
31	DONE	logic 1 when an A/D conversion completes. Cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion started.	0

19.4.3 A/D Status register (AD0STAT - 0xE003 4030)

The A/D Status register allows the status of all A/D channels to be checked simultaneously. The DONE and OVERRUN flags appearing in register ADDRn for each A/D channel are mirrored in ADSTAT. The interrupt flag (the logical OR of all DONE flags) is also found in ADSTAT.

Table 162: A/D Status register (AD0STAT - address 0xE003 4030) bit description

Bit	Symbol	Description	Reset value
0:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
3	DONE3	mirrors the DONE status flag from result register for A/D channel 3	0
4	DONE4	mirrors the DONE status flag from result register for A/D channel 4	0
5	DONE5	mirrors the DONE status flag from result register for A/D channel 5	0
6	DONE6	mirrors the DONE status flag from result register for A/D channel 6	0
7	DONE7	mirrors the DONE status flag from result register for A/D channel 7	0
8:10	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
11	OVERRUN3	mirrors the OVERRRUN status flag from result register for A/D channel 3	0
12	OVERRUN4	mirrors the OVERRRUN status flag from result register for A/D channel 4	0
13	OVERRUN5	mirrors the OVERRRUN status flag from result register for A/D channel 5	0
14	OVERRUN6	mirrors the OVERRRUN status flag from result register for A/D channel 6	0
15	OVERRUN7	mirrors the OVERRRUN status flag from result register for A/D channel 7	0
16	ADINT	A/D interrupt flag. It is logic 1 when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via register ADINTEN.	0
31:17	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

19.4.4 A/D Interrupt enable register (AD0INTEN - 0xE003 400C)

This register allows control over which A/D channels generate an interrupt when a conversion is complete. For example, it can be desirable to use some A/D channels to monitor sensors by continuously performing conversions on them. The most recent results are read by the application program whenever they are needed. In this case, an interrupt is not desirable at the end of each conversion for some A/D channels.

Table 163. A/D Interrupt enable register (AD0INTEN - address 0xE003 400C) bit description

Bit	Symbol	Value	Description	Reset value
0:2	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
3	ADINTEN3	0	completion of a conversion on ADC channel 3 will not generate an interrupt	0
		1	completion of a conversion on ADC channel 3 will generate an interrupt	
4	ADINTEN4	0	completion of a conversion on ADC channel 4 will not generate an interrupt	0
		1	completion of a conversion on ADC channel 4 will generate an interrupt	
5	ADINTEN5	0	completion of a conversion on ADC channel 5 will not generate an interrupt	0
		1	completion of a conversion on ADC channel 5 will generate an interrupt	
6	ADINTEN6	0	completion of a conversion on ADC channel 6 will not generate an interrupt	0
		1	completion of a conversion on ADC channel 6 will generate an interrupt	
7	ADINTEN1	0	completion of a conversion on ADC channel 7 will not generate an interrupt	0
		1	completion of a conversion on ADC channel 7 will generate an interrupt	

Table 163. A/D Interrupt enable register (AD0INTEN - address 0xE003 400C) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
8	ADGINTEN	0	only the individual ADC channels enabled by ADINTEN7:0 generates interrupts	1
		1	only the global DONE flag in ADDR is enabled to generate an interrupt	
31:9	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

19.4.5 A/D Data registers (AD0DR0 to AD0DR7 - 0xE003 4010 to 0xE003 402C)

The A/D Data register holds the result when an A/D conversion is complete, and also includes the flags that indicate when a conversion is completed and when a conversion overrun has occurred.

Table 164. A/D Data registers (ADDR0 to ADDR7 address - 0xE003 4010 to 0xE003 402C) bit description

Bit	Symbol	Description	Reset value
5:0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
15:6	RESULT	if bit DONE is logic 1, this field contains a binary fraction representing voltage on pin AD0, divided by voltage on pin V_{REF} (V/V_{REF}). A logic 0 indicates voltage on pin AD0 was less than, equal to, or close to that on GND_{ADC}, while 0x3FF indicates that voltage on AD0 was close to, equal to, or greater than that on V_{REF}.	n/a
29:16	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
30	OVERRUN	logic 1 in burst mode if the results of one or more conversions are lost and overwritten before the conversion that produced the result in the RESULT bits. Cleared by reading this register.	
31	DONE	logic 1 when an A/D conversion completes. Cleared when this register is read.	n/a

19.5 Operation

19.5.1 Hardware-triggered conversion

If bit BURST in ADCR is logic 0 and the START field contains 010-111, the ADC starts a conversion when a transition occurs on a selected pin or Timer Match signal. The choices include conversion on a specified edge of any of 4 Match signals, or conversion on a specified edge of either of 2 Capture/Match pins. The pin state from the selected pad or the selected Match signal, XORed with ADCR bit 27, is used in the edge detection logic.

19.5.2 Interrupts

An interrupt request is asserted to the Vectored Interrupt Controller (VIC) when bit DONE is logic 1. Software can use bit Interrupt Enable for the ADC in the VIC to control whether this assertion results in an interrupt. DONE is negated when ADDR is read.

19.5.3 Accuracy vs. digital receiver

In order to get accurate voltage readings on the monitored pin, the AD0 function must be selected in the corresponding Pin Select register (see "Pin Connect Block" in Section 12.4 “Register description” on page 62). For the pin hosting an ADC input, it is not possible to have a digital function selected and also get valid ADC readings. An inside circuit disconnects ADC hardware from the associated pin whenever a digital function is selected on that pin.

20. PWM MOSFET Gate driver switching module

20.1 Features

- Three user-selectable outputs, out of which only one signal to be used for driving the power circuitry
- Fully programmed through firmware APIs only. No direct register access to the application

20.2 Description

MPT612 provides a PWM module, which can be programmed only through the firmware APIs. This module enables the PWM signal to drive the gate driver circuitry with the defined PWM switching frequency.

The PWM module is based on the standard timer module.

In order to choose the correct switching frequency, the following inputs must be considered.

The highest resolution (minimum percentage of duty cycle change allowed) is dependent upon the PWM peripheral clock frequency and the chosen switching frequency.

The highest resolution is defined as ($1 / 2^n$), where n defines the bit resolution. If the highest resolution chosen was say 8, then the change in duty cycle resolution would be = $1 / 2^8 = 1 / 256 = 0.39$ % duty cycle. This resolution allows the system to correctly track the Maximum Power point of the PV.

The switching frequency can be calculated using Equation 6:

$$f_{SW(PWM)} = \frac{\text{PWM peripheral clock frequency } (f_{AHB(PWM)})}{2^{\text{n-bit resolution}}} \quad (6)$$

Where $f_{SW(PWM)}$ is PWM switching frequency, and $f_{AHB(PWM)}$ is PWM peripheral clock frequency.

Assuming that the PWM peripheral clock frequency is 70 MHz (maximum allowed) and needs an 8-bit resolution, then the switching frequency would be:

$$f_{SW(PWM)} = \frac{70000000}{256} = 273.457 \quad kHz \quad (7)$$

20.3 Module APIs

The API PWM duty cycle can be modified as either a percentage or as a number of counts. Refer to the supplied MPT612 SDK for information.

The maximum PWM duty cycle count is defined by Equation 8.

$$\text{maximum PWM duty cycle count} = \frac{\text{PWM peripheral clock frequency } (f_{AHB(PWM)})}{f_{SW(PWM)}} \quad (8)$$

The maximum PWM duty cycle count calculated in Equation 8 is defined as a 100 % duty cycle. For a PWM signal with 0 % duty cycle, the PWM duty cycle count is 0.

21. 32-Bit timers: Timer1

21.1 Features

- 32-bit timer counter with a programmable 32-bit prescaler
- Counter or timer operation
- Up to four (Timer1) 32-bit capture channels that can take a snapshot of the timer value at input signal transitions. A capture event can also optionally generate an interrupt.
- Four 32-bit match registers 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 (Timer1) 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
- Up to four match registers can be configured as PWM allowing to use up to three match outputs as single edge-controlled PWM outputs

21.2 Applications

- Interval timer for counting internal events
- Pulse-width demodulator via capture inputs
- Free-running timer
- Pulse-width modulator via match outputs

21.3 Description

The timer counter is designed to count cycles of the peripheral clock (PCLK) or an externally supplied clock, and it 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.

Two match registers can be used to provide a single edge-controlled PWM output on the MATn.2.0 pins. Because the MAT0.3 register is not pinned out on Timer0, it is recommended to use the MRn.3 registers to control the PWM cycle length. One other match register is required to control the PWM edge position. The remaining two match registers can be used to create PWM output with the PWM cycle rate determined by MRn.3.

21.4 Pin description

Table 165 gives a brief summary of each of the timer counter related pins.

Table 165. Timer counter pin description

Pin	Type	Description
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. all Capture signals and their selection pins are listed below: • CAP1.0: PIO10 • CAP1.1: PIO11 • CAP1.2: PIO17 • CAP1.3: PIO18 timer counter block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 21.5.3 on page 182.
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) and the PWM control register (PWMCON) control the functionality of this output. all Match signals and their selection pins are listed below: • MAT1.0: PIO12 • MAT1.1: PIO13 • MAT1.2: PIO19 • MAT1.3: PIO20

21.5 Register description

Each timer counter contains the registers shown in Table 166. More detailed descriptions follow.

Table 166. Timer counter1 register map

Generic name	Description	Access	Reset value[1]	Timer/ counter1 address and name
IR	interrupt register. IR can be written to clear interrupts. IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0	0xE000 8000 T1IR
TCR	timer control register. TCR is used to control timer counter functions. Timer counter can be disabled or reset by TCR.	R/W	0	0xE000 8004 T1TCR
TC	timer counter. 32-bit TC is incremented every PR+1 cycles of PCLK. TC is controlled by TCR.	R/W	0	0xE000 8008 T1TC
PR	prescale register. Prescale counter (below) is equal to this value. Next clock increments TC and clears PC.	R/W	0	0xE000 800C T1PR
PC	prescale counter. 32-bit PC is a counter incremented to the value stored in PR. When value in PR is reached, TC is incremented and PC is cleared. PC is observable and controllable through the bus interface.	R/W	0	0xE000 8010 T1PC
MCR	match control register. MCR is used to control if an interrupt is generated and if TC is reset when a match occurs.	R/W	0	0xE000 8014 T1MCR

Table 166. Timer counter1 register map *…continued*

Generic name	Description	Access	Reset value[1]	Timer/ counter1 address and name
MR0	match register 0. MR0 can be enabled through MCR to reset TC, stop both TC and PC, and/or generate an interrupt every time MR0 matches TC.	R/W	0	0xE000 8018 T1MR0
MR1	match register 1; see MR0 description	R/W	0	0xE000 801C T1MR1
MR2	match register 2; see MR0 description	R/W	0	0xE000 8020 T1MR2
MR3	match register 3; see MR0 description	R/W	0	0xE000 8024 T1MR3
CCR	capture control register. CCR controls which edges of the capture inputs are used to load capture registers and whether an interrupt is generated when a capture occurs.	R/W	0	0xE000 8028 T1CCR
CR0	capture register 0. CR0 is loaded with value of TC when there is an event on CAPn.0 (CAP1.0 respectively) input.	RO	0	0xE000 802C T1CR0
CR1	capture register 1; see CR0 description	RO	0	0xE000 8030 T1CR1
CR2	capture register 2; see CR0 description	RO	0	0xE000 8034 T1CR2
CR3	capture register 3; see CR0 description	RO	0	0xE000 8038 T1CR3
EMR	external match register. EMR controls match function and external match pins MAT1.3:0.	R/W	0	0xE000 803C T1EMR
CTCR	count control register. CTCR selects between Timer and Counter mode, and in Counter mode selects signal and edge(s) for counting.	R/W	0	0xE000 8070 T1CTCR
PWMCON	PWM control register. PWMCON enables PWM mode for external match pins MAT1.3:0.	R/W	0	0xE000 8074 PWM1CON

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

21.5.1 Interrupt register (IR, TIMER1: T1IR - 0xE000 8000)

The interrupt register consists of 4 bits for the match interrupts and 4 bits for the capture interrupts. If an interrupt is generated, the corresponding IR bit is HIGH. Otherwise, the bit is LOW. Writing a logic 1 to the corresponding IR bit resets the interrupt. Writing a logic 0 has no effect.

Table 167: Interrupt register (IR, 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	CR3 Interrupt	interrupt flag for capture channel 3 event	0

21.5.2 Timer control register (TCR, TIMER1: T1TCR - 0xE000 8004)

The timer control register (TCR) is used to control the operation of the timer counter.

Table 168: Timer control register (TCR, TIMER1: T1TCR - address 0xE000 8004) bit description

Bit	Symbol	Description	Reset value
0	Counter Enable	if logic 1, timer counter and prescale counter are enabled for counting. If logic 0, counters are disabled.	0
1	Counter Reset	if logic 1, timer counter and prescale counter are synchronously reset on next positive edge of PCLK. Counters remain reset until TCR[1] is returned to logic 0.	0
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

21.5.3 Count control register (CTCR, 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.

When Counter mode is chosen as a mode of operation, the CAP input (selected by 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 edge or no changes in the level of the selected CAP input. Only if the identified event corresponds to that selected by bits 1:0 in register CTCR, the timer counter register is 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 cannot exceed one half of the PCLK clock. Consequently, duration of the HIGH/LOW levels on the same CAP input in this case cannot be shorter than 1 / PCLK.

Table 169: Count control register (CTCR, TIMER1: T1TCR - address 0xE000 8070) bit description

Bit	Symbol	Value	Description	Reset value
1:0	Counter/ Timer Mode		selects which rising PCLK edges can increment the 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 CAP input selected by bits 3:2	
		10	counter mode: TC is incremented on falling edges on CAP input selected by bits 3:2	
		11	counter mode: TC is incremented on both edges on CAP input selected by bits 3:2	

Table 169: Count control register (CTCR, TIMER1: T1TCR - address 0xE000 8070) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
3:2	Count Input Select		if bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking:	00
		00	CAPn.0 (CAP1.0 for TIMER1)	
		01	CAPn.1 (CAP1.1 for TIMER1)	
		10	CAPn.2 (CAP1.2 for TIMER1)	
		11	CAP1.3 for TIMER1	
			Remark: If Counter mode is selected for a particular CAPn input in TnCTCR, the 3 bits for that input in 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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

21.5.4 Timer Counter (TC, 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 counts up through the value 0xFFFF FFFF and then wraps 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.

21.5.5 Prescale register (PR, TIMER1: T1PR - 0xE000 800C)

The 32-bit prescale register specifies the maximum value for the prescale counter.

21.5.6 Prescale counter register (PC, TIMER1: T1PC - 0xE000 8010)

The 32-bit prescale counter register controls division of PCLK by some constant value before it is applied to the timer counter. This allows control of the relationship between the resolution of the timer and the maximum time before the timer overflows. The prescale counter is incremented on every PCLK. When it reaches the value stored in the prescale register, the timer counter is incremented, and the prescale counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, and so on.

21.5.7 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 register MCR.

21.5.8 Match control register (MCR, 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 170.

Table 170: Match control register (MCR, 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 TC	0
		0	interrupt is disabled	
1	MR0R	1	reset on MR0: TC is reset if MR0 matches it	0
		0	feature disabled	
2	MR0S	1	stop on MR0: TC and PC are stopped and TCR[0] is set to 0 if MR0 matches TC	0
		0	feature disabled	
3	MR1I	1	interrupt on MR1: an interrupt is generated when MR1 matches the value in TC	0
		0	interrupt is disabled	
4	MR1R	1	reset on MR1: TC is reset if MR1 matches it	0
		0	feature disabled	
5	MR1S	1	stop on MR1: TC and PC are stopped and TCR[0] is set to 0 if MR1 matches TC	0
		0	feature disabled	
6	MR2I	1	interrupt on MR2: an interrupt is generated when MR2 matches the value in TC	0
		0	interrupt is disabled	
7	MR2R	1	reset on MR2: TC is reset if MR2 matches it	0
		0	feature disabled	
8	MR2S	1	stop on MR2: TC and PC are stopped and TCR[0] is set to 0 if MR2 matches TC	0
		0	feature disabled	
9	MR3I	1	interrupt on MR3: an interrupt is generated when MR3 matches the value in TC	0
		0	interrupt is disabled	
10	MR3R	1	reset on MR3: the TC is reset if MR3 matches it	0
		0	feature disabled	
11	MR3S	1	stop on MR3: TC and PC are stopped and TCR[0] is set to 0 if MR3 matches TC	0
		0	feature disabled	
15:12	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

21.5.9 Capture registers (CR0 - CR3)

Each capture register is associated with a device pin and can be loaded with the timer counter value when a specified event occurs on that pin. The settings in the capture control 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.

21.5.10 Capture control register (CCR, 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 171: Capture control register (CCR, 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 0s then 1s on CAPn.0 causes CR0 to be loaded with the contents of TC	0
		0	feature is disabled	
1	CAP0FE	1	capture on CAPn.0 falling edge: a sequence of 1s then 0s on CAPn.0 causes CR0 to be loaded with the contents of TC	0
		0	feature is disabled	
2	CAP0I	1	interrupt on CAPn.0 event: a CR0 load due to a CAPn.0 event generates an interrupt	0
		0	feature is disabled	
3	CAP1RE	1	capture on CAPn.1 rising edge: a sequence of 0s then 1s on CAPn.1 causes CR1 to be loaded with the contents of TC	0
		0	feature is disabled	
4	CAP1FE	1	capture on CAPn.1 falling edge: a sequence of 1s then 0s on CAPn.1 causes CR1 to be loaded with the contents of TC	0
		0	feature is disabled	
5	CAP1I	1	interrupt on CAPn.1 event: a CR1 load due to a CAPn.1 event generates an interrupt	0
		0	feature is disabled.	
6	CAP2RE	1	capture on CAPn.2 rising edge: A sequence of 0s then 1s on CAPn.2 causes CR2 to be loaded with the contents of TC	0
		0	feature is disabled	
7	CAP2FE	1	capture on CAPn.2 falling edge: a sequence of 1s then 0s on CAPn.2 causes CR2 to be loaded with the contents of TC	0
		0	feature is disabled	
8	CAP2I	1	interrupt on CAPn.2 event: a CR2 load due to a CAPn.2 event generates an interrupt	0
		0	feature is disabled	
9	CAP3RE	1	capture on CAPn.3 rising edge: A sequence of 0s then 1s on CAPn.3 causes CR3 to be loaded with the contents of TC	0
		0	feature is disabled	
10	CAP3FE	1	capture on CAPn.3 falling edge: a sequence of 1s then 0s on CAPn.3 causes CR3 to be loaded with the contents of TC	0
		0	feature is disabled	
11	CAP3I	1	interrupt on CAPn.3 event: a CR3 load due to a CAPn.3 event generates an interrupt	0
		0	feature is disabled	
15:12	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

21.5.11 External match register (EMR, TIMER1: T1EMR - 0xE000 803C)

The external match register provides both control and status of the external match pins MAT(0-3).

If the match outputs are configured as PWM output, the function of the external match registers is determined by the PWM rules (Section 21.5.13 on page 187).

Table 172: External match register (EMR, TIMER1: T1EMR - address 0xE000 803C) bit description

Bit	Symbol	Description	Reset value
0	EM0	external match 0. Reflects the state of output MAT1.0, whether this output is connected to its pin. If a match occurs between 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. Reflects the state of output MAT1.1, whether this output is connected to its pin. When a match occurs between 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. Reflects the state of output MAT1.2, whether this output is connected to its pin. When a match occurs between 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	EM3	external match 3. Reflects the state of output MAT1.3, whether this output is connected to its pin. When a match occurs between 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 EM0. Table 173 shows the encoding of these bits.	00
7:6	EMC1	external match control 1. Determines the functionality of EM1. Table 173 shows the encoding of these bits.	00
9:8	EMC2	external match control 2. Determines the functionality of EM2. Table 173 shows the encoding of these bits.	00
11:10	EMC3	external match control 3. Determines the functionality of EM3. Table 173 shows the encoding of these bits.	00
15:12	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Table 173. External match control

EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4]	Function
00	do nothing
01	clear 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

21.5.12 PWM Control register (PWMCON, TIMER1: PWM1CON - 0xE000 8074)

The PWM control register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the external match register (EMR).

For each timer, a maximum of three single edge-controlled PWM outputs can be selected on the MATn.2:0 outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

Table 174: PWM Control register (PWMCON, TIMER1: PWM1CON - address 0xE000 8074) bit description

Bit	Symbol	Description	Reset value
0	PWM enable	if logic 1, PWM mode is enabled for MATn.0. If logic 0, MATn.0 is controlled by EM0	0
1	PWM enable	if logic 1, PWM mode is enabled for MATn.1. If logic 0, MATn.1 is controlled by EM1	0
2	PWM enable	if logic 1, PWM mode is enabled for MATn.2. If logic 0, MATn.2 is controlled by EM2	0
3	PWM enable	if 1, PWM mode is enabled for MATn.3. If logic 0, MATn.3 is controlled by EM3.	0
4:32	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

21.5.13 Rules for single edge-controlled PWM outputs

- All single edge-controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
- Each PWM output goes HIGH when its match value is reached. If no match occurs (that is, the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
- If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal is cleared with the start of the next PWM cycle.
- If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output is reset to LOW on the next clock tick after the timer reaches the match value. Therefore, the PWM output always consists of a one clock tick wide positive pulse with a period determined by the PWM cycle length (that is, the timer reload value).
- If a match register is set to zero, then the PWM output goes HIGH the first time the timer returns to zero and stays HIGH continuously

Remark: If the match outputs are selected to perform as PWM outputs, the timer reset (MRnR) and timer stop (MRnS) bits in the match control register MCR must be set to zero except for the match register setting the PWM cycle length. For this register, set bit MRnR to logic 1 to enable the timer reset when the timer value matches the value of the corresponding match register.

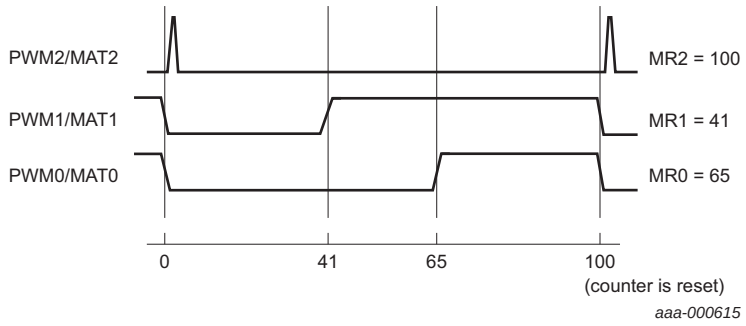


Fig 52. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by register PWCON

21.6 Example timer operation

Figure 53 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 54 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.

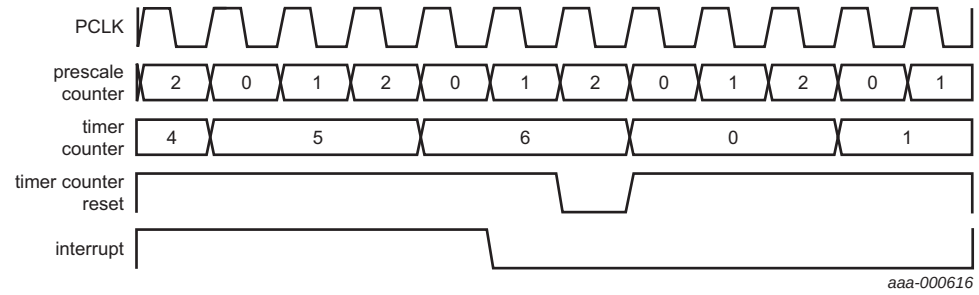


Fig 53. A timer cycle in which PR = 2, MRx = 6, and both interrupt and reset on match are enabled

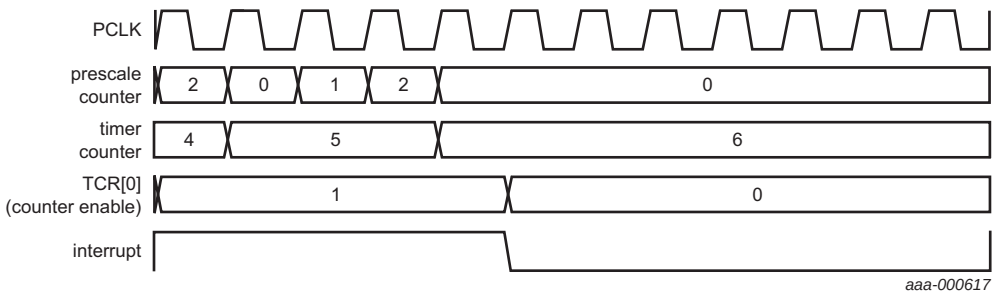


Fig 54. A timer cycle in which PR = 2, MRx = 6, and both interrupt and stop on match are enabled

21.7 Architecture

The block diagram for timer counter1 is shown in Figure 55.

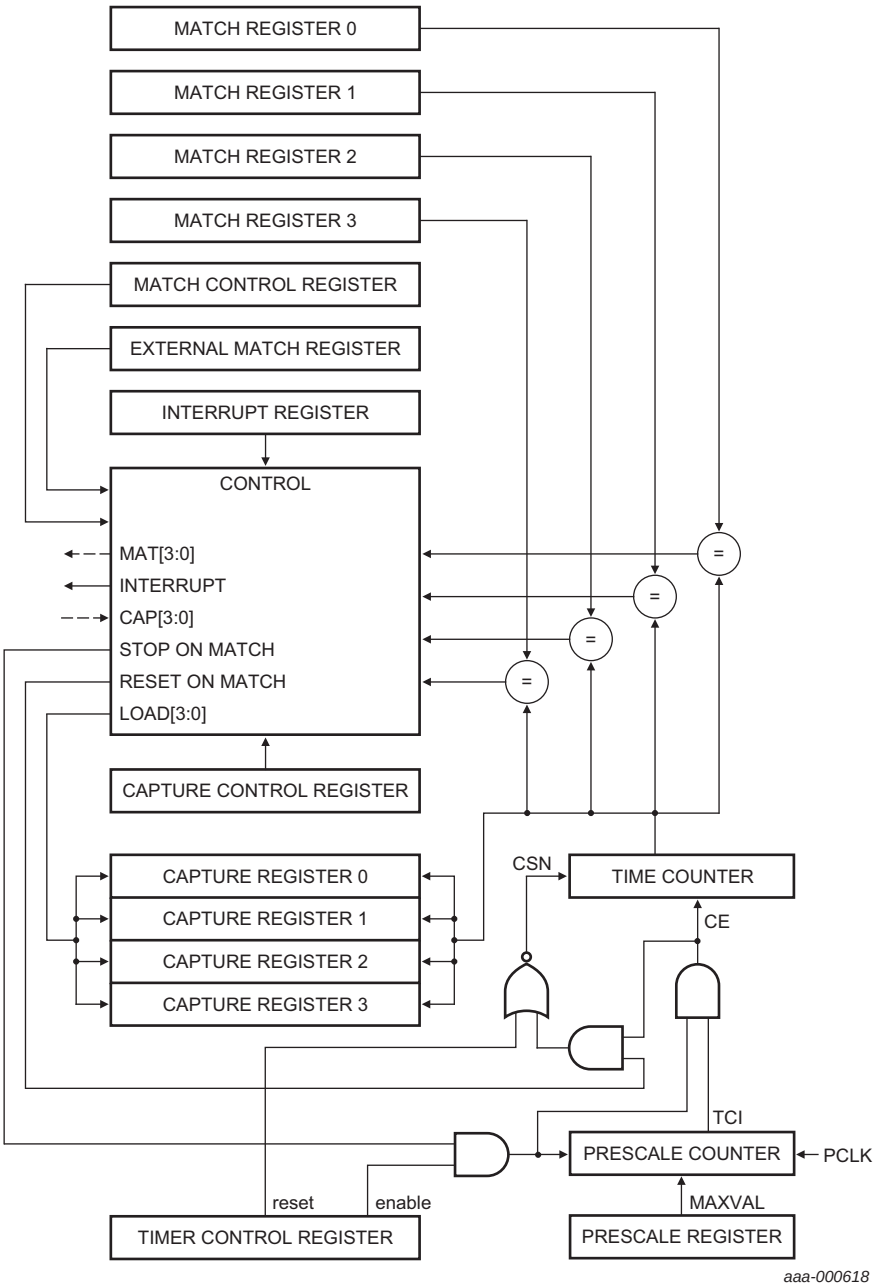


Fig 55. Timer1 block diagram

22. 16-Bit timers: Timer3

22.1 Features

- A 16-bit timer counter with a programmable 16-bit prescaler

- Counter or timer operation
- Four 16-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match
  - Stop timer on match with optional interrupt generation
  - Reset timer on match with optional interrupt generation
- Up to four (Timer3) external outputs corresponding to match registers with the following capabilities:
  - Set LOW on match
  - Set HIGH on match
  - Toggle on match
  - Do nothing on match
- For each timer, up to four match registers can be configured as PWM allowing up to three match outputs as single edge-controlled PWM outputs to be used

22.2 Applications

- Interval timer for counting internal events
- Free-running timer
- Pulse-width modulator via match outputs

22.3 Description

The timer counter is designed to count cycles of the peripheral clock (PCLK) or an externally supplied clock, and it can optionally generate interrupts or perform other actions at specified timer values, based on four match registers.

Due to the limited number of pins on the MPT612, none of the four match outputs of Timer3 are connected to device pins.

Two match registers can be used to provide a single edge-controlled PWM output on the MATn.2..0 pins. It is recommended to use the MRn.3 registers to control the PWM cycle length. One other match register is required to control the PWM edge position. The remaining two match registers can be used to create PWM output with the PWM cycle rate determined by MRn.3.

22.4 Pin description

Table 175 gives a brief summary of each of the timer counter related pins.

Table 175. Timer counter pin description

Pin	Type	Description
MAT3[3..0]	output	external match output 0/1. If a match register 0/1 (MR3:0) equals the Timer Counter (TC), this output can either toggle, go LOW, go HIGH, or do nothing. External match register (EMR) and PWM control register (PWMCON) control the functionality of this output. all Match signals and their selection pins are listed below: • MAT3.0: PIO21 • MAT3.1: PIO0 • MAT3.2: PIO1 • MAT3.3: PIO30

22.5 Register description

Each timer counter contains the registers shown in Table 176. More detailed descriptions follow.

Table 176. Timer counter3 register map

Generic name	Description	Access	Reset value[1]	Timer counter3 address and name
IR	interrupt register. IR can be written to clear interrupts. IR can be read to identify which of eight possible interrupt sources are pending.	R/W	0	0xE007 4000 T3IR
TCR	timer control register. TCR is used to control timer counter functions. Timer counter can be disabled or reset by TCR.	R/W	0	0xE007 4004 T3TCR
TC	timer counter. 16-bit TC is incremented every PR+1 cycles of PCLK. TC is controlled by TCR.	R/W	0	0xE007 4008 T3TC
PR	prescale register. Prescale counter (below) is equal to this value. Next clock increments TC and clears PC.	R/W	0	0xE007 400C T3PR
PC	prescale counter. 16-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, TC is incremented and PC is cleared. PC is observable and controllable via the bus interface.	R/W	0	0xE007 4010 T3PC
MCR	match control register. MCR is used to control if an interrupt is generated and if TC is reset when a match occurs.	R/W	0	0xE007 4014 T3MCR
MR0	match register 0. MR0 can be enabled through MCR to reset TC, stop both TC and PC, and/or generate an interrupt every time MR0 matches TC.	R/W	0	0xE007 4018 T3MR0
MR1	match register 1; see MR0 description	R/W	0	0xE007 401C T3MR1
MR2	match register 2; see MR0 description	R/W	0	0xE007 4020 T3MR2
MR3	match register 3; see MR0 description	R/W	0	0xE007 4024 T3MR3
EMR	external match register. EMR controls the match function and the external match pins MAT1.3:0.	R/W	0	0xE007 403C T3EMR
CTCR	count control register. CTCR selects between Timer and Counter mode, and in Counter mode selects signal and edge(s) for counting.	R/W	0	0xE007 4070 T3CTCR
PWMCON	PWM control register. PWMCON enables PWM mode for the external match pins MAT3.3:0.	R/W	0	0xE007 4074 PWM3CON

[1] Reset value reflects the data stored in used bits00 only. It does not include the content of reserved bits.

22.5.1 Interrupt register (IR TIMER3: T3IR - 0xE007 4000)

The interrupt register consists of 4 bits for the match interrupts and 4 bits for the capture interrupts. If an interrupt is generated, the corresponding bit in the IR is HIGH. Otherwise, the bit is LOW. Writing a logic 1 to the corresponding IR bit resets the interrupt. Writing a logic 0 has no effect.

Table 177: Interrupt register (IR, TIMER3: T3IR - address 0xE007 4000) 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
4:7	-	reserved	n/a

22.5.2 Timer control register (TCR, TIMER3: T3TCR - 0xE007 4004)

The timer control register (TCR) is used to control the operation of the timer counter.

Table 178: Timer control register (TCR, TIMER3: T3TCR - address 0xE007 4004) bit description

Bit	Symbol	Description	Reset value
0	Counter Enable	if logic 1, timer counter and prescale counter are enabled for counting. If logic 0, counters are disabled.	0
1	Counter Reset	if logic 1, timer counter and prescale counter are synchronously reset on next positive edge of PCLK. Counters remain reset until TCR[1] is returned to logic 0.	0
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

22.5.3 Count control register (CTCR, TIMER3: T3TCR - 0xE007 4070)

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.

Table 179: Count control register (CTCR, TIMER3: T3TCR - address 0xE007 4070) bit description

Bit	Symbol	Value	Description	Reset value
1:0	Counter/ Timer Mode		selects which rising PCLK edges can increment the timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC).	00
		00	timer mode: every rising PCLK edge	
7:2	-	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

22.5.4 Timer Counter (TC,TIMER3: T3TC - 0xE007 4008)

The 16-bit timer counter is incremented when the prescale counter reaches its terminal count. Unless it is reset before reaching its upper limit, the TC counts up through the value 0xFFFF FFFF and then wraps back to the value 0xE000 0000. This event does not cause an interrupt, but a match register can be used to detect an overflow if needed.

22.5.5 Prescale register (PR, TIMER3: T3PR - 0xE007 400C)

The 16-bit prescale register specifies the maximum value for the prescale counter.

22.5.6 Prescale counter register (PC, TIMER3: T3PC - 0xE007 4010)

The 16-bit prescale counter register controls division of PCLK by some constant value before it is applied to the timer counter. This allows control of the relationship between the resolution of the timer and the maximum time before the timer overflows. The prescale counter is incremented on every PCLK. When it reaches the value stored in the prescale register, the timer counter is incremented, and the prescale counter is reset on the next PCLK. This causes the TC to increment on every PCLK when PR = 0, every 2 PCLKs when PR = 1, and so on,

22.5.7 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.

22.5.8 Match control register (MCR, TIMER3: T3MCR - 0xE007 4014)

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 180.

Table 180: Match control register (MCR, TIMER3: T3MCR - address 0xE007 4014) bit description

Bit	Symbol	Value	Description	Reset value
0	MR0I	1	interrupt on MR0: an interrupt is generated when MR0 matches the value in TC	0
		0	interrupt is disabled	
1	MR0R	1	reset on MR0: TC is reset if MR0 matches it	0
		0	feature disabled	
2	MR0S	1	stop on MR0: TC and PC are stopped and TCR[0] is set to logic 0 if MR0 matches TC	0
		0	feature disabled	
3	MR1I	1	interrupt on MR1: an interrupt is generated if MR1 matches value in TC	0
		0	interrupt is disabled	
4	MR1R	1	reset on MR1: TC is reset if MR1 matches it	0
		0	feature disabled	
5	MR1S	1	stop on MR1: TC and PC are stopped and TCR[0] is set to logic 0 if MR1 matches TC	0
		0	feature disabled	
6	MR2I	1	interrupt on MR2: an interrupt is generated if MR2 matches value in TC	0
		0	interrupt is disabled	
7	MR2R	1	reset on MR2: TC is reset if MR2 matches it	0
		0	feature disabled	
8	MR2S	1	stop on MR2: TC and PC are stopped and TCR[0] is set to logic 0 if MR2 matches TC	0
		0	feature disabled	
9	MR3I	1	interrupt on MR3: an interrupt is generated if MR3 matches value in TC	0
		0	interrupt disabled	

Table 180: Match control register (MCR, TIMER3: T3MCR - address 0xE007 4014) bit description *...continued*

Bit	Symbol	Value	Description	Reset value
10	MR3R	1	reset on MR3: TC is reset if MR3 matches it	0
		0	feature disabled	
11	MR3S	1	stop on MR3: TC and PC are stopped and TCR[0] is set to logic 0 if MR3 matches TC	0
		0	feature disabled	
15:12	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

22.5.9 External match register (EMR, TIMER3: T3EMR - 0xE007 403C)

The external match register provides both control and status of the external match pins MAT(0-3).

If the match outputs are configured as PWM output, the function of the external match registers is determined by the PWM rules (Section 21.5.13 “Rules for single edge-controlled PWM outputs” on page 187).

Table 181: External match register (EMR, TIMER3: T3EMR - address 0xE007 4016) bit description

Bit	Symbol	Description	Reset value
0	EM0	external match 0. Reflects state of output MAT3.0, whether this output is connected to its pin. If a match occurs between TC and MR0, this timer output 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. Reflects the state of output MAT3.1, whether this output is connected to its pin. If a match occurs between TC and MR1, this timer output 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. Reflects the state of output MAT3.2, whether this output is connected to its pin. If a match occurs between TC and MR2, this timer output can either toggle, go LOW, go HIGH, or do nothing. Bits EMR[9:8] control the functionality of this output.	0
3	EM3	external match 3. Reflects the state of output MAT3.3, whether this output is connected to its pin. If a match occurs between TC and MR3, this timer output 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 EM0. Table 182 shows the encoding of these bits.	00
7:6	EMC1	external match control 1. Determines the functionality of EM1. Table 182 shows the encoding of these bits.	00
9:8	EMC2	external match control 2. Determines the functionality of EM2. Table 182 shows the encoding of these bits.	00
11:10	EMC3	external match control 3. Determines the functionality of EM3. Table 182 shows the encoding of these bits.	00
15:12	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Table 182. 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

22.5.10 PWM Control register (PWMCON, TIMER3: PWM3CON - 0xE007 4074)

The PWM control register is used to configure the match outputs as PWM outputs. Each match output can be independently set to perform either as PWM output or as match output whose function is controlled by the external match register (EMR).

For each timer, a maximum of three single edge-controlled PWM outputs can be selected on the MATn.2:0 outputs. One additional match register determines the PWM cycle length. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle length. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs are cleared.

Table 183: PWM Control register (PWMCON, TIMER3: PWM3CON - address 0xE007 4074) bit description

Bit	Symbol	Description	Reset value
0	PWM enable	if logic 1, PWM mode is enabled for MATn.0. If logic 0, MATn.0 is controlled by EM0.	0
1	PWM enable	if logic 1, PWM mode is enabled for MATn.1. If logic 0, MATn.1 is controlled by EM1.	0
2	PWM enable	if logic 1, PWM mode is enabled for MATn.2. If logic 0, MATn.2 is controlled by EM2.	0
3	PWM enable	if logic 1, PWM mode is enabled for MATn.3. If logic 0, MATn.3 is controlled by EM3. **Remark:** It is recommended to use MATn.3 to set PWM cycle.	0
4:32	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

22.5.11 Rules for single edge-controlled PWM outputs

- All single edge-controlled PWM outputs go LOW at the beginning of each PWM cycle (timer is set to zero) unless their match value is equal to zero.
- Each PWM output goes HIGH when its match value is reached. If no match occurs (that is, the match value is greater than the PWM cycle length), the PWM output remains continuously LOW.
- If a match value larger than the PWM cycle length is written to the match register, and the PWM signal is HIGH already, then the PWM signal is cleared on the next start of the next PWM cycle.

- If a match register contains the same value as the timer reset value (the PWM cycle length), then the PWM output is reset to LOW on the next clock tick. Therefore, the PWM output always consists of a one clock tick wide positive pulse with a period determined by the PWM cycle length (that is, the timer reload value).
- If a match register is set to zero, then the PWM output goes HIGH the first time the timer returns to zero and stays HIGH continuously.

Remark: If the match outputs are selected to perform as PWM outputs, the timer reset (MRnR) and timer stop (MRnS) bits in the match control register MCR must be set to logic 0 except for the match register setting the PWM cycle length. For this register, set bit MRnR to logic 1to enable the timer reset when the timer value matches the value of the corresponding match register.

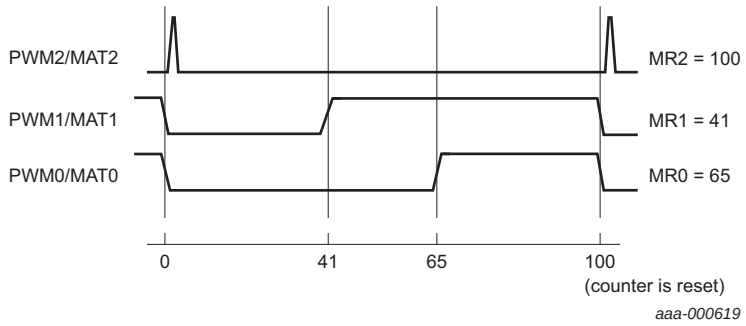


Fig 56. Sample PWM waveforms with a PWM cycle length of 100 (selected by MR3) and MAT3:0 enabled as PWM outputs by register PWCON

22.6 Example timer operation

Figure 53 on page 188 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 which 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 54 on page 188 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.

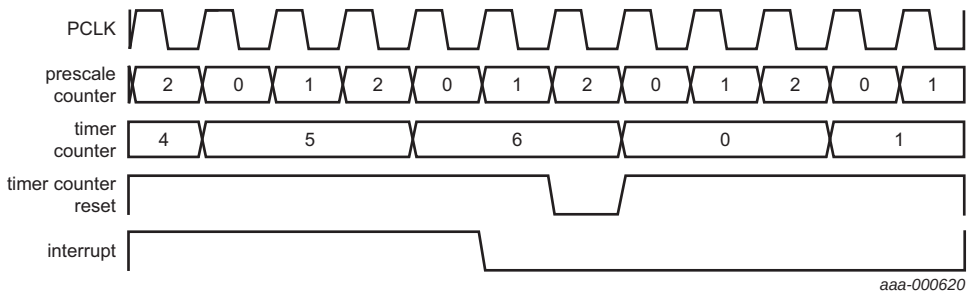


Fig 57. A timer cycle in which PR = 2, MRx = 6, and both interrupt and reset on match are enabled

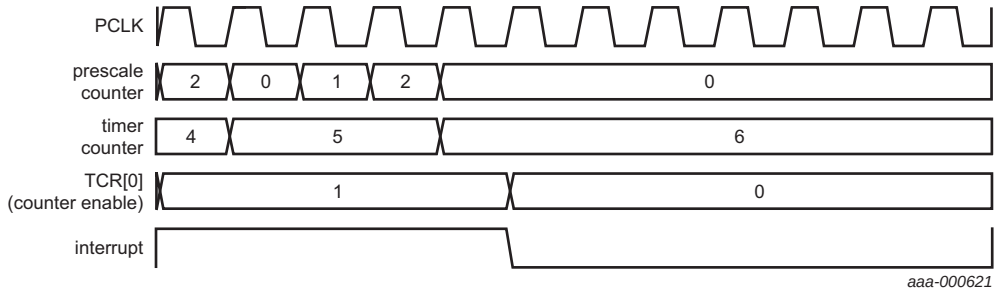
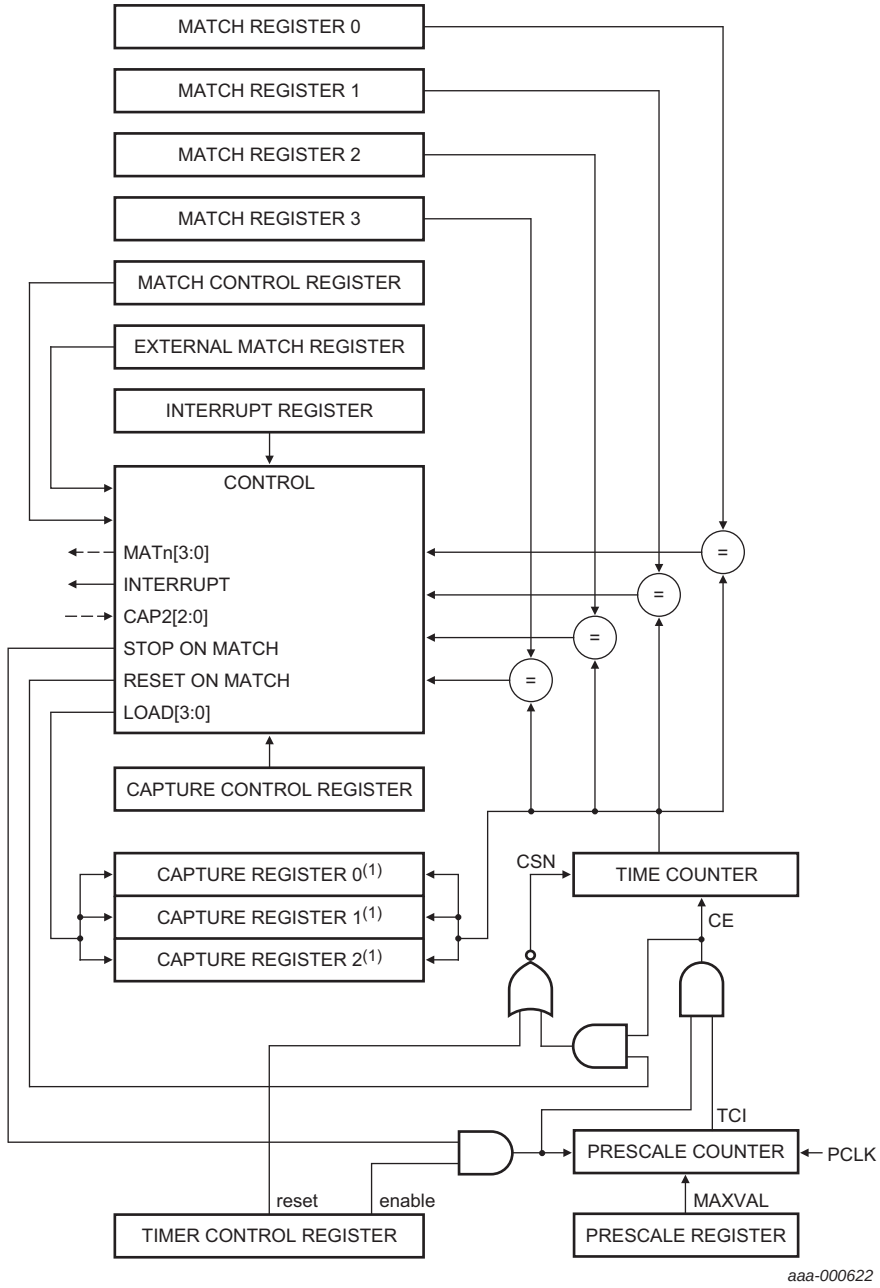


Fig 58. A timer cycle in which PR = 2, MRx = 6, and both interrupt and stop on match are enabled

22.7 Architecture

The block diagram for timer counter3 is shown in Figure 59.



(1) The capture registers are not available on TIMER3.

Fig 59. Timer3 block diagram

23. WatchDog Timer (WDT)

23.1 Features

- Internally resets chip if not periodically reloaded
- Supports Debug mode

- Watchdog timer is 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 indicates watchdog reset
- Includes programmable 32-bit timer with internal pre-scaler
- Time period can be selected from ($T_{PCLK} \times 256 \times 4$) to ($T_{PCLK} \times 2^{32} \times 4$) in multiples of $T_{PCLK} \times 4$.

23.2 Applications

The purpose of the watchdog is to reset the MPT612 within a reasonable amount of time if it enters an erroneous state. When enabled, the watchdog generates 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, see Section 10.10 “Reset” on page 52.

23.3 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 ($T_{PCLK} \times 256 \times 4$) and the maximum watchdog interval is ($T_{PCLK} \times 2^{32} \times 4$) in multiples of ($T_{PCLK} \times 4$). Use the watchdog as follows:

- Set the watchdog timer constant reload value in register WDTC
- Setup mode in register WDMOD
- Start the watchdog by writing 0xAA followed by 0x55 to register WDFEED
- Watchdog must be fed again before the watchdog counter underflows to prevent reset/interrupt

If the watchdog counter underflows, the program counter starts 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.

23.4 Register description

The watchdog contains 4 registers as shown in Table 184.

Table 184. Watchdog register map

Name	Description	Access	Reset value[1]	Address
WDMOD	watchdog mode register. Contains basic mode and status of watchdog timer.	R/W	0	0xE000 0000
WDTC	watchdog timer constant register. Determines time-out value.	R/W	0xFF	0xE000 0004
WDFEED	watchdog feed sequence register. Writing 0xAA followed by 0x55 to this register reloads watchdog timer to its preset value.	WO	NA	0xE000 0008
WDTV	watchdog timer value register. Reads out current value of watchdog timer.	RO	0xFF	0xE000 000C

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

23.4.1 Watchdog mode register (WDMOD - 0xE000 0000)

Register WDMOD controls the operation of the watchdog as per the combination of WDEN and RESET bits.

Table 185. Watchdog operating modes selection

WDEN	WDRESET	Mode of operation
0	X (0 or 1)	debug/operate without watchdog running
1	0	watchdog interrupt mode: debug with watchdog interrupt but no WDRESET enabled. when this mode is selected, a watchdog counter underflow sets WDINT flag and watchdog interrupt request is generated.
1	1	watchdog reset mode: operates with watchdog interrupt and WDRESET enabled. when this mode is selected, a watchdog counter underflow resets the MPT612. While the watchdog interrupt is also enabled in this case (WDEN = 1), it is not recognized since the watchdog reset clears the WDINT flag.

Once the WDEN and/or WDRESET bits are set, they cannot 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 is generated indefinitely.

Table 186: 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

Table 186: Watchdog mode register (WDMOD - address 0xE000 0000) bit description

Bit	Symbol	Description	Reset value
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 must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

23.4.2 WatchDog timer constant register (WDTC - 0xE000 0004)

Register WDTC determines the time-out value. Every time a feed sequence occurs, the WDTC content is reloaded in to the watchdog timer. It is a 32-bit register with 8 LSB set to logic 1 on reset. Writing values below 0xFF causes 0xFF to be loaded to the WDTC. Thus the minimum time-out interval is $T_{PCLK} \times 256 \times 4$.

Table 187: WatchDog timer constant register (WDTC - address 0xE000 0004) bit description

Bit	Symbol	Description	Reset value
31:0	Count	watchdog time-out interval	0x0000 00FF

23.4.3 Watchdog feed register (WDFEED - 0xE000 0008)

Writing 0xAA followed by 0x55 to this register reloads the watchdog timer to the WDTC value. This operation also starts the watchdog if it is enabled via register WDMOD. Setting bit WDEN in register WDMOD is not sufficient to enable the watchdog. A valid feed sequence must first be completed before the watchdog can generate an interrupt/reset. Until then, the watchdog ignores 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 interrupt/reset is 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. If an interrupt occurs during the feed sequence, an abort condition occurs.

Table 188: Watchdog feed register (WDFEED - address 0xE000 0008) bit description

Bit	Symbol	Description	Reset value
7:0	Feed	feed value must be 0xAA followed by 0x55	n/a

23.4.4 WatchDog timer value register (WDTV - 0xE000 000C)

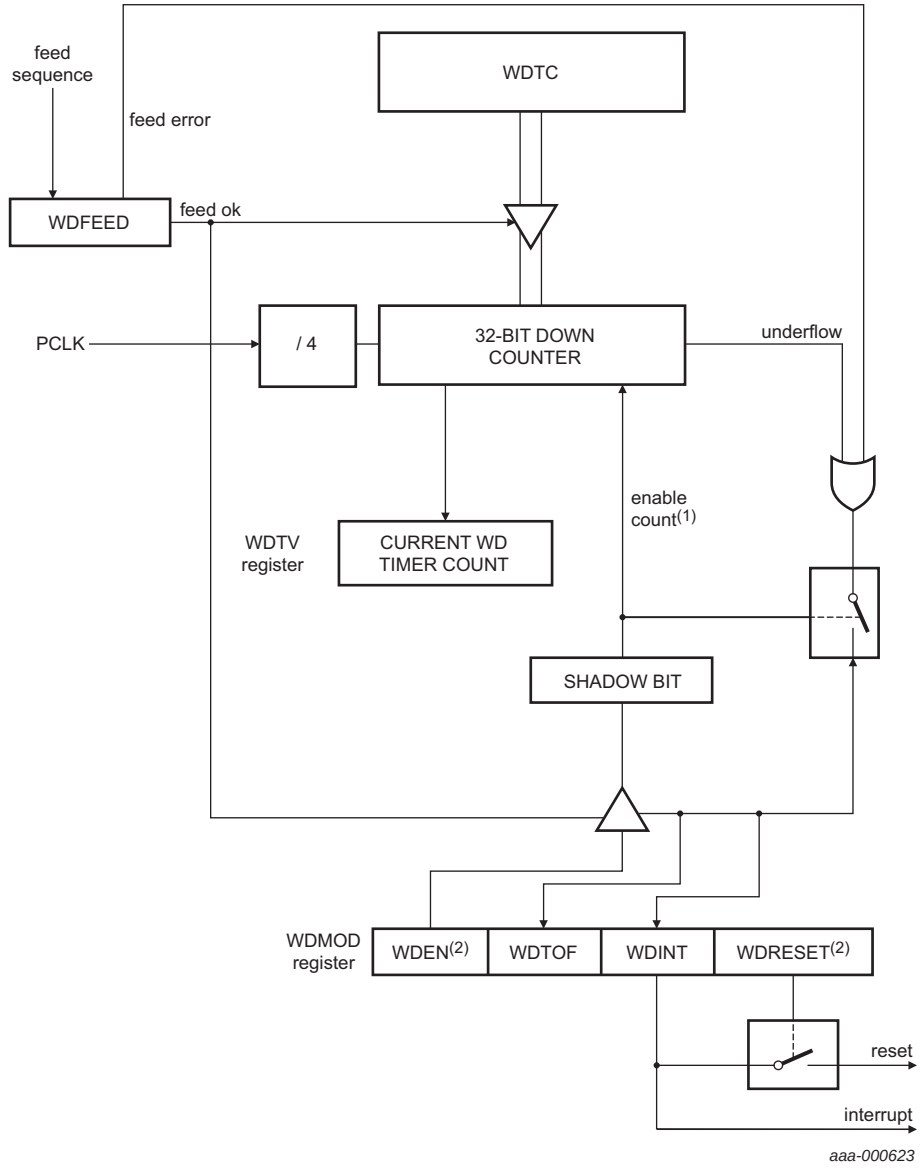
Register WDTV is used to read the current value of the watchdog timer.

Table 189: WatchDog timer value register (WDTV - address 0xE000 000C) bit description

Bit	Symbol	Description	Reset value
31:0	Count	counter timer value	0x0000 00FF

23.5 Block diagram

The block diagram of the watchdog timer is shown in Figure 60.



(1) Counter is enabled only when bit WDEN is set and a valid feed sequence is done.

(2) WDEN and WDRESET are sticky bits. Once set, they cannot be cleared until the watchdog timer underflows or an external reset occurs.

Fig 60. Watchdog timer block diagram

24. Real-Time Clock (RTC)

24.1 Introduction

References to Deep power-down mode and to control registers PWRCTRL and GPREG0 to GPREG3 only apply to revision A and above of the MPT612. Deep power-down mode is implemented in addition to Idle and Power-down modes; see Section 24.6.14 on page 210.

The power selector module is only implemented in revision A and above; see Section 24.7.1 on page 212.

24.2 Features

- Measures the passage of time to maintain a calendar and clock
- Ultra low-power design supports battery powered systems
- Provides seconds, minutes, hours, day of month, month, year, day of week, and day of year
- Can be used with dedicated 32 kHz oscillator or programmable prescaler from APB clock
- Dedicated power supply pin can be connected to a battery or to the main 3.3 V

24.3 Description

The Real-Time Clock (RTC) is a set of counters for measuring time when system power is on and optionally when it is off. It uses little power in Power-down mode. On the MPT612, the RTC can be clocked by a separate 32.768 kHz oscillator or by a programmable prescale divider based on the APB clock.

The RTC is powered by its own power supply pin, $V_{DD(RTC)}$, which must be connected to a battery or to the same 1.8 V core supply used by the rest of the device. Note that the PLL is disabled when waking up from power-down. See Section 10.8.8 “PLL and Power-down mode” on page 48 for the PLL start-up procedure.

24.4 Architecture

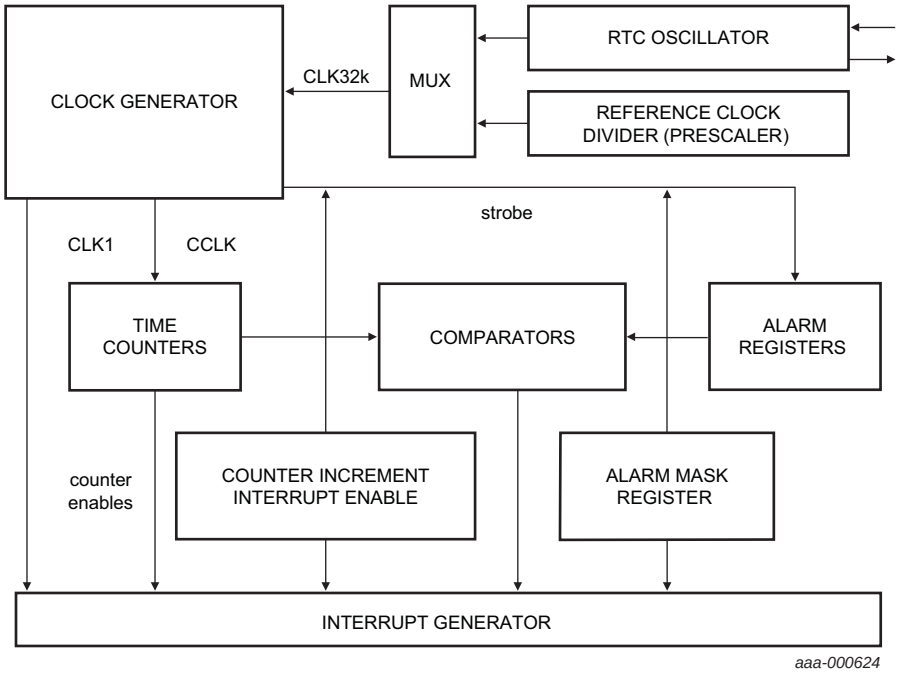


Fig 61. RTC block diagram

24.5 Pin description

Table 190. RTC pin description

Name	Type	Description
RTCX1	I	input to RTC oscillator circuit
RTCX2	O	output from RTC oscillator circuit **Remark:** If RTC is not used, RTCX1/2 pins can be left floating
$V_{DD(RTC)}$	I	**RTC power supply:** Voltage on this pin supplies the power to RTC. **Remark:** $V_{DD(RTC)}$ must always be connected to either pin V_{DDC} or an independent power supply (external battery). If V_{DDC} is present, RTC battery is disconnected and RTC is powered from V_{DDC} to conserve battery power; see Section 24.7.1.

24.6 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 24.6.2). The second set of eight locations are the Time Counter Group (Section 24.6.12 on page 209). The third set of eight locations contain the Alarm Register Group (Section 24.6.16 on page 211). The remaining registers control the Reference Clock Divider.

The Real-Time Clock includes the register shown in Table 191. Detailed descriptions of the registers follow. Most registers are not changed by a Reset, and the Reset value is noted as NC (Not Changed).

Table 191. Real-time clock (RTC) register map

Name	Size	Description	Access	Reset value[1]	Address
ILR	2	interrupt location register	R/W	NC	0xE002 4000
CTC	15	clock tick counter	RO	NC	0xE002 4004
CCR	4	clock control register	R/W	NC	0xE002 4008
CIIR	8	counter increment interrupt register	R/W	NC	0xE002 400C
AMR	8	alarm mask register	R/W	NC	0xE002 4010
CTIME0	32	consolidated time register 0	RO	NC	0xE002 4014
CTIME1	32	consolidated time register 1	RO	NC	0xE002 4018
CTIME2	32	consolidated time register 2	RO	NC	0xE002 401C
SEC	6	seconds counter	R/W	NC	0xE002 4020
MIN	6	minutes register	R/W	NC	0xE002 4024
HOUR	5	hours register	R/W	NC	0xE002 4028
DOM	5	day of month register	R/W	NC	0xE002 402C
DOW	3	day of week register	R/W	NC	0xE002 4030
DOY	9	day of year register	R/W	NC	0xE002 4034
MONTH	4	months register	R/W	NC	0xE002 4038
YEAR	12	years register	R/W	NC	0xE002 403C
PWRCTRL	3	deep power-down control register	R/W	111	0xE002 4040
GPREG0	32	general purpose 0 register	R/W	NC	0xE002 4044
GPREG1	32	general purpose 1 register	R/W	NC	0xE002 4048

Table 191. Real-time clock (RTC) register map *...continued*

Name	Size	Description	Access	Reset value[1]	Address
GPREG2	32	general purpose 2 register	R/W	NC	0xE002 404C
ALSEC	6	alarm value for seconds	R/W	NC	0xE002 4060
ALMIN	6	alarm value for minutes	R/W	NC	0xE002 4064
ALHOUR	5	alarm value for hours	R/W	NC	0xE002 4068
ALDOM	5	alarm value for day of month	R/W	NC	0xE002 406C
ALDOW	3	alarm value for day of week	R/W	NC	0xE002 4070
ALDOY	9	alarm value for day of year	R/W	NC	0xE002 4074
ALMON	4	alarm value for months	R/W	NC	0xE002 4078
ALYEAR	12	alarm value for year	R/W	NC	0xE002 407C
PREINT	13	prescaler value, integer portion	R/W	0	0xE002 4080
PREFRAC	15	prescaler value, fractional portion	R/W	0	0xE002 4084

[1] Registers in RTC other than register PWRCTRL and those that are part of the prescaler are not affected by chip reset. If RTC is enabled, these registers must be initialized by software. Reset value reflects the data stored in used bits only. It does not include the content of reserved bits.

24.6.1 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.

If the RTC is operating from its own oscillator on the RTCX1-2 pins, the RTC interrupt can bring the MPT612 out of Power-down or Deep power-down mode. When the RTC interrupt is enabled for wake-up and its selected event occurs, the oscillator wake-up cycle associated with the X1/2 pins is started. For details on the RTC-based wake-up process, see Section 10.5.3 “Interrupt wake-up register (INTWAKE - 0xE01F C144)” on page 39 and Section 10.12 “Wake-up timer” on page 56.

24.6.2 Miscellaneous register group

Table 192 summarizes the registers located from 0 to 7 of A[6:2]. More detailed descriptions follow.

Table 192. Miscellaneous registers

Name	Size	Description	Access	Address
ILR	2	interrupt location. Reading this location indicates source of an interrupt. Writing a logic 1 to the appropriate bit at this location clears 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 clock divider.	R/W	0xE002 4008
CIIR	8	counter increment interrupt. Selects which counters generate an interrupt when they are incremented.	R/W	0xE002 400C
AMR	8	alarm mask register. Controls which 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

24.6.3 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 193). Writing a logic 1 to the appropriate bit clears the corresponding interrupt. Writing a logic 0 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 193: Interrupt location register (ILR - address 0xE002 4000) bit description

Bit	Symbol	Description	Reset value
0	RTCCIF	if logic 1, counter increment interrupt block generated an interrupt. Writing a logic 1 to this bit location clears counter increment interrupt.	n/a
1	RTCALF	if logic 1, the alarm registers generated an interrupt. Writing a logic 1 to this bit location clears alarm interrupt.	n/a
7:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.6.4 Clock tick counter register (CTC - 0xE002 4004)

The clock tick counter is read only. It can be reset to zero by the clock control register (CCR). The CTC consists of the bits of the clock divider counter.

Table 194: Clock tick counter register (CTC - address 0xE002 4004) bit description

Bit	Symbol	Description	Reset value
0	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
15:1	Clock Tick Counter	before the seconds counter, CTC counts 32,768 clocks per second. Due to RTC prescaler, these 32,768 time increments cannot all be of the same duration. Refer to Section 24.8 on page 213 for details.	n/a

If the RTC is driven by the external 32.786 kHz oscillator, subsequent read operations of the CTC can yield an incorrect result. The CTC is implemented as a 15-bit ripple counter so that not all 15 bits change simultaneously. The LSB changes first, then the next, and so on. Since the 32.786 kHz oscillator is asynchronous to the CPU clock, it is possible for a CTC read to occur during the time when the CTCR bits are changing, resulting in an incorrect large difference between back-to-back reads.

If the RTC is driven by the PCLK, the CPU and the RTC are synchronous because both of their clocks are driven from the PLL output. Therefore, incorrect consecutive reads cannot occur.

24.6.5 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 195.

Table 195: Clock control register (CCR - address 0xE002 4008) bit description

Bit	Symbol	Description	Reset value
0	CLKEN	clock enable. If logic 1, time counters are enabled. When logic 0, they are disabled so that they can be initialized.	n/a
1	CTCRST	CTC reset. If logic 1, the elements in the Clock Tick Counter are reset. The elements remain reset until CCR[1] is changed to logic 0.	n/a
3:2	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
4	CLKSRC	if logic 0, the Clock Tick Counter takes its clock from the prescaler, as on earlier devices in the NXP Embedded ARM family. If logic 1, CTC takes its clock from the 32 kHz oscillator that is connected to the RTCX1 and RTCX2 pins (see Section 24.9 “RTC external 32 kHz oscillator component selection” on page 216 for hardware details).	n/a
7:5	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.6.6 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 logic 1 to bit 0 of the interrupt location register (ILR[0]).

Table 196: Counter increment interrupt register (CIIR - address 0xE002 400C) bit description

Bit	Symbol	Description	Reset value
0	IMSEC	if logic 1, an increment of the second value generates an interrupt	n/a
1	IMMIN	if logic 1, an increment of the minute value generates an interrupt	n/a
2	IMHOUR	if logic 1, an increment of the hour value generates an interrupt	n/a
3	IMDOM	if logic 1, an increment of the day of month value generates an interrupt	n/a
4	IMDOW	if logic 1, an increment of the day of week value generates an interrupt	n/a
5	IMDOY	if logic 1, an increment of the day of year value generates an interrupt	n/a
6	IMMON	if logic 1, an increment of the month value generates an interrupt	n/a
7	IMYEAR	if logic 1, an increment of the year value generates an interrupt	n/a

24.6.7 Alarm mask register (AMR - 0xE002 4010)

The alarm mask register (AMR) allows the user to mask any of the alarm registers. Table 197 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 logic 1 is written to the appropriate bit of the interrupt location register (ILR). If all mask bits are set, then the alarm is disabled.

Table 197: Alarm mask register (AMR - address 0xE002 4010) bit description

Bit	Symbol	Description	Reset value
0	AMRSEC	if logic 1, the second value is not compared for the alarm	n/a
1	AMRMIN	if logic 1, the minutes value is not compared for the alarm	n/a
2	AMRHOUR	if logic 1, the hour value is not compared for the alarm	n/a
3	AMRDOM	if logic 1, the day of month value is not compared for the alarm	n/a
4	AMRDOW	if logic 1, the day of week value is not compared for the alarm	n/a
5	AMRDOY	if logic 1, the day of year value is not compared for the alarm	n/a
6	AMRMON	if logic 1, the month value is not compared for the alarm	n/a
7	AMRYEAR	if logic 1, the year value is not compared for the alarm	n/a

24.6.8 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 198, Table 199, and Table 200. The least significant bit of each register is read back at bit 0, 8, 16, or 24.

The consolidated time registers are read only. The time counter addresses must be used to write new values to the time counters.

24.6.9 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 198: Consolidated time register 0 (CTIME0 - address 0xE002 4014) bit description

Bit	Symbol	Description	Reset value
5:0	Seconds	seconds value in the range 0 to 59	n/a
7:6	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
13:8	Minutes	minutes value in the range 0 to 59	n/a
15:14	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
20:16	Hours	hours value in the range 0 to 23	n/a

Table 198: Consolidated time register 0 (CTIME0 - address 0xE002 4014) bit description

Bit	Symbol	Description	Reset value
23:21	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
26:24	Day Of Week	day of week value in the range 0 to 6	n/a
31:27	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.6.10 Consolidated time register 1 (CTIME1 - 0xE002 4018)

The consolidated time register 1 contains the day of month, month, and year values.

Table 199: 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 1 to 28, 29, 30, or 31 (depending on the month and whether it is a leap year)	n/a
7:5	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
11:8	Month	month value in the range 1 to 12	n/a
15:12	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a
27:16	Year	year value in the range 0 to 4095	n/a
31:28	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.6.11 Consolidated time register 2 (CTIME2 - 0xE002 401C)

The consolidated time register 2 contains just the day of year value.

Table 200: 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 1 to 365 (366 for leap years)	n/a
31:12	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.6.12 Time counter group

The time value consists of the eight counters shown in Table 201 and Table 202. These counters can be read or written at the locations shown in Table 202.

Table 201. Time counter relationships and values

Counter	Size	Enabled by	Minimum value	Maximum value
Second	6	Clk1 (see Figure 61 on page 203)	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

Table 201. Time counter relationships and values *…continued*

Counter	Size	Enabled by	Minimum value	Maximum value
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 202. Time counter registers

Name	Size	Description	Access	Address
SEC	6	seconds value in the range 0 to 59	R/W	0xE002 4020
MIN	6	minutes value in the range 0 to 59	R/W	0xE002 4024
HOUR	5	hours value in the range 0 to 23	R/W	0xE002 4028
DOM	5	day of month value in the range 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 0 to 6[1]	R/W	0xE002 4030
DOY	9	day of year value in the range 1 to 365 (366 for leap years)[1]	R/W	0xE002 4034
MONTH	4	month value in the range 1 to 12	R/W	0xE002 4038
YEAR	12	year value in the range 0 to 4095	R/W	0xE002 403C

[1] These values are 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.

24.6.13 Leap year calculation

The RTC does a simple bit comparison to see if the two lowest order bits of the year counter are logic 0. 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.

24.6.14 Power control register group

The power control registers are summarized in Table 203. Register PWRCTRL is used to control the power supply to the main core, SRAM, and RTC oscillator. During Deep power-down mode, three general-purpose registers (GPREG[2:0]) remain fully powered up and can be used to store information while the rest of the chip is powered down. The registers are available in newer versions of the MPT612 only; see Section 24.1 “Introduction” on page 202.

Table 203. Power control registers

Name	Size	Description	Access	Address
PWRCTRL	3	Deep power-down control register	R/W	0xE002 4040
GPREG0	32	general-purpose storage register 0	R/W	0xE002 4044
GPREG1	32	general-purpose storage register 1	R/W	0xE002 4048
GPREG2	32	general-purpose storage register 2	R/W	0xE002 404C

24.6.15 Deep power-down control register (PWRCTRL - 0xE002 4040)

The deep power-down control register controls the power to the main core and either enables or disables the external 32 kHz oscillator and the SRAM block.

Table 204: Deep power-down control register (PWRCTRL - address 0xE002 4040) bit description

Bit	Symbol	Value	Description	Reset value
0	PMAIN		main power control	1
		1	power is applied to the entire chip	
		0	power to the main core is removed by an on-chip switch, and Deep power-down mode is entered	
1	PSRAM		SRAM power control	1
		1	SRAM remains powered up when Deep power-down mode is entered, and all SRAM data are retained. The power is either supplied by V_{DDC}, if available, or $V_{DD(RTC)}$; see Section 24.7.1 on page 212.	
		0	power is removed from SRAM when Deep power-down mode is entered	
2	POSC		32 kHz oscillator control	1
		1	32 kHz oscillator active	
		0	32 kHz oscillator disabled	
31:3	-		reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

Bits 0 to 2 in register PWRCTRL are set to logic 1 and full power is restored to the chip by any one of the following conditions:

- a reset pulse
- a low level on any of the three external interrupt pins
- a match in the RTC's alarm register
- a signal from the POR unit

Remark: A LOW level applied to any of the EINT[2:0] external interrupt pins always wakes up the part from Deep power-down mode regardless of whether they are enabled as external interrupts in the pin connect block (also see Section 10.5.3 "Interrupt wake-up register (INTWAKE - 0xE01F C144)" on page 39).

24.6.16 Alarm register group

The alarm registers are shown in Table 205. The values in these registers are compared with the time counters. If all the unmasked (See Section 24.6.7 on page 208) alarm registers match their corresponding time counters then an interrupt is generated. The interrupt is cleared when a logic 1 is written to bit one of the interrupt location register (ILR[1]).

Table 205. 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

24.7 RTC usage notes

If the RTC is used, $V_{DD(RTC)}$ must always be connected to either pin V_{DDC} or an independent power supply (external battery). If the clock source is lost, interrupted, or altered, no provision is made in the MPT612 to retain RTC status upon the $V_{DD(RTC)}$ power loss, or to maintain time incrementation.

Since the RTC operates using one of two available clocks (the APB clock (PCLK) or the 32 kHz signal coming from the RTCX1-2 pins), any interruption of the selected clock causes the time to drift away from the time value it would have provided otherwise. The variance could be due to actual clock time if the RTC was initialized to that, or simply an error in elapsed time since the RTC was activated.

While the signal from RTCX1-2 pins can be used to supply the RTC clock at anytime, selecting the PCLK as the RTC clock and entering Power-down mode causes a lapse in the time update. Also, feeding the RTC with the PCLK and altering this time base during system operation (by reconfiguring the PLL, the APB divider, or the RTC prescaler) results in some form of accumulated time error. Accumulated time errors can also occur when the RTC clock source is switched between the PCLK and the RTCX pins.

Once the 32 kHz signal from RTCX1-2 pins is selected as a clock source, the RTC can operate completely without the presence of the APB clock (PCLK). Therefore, power sensitive applications (that is, battery powered application) utilizing the RTC reduce the power consumption by using the signal from RTCX1-2 pins and writing a logic 0 to bit PCRTC in the PCONP power control register (see Section 10.9 “Power control” on page 49).

Remark: If the RTC is running from the 32 kHz signal and powered by $V_{DD(RTC)}$, the internal registers can be read. However, they cannot be written to unless bit PCRTC in register PCONP is set to logic 1; see Section 10.9.3 “Power control for peripherals register (PCONP - 0xE01F COC4)” on page 51.

24.7.1 Power selector

Newer revisions (see Section 24.1 “Introduction” on page 202) of the MPT612 include a power selector module which switches the power supply for the RTC and 32 kHz oscillator between $V_{DD(RTC)}$ and V_{DDC} to conserve battery power.

The power selector module monitors the power on the V_{DDC} power supply pin:

- If V_{DDC} is present, the RTC and the 32 kHz oscillator are powered by V_{DDC} during normal operation **and** during Deep power-down mode. If the SRAM block is selected to remain active during Deep power-down mode, it remains powered by V_{DDC}.
- If V_{DDC} is removed from the power supply pin, the power supply for the RTC and 32 kHz oscillator switches to the battery power supply on pin $V_{DD(RTC)}$. If the SRAM block is selected to remain active during Deep power-down mode, the SRAM power supply switches to $V_{DD(RTC)}$ as well - if Deep power-down mode and SRAM active have been selected in register PWRCTRL; see Section 24.6.15 on page 211.

Remark: Simply removing power from pin V_{DDC} does not cause the SRAM to be powered from the battery supply. In order to keep SRAM powered, Deep power-down mode must be entered:

1. Set bit 0 in register PWRCTRL to logic 0.
2. Set bit 1 in register PWRCTRL to logic 1 to allow the SRAM to remain powered.

24.8 Reference clock divider (prescaler)

The reference clock divider (as of now 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 run always 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 MPT612, 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 required 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 206. 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

24.8.1 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 207: Prescaler integer register (PREINT - address 0xE002 4080) bit description

Bit	Symbol	Description	Reset value
12:0	Prescaler Integer	contains integer portion of RTC prescaler value	0
15:13	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.8.2 Prescaler fraction register (PREFRAC - 0xE002 4084)

This is the fractional portion of the prescale value, and can be calculated as:

PREFRAC = PCLK – ((PREINT + 1) × 32768).

Table 208: Prescaler fraction register (PREFRAC - address 0xE002 4084) bit description

Bit	Symbol	Description	Reset value
14:0	Prescaler Fraction	contains fractional portion of RTC prescaler value	0
15	-	reserved, user software must not write logic 1s to reserved bits; value read from a reserved bit is not defined	n/a

24.8.3 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.

Similarly, any PCLK rate greater than 65.536 kHz (as long as it is an even number of cycles per second) can 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; see Section 24.6.4 “Clock tick counter register (CTC - 0xE002 4004)” on page 206.

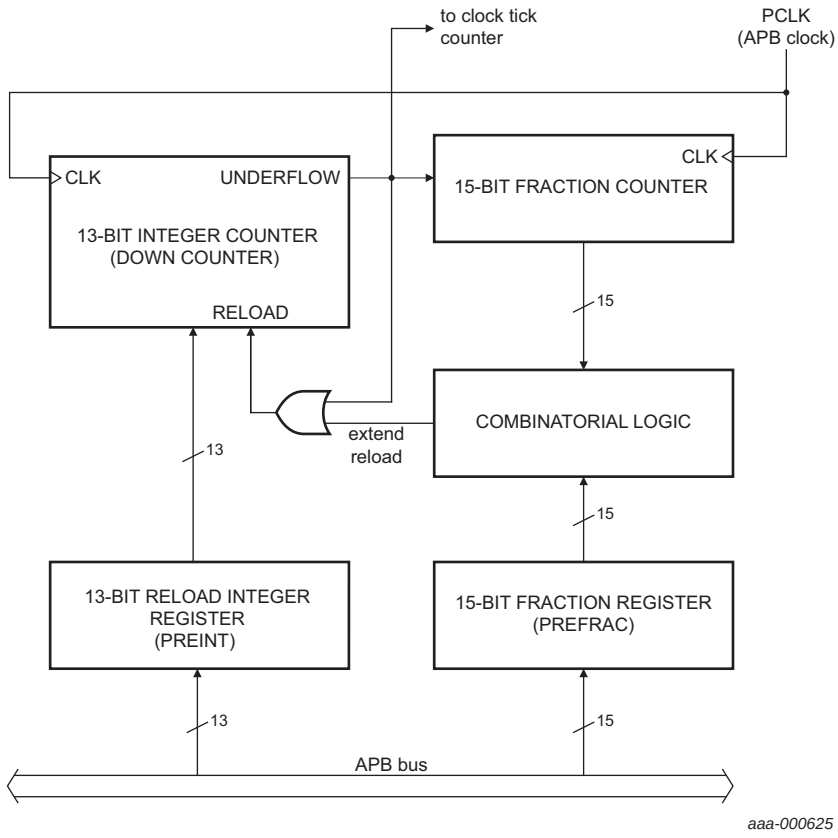


Fig 62. RTC prescaler block diagram

24.8.4 Prescaler operation

The Prescaler block labeled "Combination Logic" in Figure 62 determines when the decrement of the 13-bit PREINT counter extends 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 209.

For example, if PREFRAC bit 14 is logic 1 (representing the fraction 1 / 2), then half of the cycles counted by the 13-bit counter need to be longer. If there is a logic 1 in the LSB of the fraction counter, the logic causes every alternate count (whenever the LSB of the fraction counter = 1) to extend by one PCLK, evenly distributing the pulse widths. Similarly, a logic 1 in PREFRAC bit 13 (representing the fraction 1 / 4) causes every fourth cycle (whenever the two LSBs of the fraction counter = 10) counted by the 13-bit counter to be longer.

Table 209. 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	-	-	-	-	-	-	-	-	-	-	-	-

Table 209. Prescaler cases where the integer counter reload value is incremented *...continued*

Fraction counter	PREFRAC bit														
	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
--- ---- ---- 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

24.9 RTC external 32 kHz oscillator component selection

The RTC external oscillator circuit is shown in Figure 63. Since the feedback resistance is integrated on chip, only a crystal, the capacitances C_{X1} and C_{X2} need to be connected externally to the MPT612.

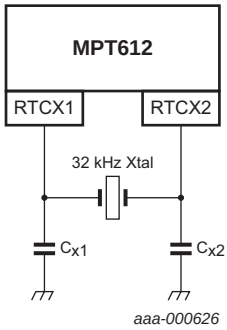


Fig 63. RTC 32 kHz crystal oscillator circuit

Table 210 gives the crystal parameters to be used. C_L is the typical load capacitance of the crystal and is specified by the crystal manufacturer. The actual C_L influences oscillation frequency. When using a crystal that is manufactured for a different load capacitance, the circuit will oscillate at a slightly different frequency (depending on the quality of the crystal) compared to the specified one. Therefore for an accurate time reference it is advised to use the load capacitors as specified in Table 210 that belong to a specific C_L. The value of external capacitances C_{X1} and C_{X2} specified in this table are calculated from the internal parasitic capacitances and the C_L. Parasitics from PCB and package are not taken into account.

For layout guidelines, see Section 10.4.2 “XTAL and RTC Printed-Circuit Board (PCB) layout guidelines” on page 37.

Table 210. Recommended values for the RTC external 32 kHz oscillator $C_{X1/X2}$ components

Crystal load capacitance C_L	Maximum crystal series resistance R_S	External load capacitors C_{X1}, C_{X2}
11 pF	< 100 kΩ	18 pF, 18 pF
13 pF	< 100 kΩ	22 pF, 22 pF
15 pF	< 100 kΩ	27 pF, 27 pF

25. Flash memory system and programming

25.1 Introduction

The MPT612 has three levels of Code Read Protection (CRP) implemented:

- CRP1: disables access to chip via the JTAG pins and allows partial flash updates (excluding flash sector 0) using a limited set of the ISP commands. This mode is useful when CRP is required and flash field updates are needed but all sectors cannot be erased
- CRP2: disables access to chip via the JTAG pins and only allows full flash erase and update using a reduced set of the ISP commands
- CRP3: running an application with this level, fully disables any access to chip via the JTAG pins and ISP. This mode effectively disables ISP override using pin PIO14. The user's application must provide a flash update mechanism (if needed) using IAP calls, or call the re-invoke ISP command to enable flash update via pin UART0.

25.2 Boot loader

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

25.3 Features

- In-System Programming: In-System Programming (ISP) is the programming or reprogramming of the on-chip flash memory using the bootloader 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 performs erase and write operations on the on-chip flash memory, as directed by the end-user application code.

25.4 Applications

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

25.5 Description

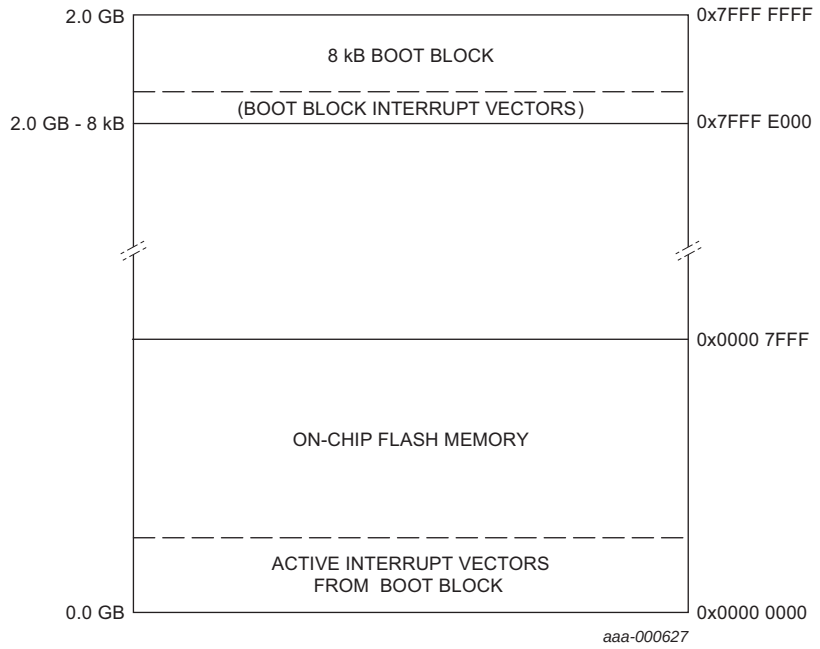
The bootloader code is executed every time the part is powered on or reset. The loader can execute the ISP command handler or the user application code. A LOW level after reset at pin PIO14 is considered as an external hardware request to start the ISP

command handler. Assuming that a correct signal is present on pin XTAL1 when the rising edge on pin RESET is generated, it can take up to 3 ms before PIO14 is sampled and the decision on whether to continue with user code or ISP handler is made. If PIO14 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 (PIO14 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 PIO14 that is used as a hardware request for ISP requires special attention. Since PIO14 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 can occur.

25.5.1 Memory map after any reset

The boot block is 8 kB in size and resides in the top of the on-chip memory space (starting from 0x7FFF E000). Both the ISP and IAP software use parts of the on-chip RAM. The RAM usage is described later. 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 address 0x0000 0000. The reset vector contains a jump instruction to the entry point of the flash bootloader software.



Remark: Memory regions are not drawn to scale.

Fig 64. Map of lower memory after reset for MPT612 with 32 kB of flash memory

25.5.2 Criterion for valid user code

Criterion for valid user code: The reserved ARM interrupt vector location (0x0000 0014) must 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 bootloader 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, the execution control is transferred to the user code by loading the program counter with 0x0000 0000. Hence the user flash reset vector must 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 must send a '?' (0x3F) as a synchronization character and wait for a response. The host side serial port settings must 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, the host must send the same string ("Synchronized<CR><LF>"). The auto-baud routine looks at the received characters to verify synchronization. If synchronization is verified, then "OK<CR><LF>" string is sent to the host. The host must respond by sending the value of 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 must be "10000<CR><LF>". "OK<CR><LF>" string is sent to the host after receiving the crystal frequency value. If synchronization is not verified, the auto-baud routine waits again for a synchronization character. For auto-baud to work correctly, the crystal frequency must 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 25.9 “ISP commands” on page 225.

25.5.3 Communication protocol

All ISP commands must be sent as single ASCII strings. Strings must 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.

25.5.4 ISP command format

"Command Parameter_0 Parameter_1 ... Parameter_n<CR><LF>" "Data" (data only for write commands).

25.5.5 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).

25.5.6 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 a 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 must send the check-sum after transmitting 20 UU-encoded lines. The length of any UU-encoded line must not exceed 61 characters (bytes) that is, it can hold 45 data bytes. The receiver must compare it with the check-sum of the received bytes. If the check-sum matches, then the receiver must respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match, the receiver must respond with "RESEND<CR><LF>". In response, the sender must retransmit the bytes.

25.5.7 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 must also support the same flow control scheme.

25.5.8 ISP command abort

Commands can be aborted by sending the ASCII control character "ESC". This feature is not documented as a command in the "ISP Commands" section. Once the escape code is received, the ISP command handler waits for a new command.

25.5.9 Interrupts during ISP

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

25.5.10 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 must 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.

25.5.11 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 can be lost upon reset. Flash programming commands use the top 32 bytes of on-chip RAM. The stack is at RAM top − 32. The maximum stack usage is 256 bytes and it grows downwards.

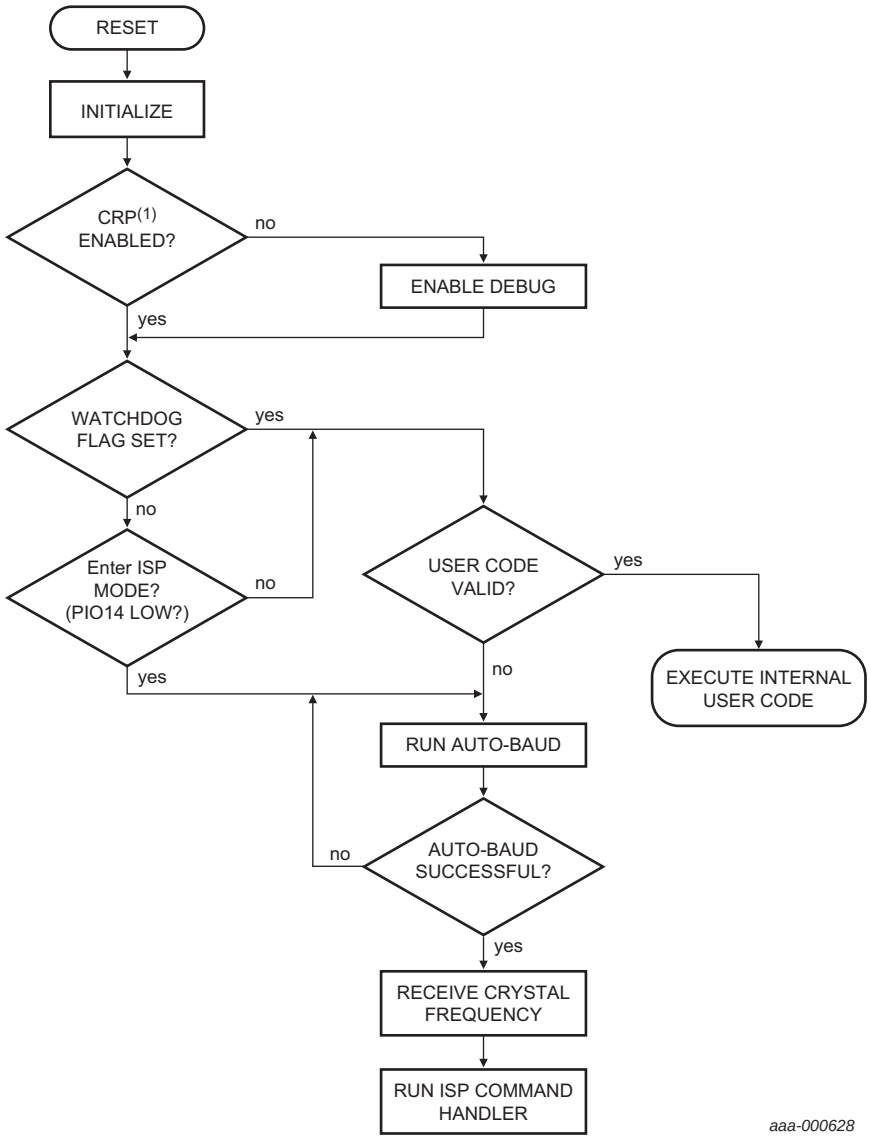
25.5.12 RAM used by IAP command handler

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.

25.5.13 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 bootloader does not initialize the stack for RealMonitor.

25.5.14 Boot process flowchart



(1) Code Read Protection.

Fig 65. Boot process flowchart (CRP2 level only)

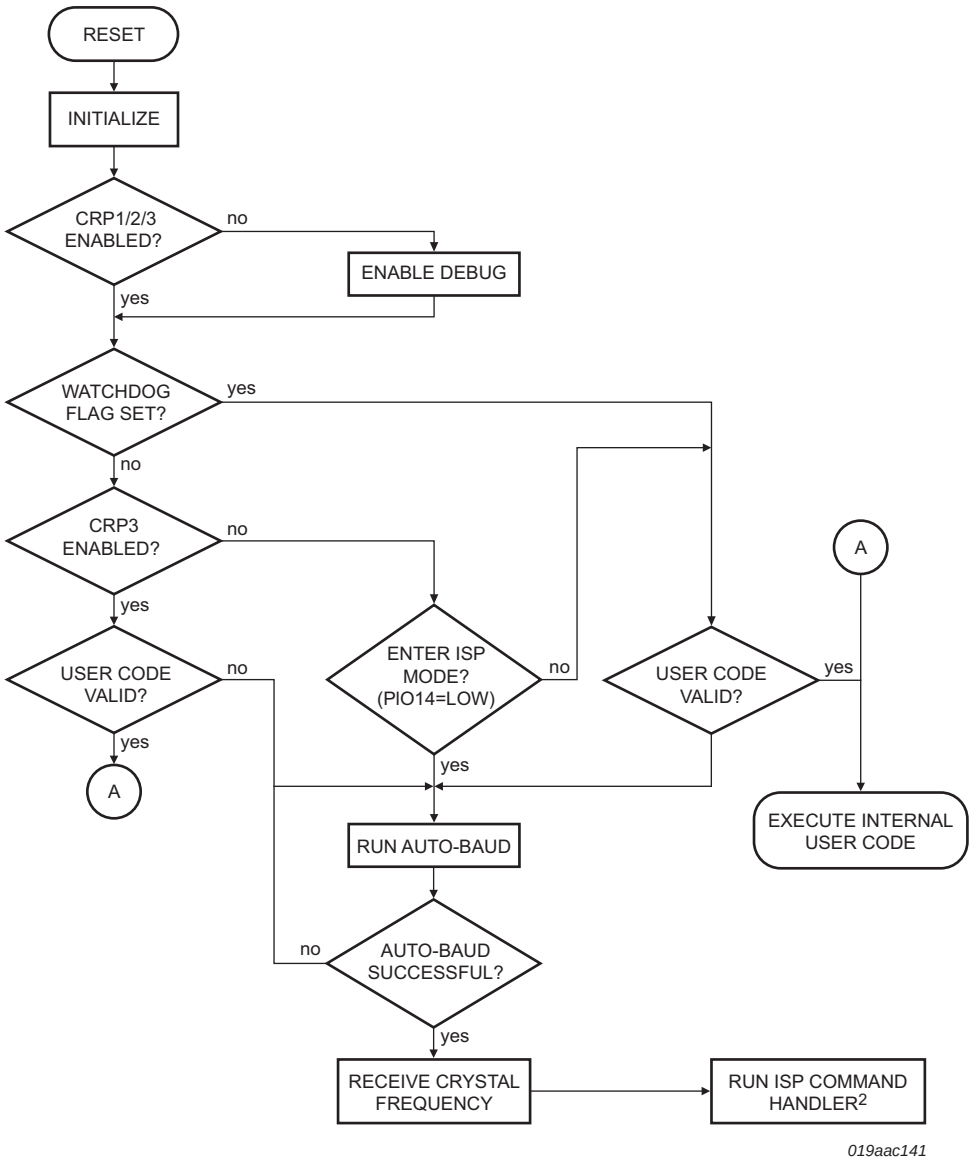


Fig 66. Boot process flow chart (CRP1/2/3 levels implemented - see Section 25.1 on page 217)

25.6 Sector numbers

Some IAP and ISP commands operate on "sectors" and specify sector numbers. Table 211 indicates the correspondence between sector numbers and memory addresses for MPT612 devices containing 32 kB of flash. IAP, ISP, and RealMonitor routines are located in the boot block. The boot block is present at addresses 0x7FFF E000 to 0x7FFF FFFF in all devices. ISP and IAP commands do not allow write/erase/go operation on the boot block. The entire 32 kB of flash memory on the MPT612 is available for the user's application and nxLibMpt firmware.

Table 211. Flash sectors in MPT612

Sector number	Sector size [kB]	Address range	MPT612 32 kB
0	4	0x0000 0000 to 0x0000 0FFF	+
1	4	0x0000 1000 to 0x0000 1FFF	+
2	4	0x0000 2000 to 0x0000 2FFF	+
3	4	0x0000 3000 to 0x0000 3FFF	+
4	4	0x0000 4000 to 0x0000 4FFF	+
5	4	0x0000 5000 to 0x0000 5FFF	+
6	4	0x0000 6000 to 0x0000 6FFF	+
7	4	0x0000 7000 to 0x0000 7FFF	+

25.7 Flash content protection mechanism

The MPT612 is equipped with the Error Correction Code (ECC) capable flash memory. The purpose of an error correction module is twofold. First, it decodes data words read from the memory into output data words. Second, 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 the 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, and flash bytes from 0x0000 0010 to 0x0000 001F are protected by the second ECC byte, and so on,

Whenever the CPU requests a read from the user's flash, both 128 bits of raw data containing the specified memory location and the matching ECC byte are evaluated. If the ECC mechanism detects a single error in the fetched data, a correction is applied before data are provided to the CPU. If a write request into the user's flash is made, a write of user-specified content is accompanied by a matching ECC value calculated and stored in the ECC memory.

When a sector of flash memory is erased, the corresponding ECC bytes are also erased. Once an ECC byte is written, it cannot 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.

25.8 Code Read Protection (CRP)

Code Read Protection is a mechanism that allows the user to enable different levels of security in the system so that access to the on-chip flash and use of the ISP can be restricted. When needed, CRP is invoked by programming a specific pattern in flash location at 0x0000 01FC. IAP commands are not affected by the code read protection.

Remark: Any CRP change becomes effective only after the device has gone through a reset.

Table 212. 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 cannot access RAM below 0x4000 0200 • copy RAM to flash command cannot write to sector 0 • erase command can erase sector 0 only when all sectors are selected for erase • compare command is disabled This mode is useful when CRP is required and flash field updates are needed but all sectors cannot be erased. Since compare command is disabled, in case of partial updates the secondary loader must implement the 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. If a valid user code is present in flash, sector 0ISP entry by pulling PIO14 LOW is disabled. This mode effectively disables ISP override using pin PIO14. It is up to the user's application to provide a flash update mechanism using IAP calls or to call the reinvoke ISP command if necessary. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Table 213. Code Read Protection hardware/software interaction

CRP option	User code valid	PIO14 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	n/a
No	yes	LOW	yes	yes	yes
CRP1	yes	HIGH	no	no	n/a
CRP1	yes	LOW	no	yes	yes
CRP2	yes	HIGH	no	no	n/a
CRP2	yes	LOW	no	yes	no
CRP3	yes	X	no	no	n/a
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 is terminated with return code CODE_READ_PROTECTION_ENABLED.

25.9 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 the ISP command handler only when the received ISP command has been executed and the new ISP command can be given by the host. Exceptions from this rule are commands: Set Baud Rate, Write to RAM, Read Memory, and Go commands.

Table 214. ISP command summary

ISP Command	Usage	Described in
Unlock	U <unlock code>	Table 215
Set Baud Rate	B <baud rate> <stop bit>	Table 216
Echo	A <setting>	Table 218
Write to RAM	W <start address> <number of bytes>	Table 219
Read Memory	R <address> <number of bytes>	Table 220
Prepare sector(s) for write operation	P <start sector number> <end sector number>	Table 221
Copy RAM to Flash	C <flash address> <RAM address> <number of bytes>	Table 222
Go	G <address> <mode>	Table 223
Erase sector(s)	E <start sector number> <end sector number>	Table 224
Blank check sector(s)	I <start sector number> <end sector number>	Table 225
Read Part ID	J	Table 226
Read Boot code version	K	Table 228
Compare	M <address1> <address2> <number of bytes>	Table 229

25.9.1 Unlock <unlock code>

Table 215. ISP Unlock command

Command	U
Input	Unlock code: 23130_{10}
Return Code	CMD_SUCCESS | INVALID_CODE | PARAM_ERROR
Description	this command unlocks flash Write, Erase, and Go commands
Example	"U 23130<CR><LF>" unlocks the flash Write, Erase and Go commands

25.9.2 Set Baud Rate <baud rate> <stop bit>

Table 216. 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 changes 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 217. Correlation between possible ISP baud rates and external crystal frequency (MHz)

ISP Baud rate 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	+	+	+			

25.9.3 Echo <setting>

Table 218. 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

25.9.4 Write to RAM <start address> <number of bytes>

The host must send the data only after receiving the CMD_SUCCESS return code. The host must 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 must not exceed 61 characters (bytes) that is, it can hold 45 data bytes. If the data fits in less than 20 UU-encoded lines, then the check-sum must 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 must retransmit the bytes.

Table 219. ISP Write to RAM command

Command	W
Input	**start address**: RAM address where data bytes are to be written. This address must be a word boundary. **number of bytes:** number of bytes to be written. Count must 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	downloads data to RAM. Data must be in UU-encoded format. Blocked when code read protection is enabled.
Example	"W 1073742336 4<CR><LF>" writes 4 bytes of data to address 0x4000 0200

25.9.5 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 must not exceed 61 characters (bytes) that is, it can hold 45 data bytes. When the data fits in less than 20 UU-encoded lines, then the check-sum is the actual number of bytes sent. The host must compare it with the checksum of the received bytes. If the check-sum matches then the host must respond with "OK<CR><LF>" to continue further transmission. If the check-sum does not match, the host must respond with "RESEND<CR><LF>". In response, the ISP command handler sends the data again.

Table 220. ISP Read memory command

Command	R
Input	**start address:** address from where data bytes are to be read. This address must be a word boundary. **number of bytes:** number of bytes to be read. Count must 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	reads data from RAM or flash memory. Blocked when code read protection is enabled.
Example	"R 1073741824 4<CR><LF>" reads 4 bytes of data from address 0x4000 0000

25.9.6 Prepare sector(s) for write operation <start sector number> <end sector number>

This command makes flash write/erase operation a two-step process.

Table 221. ISP Prepare sector(s) for write operation command

Command	P
Input	**start sector number** **end sector number:** must be greater than or equal to start sector number
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR | PARAM_ERROR
Description	must be executed before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot block cannot be prepared by this command. To prepare a single sector, use the same "Start" and "End" sector numbers.
Example	"P 0 0<CR><LF>" prepares flash sector 0

25.9.7 Copy RAM to flash <flash address> <RAM address> <no of bytes>

Table 222. ISP Copy command

Command	C
Input	**flash address (DST):** destination flash address where data bytes are to be written. Destination address must 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. Must 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	programs flash memory. The "Prepare Sector(s) for Write Operation" command must 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. Blocked when code read protection is enabled.
Example	"C 0 1073774592 512<CR><LF>" copies 512 bytes from the RAM address 0x4000 8000 to flash address 0

25.9.8 Go <address> <mode>

Table 223. ISP Go command

Command	G
Input	**address:** flash or RAM address from which the code execution is to start. This address must 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	executes a program residing in RAM or flash memory. It cannot be possible to return to the ISP command handler once this command is successfully executed. Blocked when code read protection is enabled.
Example	"G 0 A<CR><LF>" branches to address 0x0000 0000 in ARM mode

25.9.9 Erase sector(s) <start sector number> <end sector number>

Table 224. ISP Erase sector command

Command	E
Input	**start sector number** **end sector number:** Must 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	erases one or more sector(s) of on-chip flash memory. The boot block cannot be erased using this command. This command only allows erasure of all user sectors when code read protection is enabled.
Example	"E 2 3<CR><LF>" erases flash sectors 2 and 3

25.9.10 Blank check sector(s) <sector number> <end sector number>

Table 225. ISP Blank check sector command

Command	I
Input	**start sector number:** **end sector number:** Must 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	blank-checks one or more sectors of on-chip flash memory **blank check on sector 0 always fails as first 64 bytes are** remapped **to flash boot block**
Example	"I 2 3<CR><LF>" blank checks flash sectors 2 and 3

25.9.11 Read part identification number

Table 226. ISP Read part identification number command

Command	J
Input	none
Return Code	CMD_SUCCESS followed by part identification number in ASCII; see Table 227
Description	reads the part identification number

Table 227. MPT612 part identification numbers

Device	ASCII/dec coding	Hex coding
MPT612	327441	0x0004 FF11

25.9.12 Read boot code version number

Table 228. 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	reads the boot code version number

25.9.13 Compare <address1> <address2> <no of bytes>

Table 229. ISP Compare command

Command	M
Input	**address1 (DST):** starting flash or RAM address of data bytes to be compared. address must be a word boundary. **address2 (SRC):** starting flash or RAM address of data bytes to be compared. Address must be a word boundary. **number of bytes:** number of bytes to be compared; must 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	compares 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** remapped **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

25.9.14 ISP Return codes

Table 230. ISP Return codes summary

Return code	Mnemonic	Description
0	CMD_SUCCESS	command 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 considered where applicable.
5	DST_ADDR_NOT_MAPPED	destination address is not mapped in the memory map. Count value is considered where applicable.
6	COUNT_ERROR	byte count is not a multiple of 4 or is not a permitted value
7	INVALID_SECTOR	sector number is invalid or end sector number is greater than start sector number
8	SECTOR_NOT_BLANK	sector is not blank
9	SECTOR_NOT_PREPARED_FOR_WRITE_OPERATION	command to prepare sector for write operation was not executed
10	COMPARE_ERROR	source and destination data not equal
11	BUSY	flash programming hardware interface is busy
12	PARAM_ERROR	insufficient number of parameters or invalid parameter
13	ADDR_ERROR	address is not on word boundary

Table 230. ISP Return codes summary *...continued*

Return code	Mnemonic	Description
14	ADDR_NOT_MAPPED	address is not mapped in the memory map. Count value is considered 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

25.10 IAP commands

An In Application Programming routine must 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 must be able to hold all the results in case the number of results are more than the number of parameters. Parameter passing is illustrated in Figure 67. 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 is Thumb code.

The IAP function can 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[2];
```

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;
```

The following statement can be used to call IAP:

```
iap_entry (command, result);
```

The IAP call can be simplified further using the symbol definition file feature supported by ARM Linker in ADS (ARM Developer Suite). Assembly code can also be used to call the IAP routine.

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 (ARM Thumb Procedure Call Standard SWS ESPC 0002 A-05), up to 4 parameters can be passed in registers r0, r1, r2 and r3 respectively. Additional parameters are passed on the stack. Up to 4 parameters can be returned in registers r0, r1, r2 and r3 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 may create problems due to differences in the C compiler implementation from different vendors. The suggested parameter passing scheme reduces such a 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. If IAP flash programming is permitted in the application, the user program must not use this space.

Table 231. IAP command summary

IAP Command	Command code	Described in
Prepare sector(s) for write operation	50_{10}	Table 232
Copy RAM to Flash	51_{10}	Table 233
Erase sector(s)	52_{10}	Table 234
Blank check sector(s)	53_{10}	Table 235
Read Part ID	54_{10}	Table 236
Read Boot code version	55_{10}	Table 237
Compare	56_{10}	Table 238
Reinvoke ISP	57_{10}	Table 239

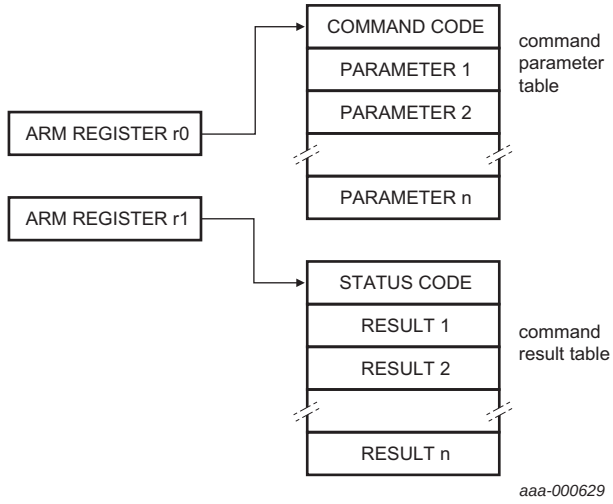


Fig 67. IAP Parameter passing

25.10.1 Prepare sector(s) for write operation

This command makes flash write/erase operation a two-step process.

Table 232. IAP Prepare sector(s) for write operation command

Command	Prepare sector(s) for write operation
Input	**command code: 5010** **param0:** start sector number **param1:** end sector number (must be greater than or equal to start sector number)
Return Code	CMD_SUCCESS | BUSY | INVALID_SECTOR
Result	none
Description	execute before executing "Copy RAM to Flash" or "Erase Sector(s)" command. Successful execution of the "Copy RAM to Flash" or "Erase Sector(s)" command causes relevant sectors to be protected again. The boot sector cannot be prepared by this command. To prepare a single sector, use the same "Start" and "End" sector numbers.

25.10.2 Copy RAM to flash

Table 233. IAP Copy RAM to flash command

Command	Copy RAM to flash
Input	**command code: 5110** **param0(DST):** destination flash address where data bytes are to be written. Address must be a 256 byte boundary. **param1(SRC):** source RAM address from which data bytes are to be read. Address must be a word boundary. **param2:** number of bytes to be written. Must 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	programs flash memory. Affected sectors must be prepared first by calling "Prepare Sector for Write Operation" command. Affected sectors are automatically protected again once the copy command is successfully executed. The boot sector cannot be written by this command.

25.10.3 Erase sector(s)

Table 234. IAP Erase sector(s) command

Command	Erase sector(s)
Input	**command code: 5210** **param0:** start sector number **param1:** end sector number (must 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	erases a sector or multiple sectors of on-chip flash memory. The boot sector cannot be erased by this command. To erase a single sector, use the same "Start" and "End" sector numbers.

25.10.4 Blank check sector(s)

Table 235. IAP Blank check sector(s) command

Command	Blank check sector(s)
Input	**command code: 5310** **param0:** start sector number **param1:** end sector number (must 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	blank-checks a sector or multiple sectors of on-chip flash memory. To blank-check a single sector, use the same "Start" and "End" sector numbers.

25.10.5 Read part identification number

Table 236. IAP Read part identification number command

Command	Read part identification number
Input	**command code: 5410** **parameters:** none
Return Code	CMD_SUCCESS |
Result	**result0:** part identification number; see Table 227 on page 230 for details
Description	reads the part identification number

25.10.6 Read boot code version number

Table 237. IAP Read boot code version number command

Command	Read boot code version number
Input	**command code: 5510** **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	reads the boot code version number

25.10.7 Compare <address1> <address2> <no of bytes>

Table 238. IAP Compare command

Command	Compare
Input	**command code: 5610** **param0(DST):** starting flash or RAM address of data bytes to be compared. Address must be a word boundary. **param1(SRC):** starting flash or RAM address of data bytes to be compared. Address must be a word boundary. **param2:** number of bytes to be compared; must 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	compares the memory contents at two locations. **the result may not be correct if the source or destination includes any of the first 64 bytes starting from address zero. The first 64 bytes can be remapped to RAM.**

25.10.8 Reinvoke ISP

Table 239. Reinvoke ISP

Command	Compare
Input	**command code: 5710**
Return Code	none
Result	none
Description	invokes the bootloader in ISP mode. Maps boot vectors, configures PIO1 as an input and sets the APB divider register to 0 before entering the ISP mode. This command can be used when a valid user program is present in the internal flash memory and the PIO14 pin is not accessible to force the ISP mode. This command does not disable the PLL hence it is possible to invoke the bootloader when the part is running off the PLL. In such cases, the ISP utility must pass the PLL frequency after autobaud handshake. Another option is to disable the PLL before making this IAP call. **Remark:** TIMER1 registers must be programmed with reset values before "Reinvoke ISP" command is used.

25.10.9 IAP status codes

Table 240. 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 considered where applicable.
5	DST_ADDR_NOT_MAPPED	destination address is not mapped in the memory map. Count value is considered where applicable.
6	COUNT_ERROR	byte count is not a 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

25.11 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.

26. EmbeddedICE logic

26.1 Features

- In order to start the debugging session, no target resources are required by the software debugger
- 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
- ARM7TDMI-S core or System state can be examined, saved, or changed depending on type of instruction inserted
- Instructions can be executed at a slow debug speed or at a fast system speed

26.2 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. The 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.

26.3 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 (that is, on a data access) or a break point (that is, 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 would be to set the first breakpoint to 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 is ready for examination.

1. For more details, refer to *IEEE Standard 1149.1 - 1990 Standard Test Access Port and Boundary Scan Architecture*.

- 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 occurs 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 built-in. 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.

26.4 Pin description

Table 241. EmbeddedICE pin description

Pin name	Type	Description
TMS	input	**test mode select.** TMS pin selects the next state in the TAP state machine.
TCK	input	**test clock.** Allows shifting of data in, on TMS and TDI pins. It is a positive edge-triggered clock with TMS and TCK signals that define the internal state of the device. **Remark:** Clock must be slower than $\frac{1}{6}$ of the CPU clock (CCLK) for the JTAG interface to operate.
TDI	input	**test data in.** Serial data input for the shift register
TDO	output	**test data output.** Serial data output from shift register. Data is shifted out of the device on the negative edge of TCK signal.
TRST	input	**test reset.** TRST pin can be used to reset the test logic within EmbeddedICE logic.
JTAGSEL	input	**debug select.** When LOW at reset, pins PIO27 to PIO31 are configured for alternate use via the pin connect block. When HIGH at reset, the debug mode is entered. For functionality provided by JTAGSEL; see Section 26.8 on page 241.
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.

26.5 Reset state of multiplexed pins

On the MPT612, pins TMS, TCK, TDI, TDO, and TRST are multiplexed with PIO27 to PIO31. To make them occur as a debug port, JTAGSEL must be held HIGH during and after reset. Additionally, pin RTCK must be HIGH when the reset is released. Pin RTCK can be driven HIGH externally or allowed to float HIGH via its on-chip pull-up. To make them occur 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.

26.6 Register description

The EmbeddedICE logic contains 16 registers as shown in Table 242. 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 242. EmbeddedICE logic registers

Name	Width	Description	Address
Debug Control	6	force debug state, disable interrupts	0 0000
Debug Status	5	debug status	0 0001
Debug Comms Control register	32	debug communication control register	0 0100
Debug Comms Data register	32	debug communication data register	0 0101
Watchpoint 0 Address Value	32	holds watchpoint 0 address value	0 1000
Watchpoint 0 Address Mask	32	holds watchpoint 0 address mask	0 1001
Watchpoint 0 Data Value	32	holds watchpoint 0 data value	0 1010
Watchpoint 0 Data Mask	32	holds watchpoint 0 data mask	0 1011
Watchpoint 0 Control Value	9	holds watchpoint 0 control value	0 1100
Watchpoint 0 Control Mask	8	holds watchpoint 0 control mask	0 1101
Watchpoint 1 Address Value	32	holds watchpoint 1 address value	1 0000
Watchpoint 1 Address Mask	32	holds watchpoint 1 address mask	1 0001
Watchpoint 1 Data Value	32	holds watchpoint 1 data value	1 0010
Watchpoint 1 Data Mask	32	holds watchpoint 1 data mask	1 0011
Watchpoint 1 Control Value	9	holds watchpoint 1 control value	1 0100
Watchpoint 1 Control Mask	8	holds watchpoint 1 control mask	1 0101

26.7 Block diagram

The block diagram of the debug environment is shown in Figure 68.

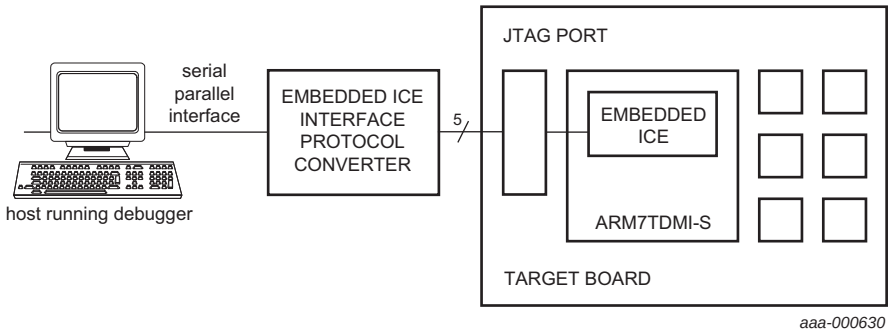


Fig 68. EmbeddedICE debug environment block diagram

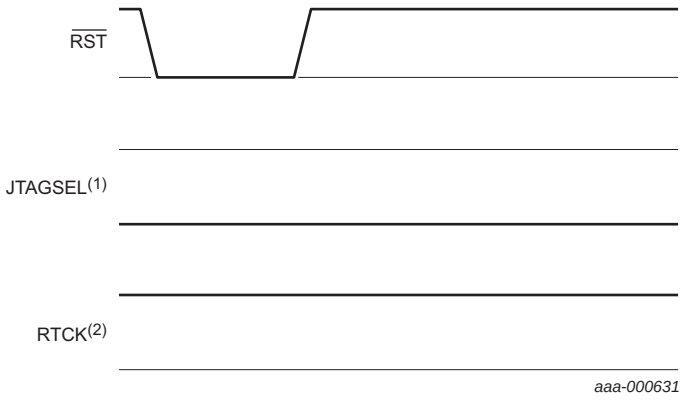
26.8 Debug mode

Debug mode connects the JTAG pins to the EmbeddedICE for program debugging using an emulator or other development tool.

26.8.1 Enable debug mode

Debug mode is enabled using pins JTAGSEL and RTCK.

To enable Debug mode, JTAGSEL must be HIGH during and after the CPU is reset. Keep JTAGSEL LOW at all times for normal (non-debug) operation; see Figure 69.



(1) JTAGSEL is tied or pulled LOW at all times. An internal pull-down causes JTAGSEL to be LOW if it is not pulled HIGH externally.

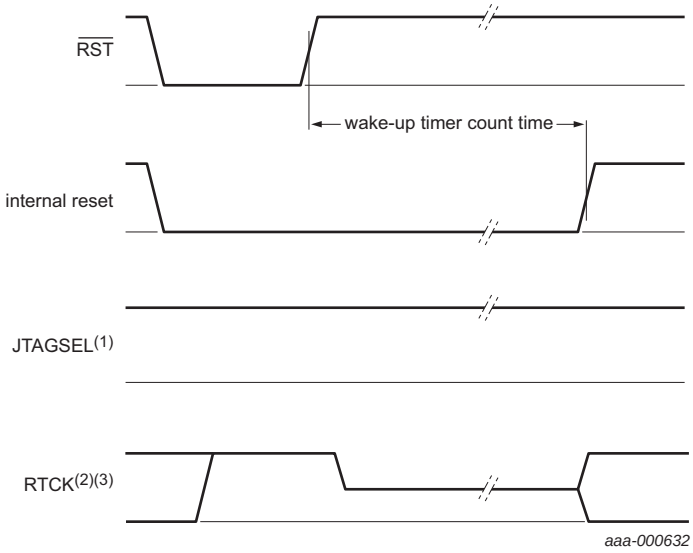
(2) RTCK is not connected in the application and is pulled up internally.

Fig 69. Waveforms for normal operation (not in Debug mode)

For debugging with JTAG pins, RTCK must be HIGH as the RST pin is released; see Figure 70. RTCK can 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 can be driven by an external signal if necessary.

This procedure establishes the PIO27 to PIO31 pins as the JTAG test/debug interface. Pin connect block settings have no affect on PIO27 to PIO31 pins if they are initialized as JTAG pins.

For the effect of hardware override related to JTAGSEL and RTCK, see Section 11.2 “MPT612 pin description” on page 58.



(1) JTAGSEL must be HIGH.

(2) RTCK must be HIGH as $\overline{\text{RST}}$ is released. An internal pull-up causes RTCK to be HIGH if it is not pulled LOW externally.

(3) The RTCK output driver is turned on when the internal chip reset is released by the wake-up timer.

Fig 70. Waveforms for Debug mode using the primary JTAG pins

26.8.2 JTAG pin selection

The primary JTAG port can be selected for debugging only when pins JTAGSEL and RTCK are HIGH at reset; see Figure 70. If at least one of the JTAGSEL or RTCK lines is LOW at reset, JTAG is not enabled and cannot be used for later debugging.

27. RealMonitor

27.1 Features

- Allows a user debug session to a currently running system to be established without halting or resetting the system
- Allows user time-critical interrupt code to continue executing while other user application code is being debugged

Remark: RealMonitor is a configurable software module which enables real time debug. RealMonitor is developed by ARM Inc. Information presented in this section is taken from the ARM document *RealMonitor Target Integration Guide (ARM DUI 0142A)*. It applies to a specific configuration of RealMonitor software programmed in the on-chip ROM boot memory of this device.

Refer to the white paper *"Real Time Debug for System-on-chip"*, available at the official ARM website, for background information.

27.2 Applications

Real-time debugging.

27.3 Description

RealMonitor is a lightweight debug monitor that allows interrupts to be serviced while the user debugs his 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 various modes, and communicate with the debug host using various 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 the user to observe and debug while parts of the application continue to run. Refer to Section 27.4 for details.

27.3.1 RealMonitor components

As shown in Figure 71, RealMonitor is split in to two functional components.

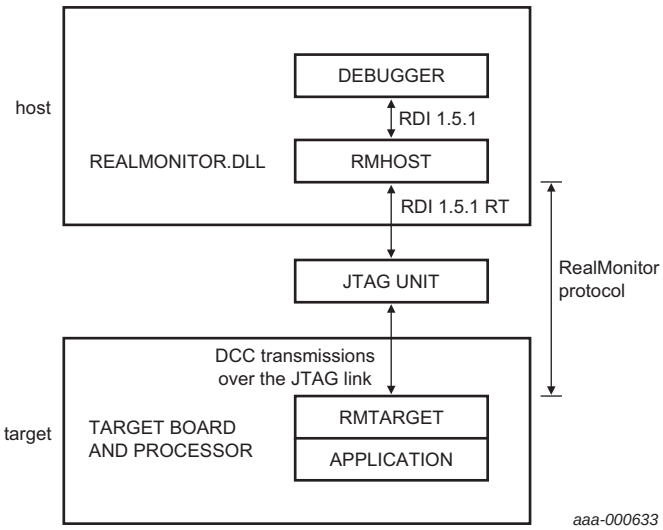


Fig 71. RealMonitor components

27.3.2 RMHost

RMHost 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)*.

27.3.3 RMTarget

RMTarget 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)*.

27.3.4 How RealMonitor works

In general terms, the RealMonitor operates as a state machine, as shown in Figure 72. 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 the 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.

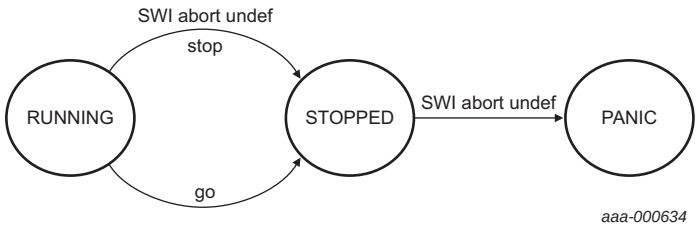


Fig 72. RealMonitor as a state machine

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 71.

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 the user application is running, RMTarget typically uses IRQs generated by the DCC. This means that if the user application also wants to use IRQs it must pass any DCC-generated interrupts to RealMonitor.

To allow non-stop 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 the user application to continue running without stopping the processor. RealMonitor considers the 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 triggered by certain events in the 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 the 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.

If one of these exceptions occurs and not handled by the user application, the following occurs:

- 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 a read from the DCC a higher priority than a write 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.

27.4 How to enable RealMonitor

To enable RealMonitor, the following steps must be performed. A code example which implements all the steps can be found at the end of this section.

27.4.1 Adding stacks

The user must ensure that stacks are set up within the application for each of the processor modes used by RealMonitor. For each mode, RealMonitor requires a fixed number of words of stack space. The user must therefore allow sufficient stack space for both RealMonitor and application.

RealMonitor stack requirements are shown in Table 243.

Table 243. RealMonitor stack requirement

Processor mode	RealMonitor stack usage (bytes)
Undef	48
Prefetch Abort	16
Data Abort	16
IRQ	8

27.4.2 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.

27.4.3 Undef mode

A stack for this mode is always required. RealMonitor uses 12 words while processing an undefined instruction exception.

27.4.4 SVC mode

RealMonitor makes no use of this stack.

27.4.5 Prefetch abort mode

RealMonitor uses four words on entry to its Prefetch abort interrupt handler.

27.4.6 Data abort mode

RealMonitor uses four words on entry to its data abort interrupt handler.

27.4.7 User/System mode

RealMonitor makes no use of this stack.

27.4.8 FIQ mode

RealMonitor makes no use of this stack.

27.4.9 Handling exceptions

This section describes the importance of sharing exception handlers between RealMonitor and the user application.

27.4.10 RealMonitor exception handling

To function properly, RealMonitor must be able to intercept certain interrupts and exceptions. Figure 73 illustrates how exceptions can be claimed by RealMonitor itself, or shared between RealMonitor and the application. If the user application requires exception sharing, they must provide a 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.

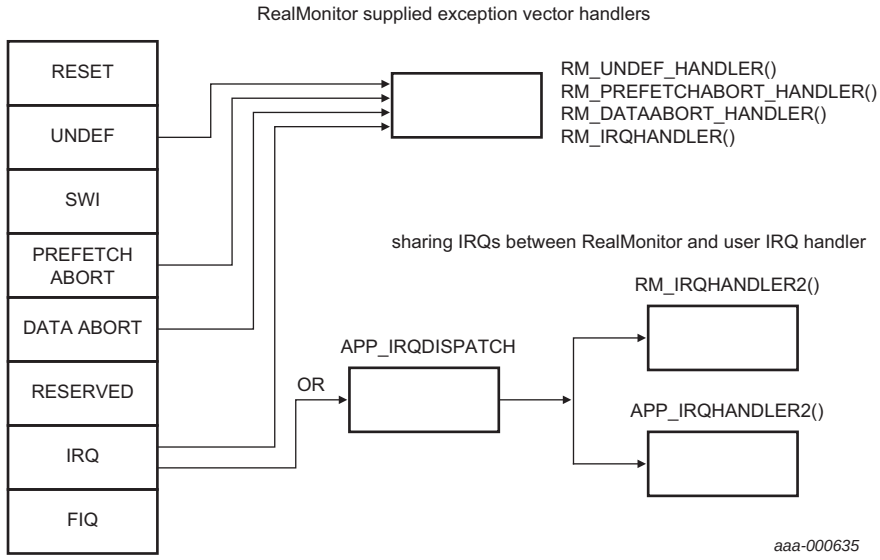


Fig 73. Exception handlers

27.4.11 RMTarget initialization

While the processor is in a privileged mode, and IRQs are disabled, the user must include a line of code within the start-up sequence of the application to call rm_init_entry().

27.4.12 Code example

The following example shows how to set up the 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 flash programming routines.
        ;Refer to Flash Memory System and Programming 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 set up
; * 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
```

27.5 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 using 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 or disables support for 8/16/32-bit read/write.

RM_OPT_EXECUTECODE=FALSE

Enables or 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 RealMonitor 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. The 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).

28. Abbreviations

Table 244. Abbreviations

Acronym	Description
ADC	Analog-to-Digital Converter
AMBA	Advanced Microcontroller Bus Architecture
APB	ARM Peripheral Bus
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DCC	Debug Communications Channel
FIFO	First In, First Out
FIQ	Fast Interrupt reQuest
GPIO	General-Purpose Input/Output
JTAG	Joint Test Action Group
IAP	In-Application Programming
IRQ	Interrupt Request
ISP	In-System Programming
MPPT	Maximum Power Point Tracking
NA	Not Applicable
PLL	Phase-Locked Loop
PV	PhotoVoltaic
PWM	Pulse-Width Modulator
RAM	Random Access Memory
SPI	Serial Peripheral Interface
RO	Read Only

Table 244. Abbreviations *...continued*

Acronym	Description
SRAM	Static Random Access Memory
SSP	Synchronous Serial Port
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
VIC	Vector Interrupt Controller
WO	Write Only

29. Legal information

29.1 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.

29.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

29.3 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.

30. Tables

continued >>

continued >>

continued >>

continued >>

31. Figures

continued >>

continued >>

32. Contents

continued >>

continued >>

continued >>

continued >>

continued >>

continued >>

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 16 December 2011
Document identifier: UM10413