Cortex™-M1 FPGA Development Kit

Altera Edition version 1.1

Cortex-M1 User Guide

ARM®

Copyright © 2007, 2008 ARM Limited. All rights reserved.
ARM DUI 0395B

Cortex-M1 FPGA Development Kit
Cortex-M1 User Guide

Copyright © 2007, 2008 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
21 November 2007	A	Non-Confidential	First release for version 1.0
22 August 2008	B	Non-Confidential	First release for version 1.1

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

Altera is a trademark and service mark of Altera Corporation in the United States and other countries. Altera products contain intellectual property of Altera Corporation and are protected by copyright laws and one or more U.S. and foreign patents and patent applications.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Contents
Cortex-M1 FPGA Development Kit Cortex-M1 User Guide

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

List of Tables
Cortex-M1 FPGA Development Kit Cortex-M1 User Guide

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

List of Figures
Cortex-M1 FPGA Development Kit Cortex-M1 User Guide

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Preface

This preface introduces the *Cortex-M1 FPGA Development Kit Altera Edition Cortex-M1 User Guide*. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

About this guide

This is the *Cortex-M1 User Guide* (UG) for the processor in the *Cortex-M1 FPGA Development Kit Altera Edition*.

Intended audience

This guide is written for FPGA system designers and programmers who want to incorporate and program the Cortex-M1 processor in their own SoC design using an Altera FPGA and Altera *System-On-a-Programmable-Chip* (SOPC) Builder. This guide assumes the intended audience is experienced in using SOPC Builder.

Using this guide

This guide is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for a description of the processor features and a functional overview.

Chapter 2 ***Programmer's Model***

Read this chapter for a description of programmer's model and the processor memory map.

Chapter 3 ***Processor Instantiation and Configuration***

Read this chapter for a description of how to instantiate and configure the processor.

Chapter 4 ***System Timer***

Read this chapter for a description of the system timer, the registers that it uses, and examples of how to use it.

Chapter 5 ***Nested Vectored Interrupt Controller***

Read this chapter for a description of the *Nested Vectored Interrupt Controller* (NVIC), the registers that it uses, and examples of how to use it.

Chapter 6 ***System Control Block***

Read this chapter for a description of the system control block, the registers that it uses, and examples of how to use it.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Appendix A *Embedded Software Examples*

Read this chapter for descriptions of the example embedded software, and the C declarations that the examples use.

Glossary Read the Glossary for definitions of terms used in this guide.

Conventions

Conventions that this guide uses are described in:

- *Typographical*
- *Numbering* on page xiv.

Typographical

The typographical conventions are:

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace` Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic` Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold` Denotes language keywords when used outside example code.

< and > Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals
- LOW for active-LOW signals.

Lower-case n At the start or end of a signal name denotes an active-LOW signal.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of `0x3F`. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Additional reading

This section lists publications by ARM and by third parties.

See:

- `http://infocenter.arm.com/help/index.jsp` for access to ARM documentation
- `http://www.altera.com` for access to Altera documentation.

ARM publications

This guide contains information that is specific to the processor in the Cortex-M1 FPGA Development Kit Altera Edition. See the following documents for other relevant information:

- *ARMv6-M Instruction Set Quick Reference Guide* (QRC0011)
- *Application Binary Interface for the ARM Architecture (The Base Standard)* (IHI0036)

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Other publications

This section lists relevant documents published by third parties:

- *Altera Corporation, Avalon Memory-Mapped Interface Specification.*

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms if appropriate.

Feedback on this book

If you have any comments on this book, send an e-mail to `errata@arm.com`. Give:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 1
Introduction

This chapter introduces the processor and describes its main features. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

1.1 About the processor

The ARM Cortex-M1 processor has been developed specifically for implementation in FPGA. The Altera edition of the ARM Cortex-M1 processor is provided in a format that enables integration using Altera Quartus II Software, which contains the SOPC Builder tool, and Altera FPGA devices. Version 1.1 of the Altera edition of the Cortex-M1 processor is based on revision r0p1 of the Cortex-M1 processor core. The processor features include:

- A general-purpose 32-bit microprocessor, which executes the ARMv6-M subset of the Thumb-2 instruction set and offers high performance operation and small size in FPGAs. It has:
  - a three-stage pipeline
  - a three-cycle hardware multiplier
  - little-endian format for accessing all memory.
- A system control block containing memory-mapped control registers.
- An integrated *Operating System* (OS) extensions system timer.
- An integrated *Nested Vectored Interrupt Controller* (NVIC) for low-latency interrupt processing.
- A memory model that supports accesses to both memory and peripheral registers.
- An Avalon-MM master interface that enables you to integrate the processor into systems using the Altera SOPC Builder tool.
- Integrated and configurable *Tightly Coupled Memories* (TCMs)
- Optional debug support.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

1.2 Functional overview

Figure 1-1 shows functional blocks of the processor in a debug-enabled configuration.

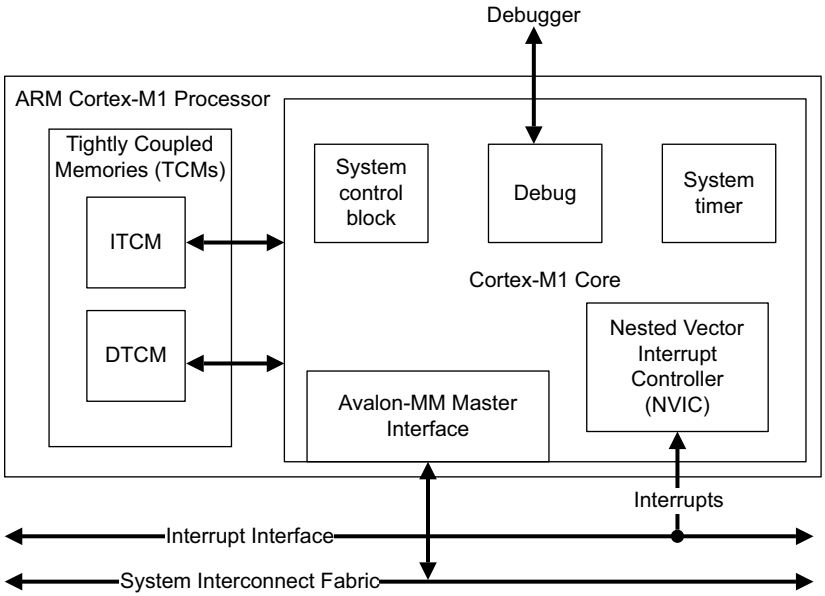

Figure 1-1 Processor block diagram

The blocks are described in the following sections:

- *Cortex-M1 Core*
- *TCMs* on page 1-4.

1.2.1 Cortex-M1 Core

The Cortex-M1 Core comprises:

System control block

The system control block provides processor status and control functions including exception management. For more information, see Chapter 6 *System Control Block*.

System timer Integrated 24-bit system timer. You can use this as a *Real Time Operating System* (RTOS) tick timer or a simple counter. For more information, see Chapter 4 *System Timer*.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

NVIC The NVIC is an embedded interrupt controller supporting low latency interrupt processing. Up to 32 external interrupts are supported. For more information, see Chapter 5 *Nested Vectored Interrupt Controller*.

Avalon-MM master interface

The Avalon-MM Master interface enables the processor to be integrated into systems that are constructed using SOPC Builder. All external processor reads and writes are through this interface, which is capable of pipelined transfers. For more information on the Avalon-MM protocol, see the *Avalon Memory-Mapped Interface Specification, Altera Corporation*.

Optional debug When implemented within the processor, the optional debug logic enables connection to a debugger. The debug features:

- processor halt
- processor register access
- processor stepping.

The optional debug logic also facilitates:

- watchpoints for monitoring variable data at a specific point
- breakpoints for stopping on a particular line of code.

1.2.2 TCMs

The TCMs are low-latency memories internal to the processor. There is a separate *Instruction Tightly Coupled Memory* (ITCM) for instruction accesses and *Data Tightly Coupled Memory* (DTCM) for data accesses. Each of these has a configurable size from 0KB-64KB. Highest performance is achieved when code is executing from the TCMs. For more information on configuring the TCMs see *Processor configuration* on page 3-4.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 2
Programmer's Model

This chapter describes the programmer's model of the processor. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1 About the programmer's model

The processor implements the ARMv6-M architecture and executes the ARMv6-M subset of the Thumb-2 instruction set. For information on the instructions that you can use, see the *ARMv6-M Instruction Set Quick Reference Guide*.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2 Modes of operation and execution

The processor has two modes of operation, Thread mode and Handler mode:

- Thread mode is entered on reset and is the fundamental mode for application execution.
- Handler mode is entered as a result of an exception. All exceptions execute in Handler mode.

Execution is always Privileged in the processor. This means that application code has access to all system resources.

The processor has two banked *Stack Pointer* (SP) registers for use in Thread mode and Handler mode:

Main This stack pointer can be used in either Thread or Handler modes

Process This stack pointer can only be used in Thread mode.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.3 Registers

The processor has the following registers in its programmer's model:

- general purpose registers R0-R12
- two SP registers, SP_main and SP_process (banked versions of R13)
- *Link Register* (LR), R14
- the *Program Counter* (PC), R15
- Program status register for flags, exception and interrupt level, and execution state bits.

Figure 2-1 shows the processor register set.

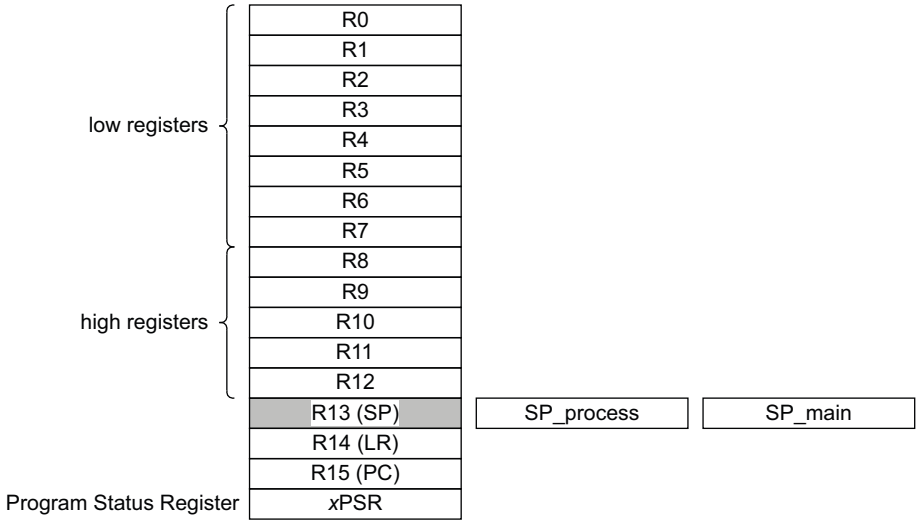

Figure 2-1 Processor register set

2.3.1 General purpose registers

The general purpose registers, R0-R12, are split into the low registers and high registers:

- The low registers, R0-R7, are accessible by most instructions
- The high registers, R8-R12, are only accessible by a smaller set instructions.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.3.2 Stacks

The processor supports two separate stacks:

Process stack You can configure Thread mode to use either SP_process or SP_main for its stack pointer. By default Thread Mode uses SP_main out of reset.

Main stack Handler mode uses SP_main.

Only one SP register, SP_process or SP_main, is visible at any time, using R13. On exception entry, the processor pushes the pre-empted context onto the stack that is active at the time. In handler mode, only SP_main can be active and is used for stacking the context of any nested exceptions. The stack pointer is selected as follows:

- Thread mode can use SP_main or the SP_process stack, depending on the value of bit [1] of the CONTROL register that an `MSR` or `MRS` instruction can access. Appropriate EXC_RETURN values can also set this bit when exiting an exception. An exception that pre-empts a user thread saves the context of the user thread on the stack that the Thread mode is using.
- Exception handlers always use SP_main.

Note

`MSR` and `MRS` instructions have visibility of both stack pointers.

2.3.3 Link Register

The LR stores a value relating to the return address, known as the return link, from a subroutine that is entered using a branch with link instruction. Exception entry uses the LR to provide exception return information. You can use the LR as a general-purpose register when it is not being used as a return link.

2.3.4 Program Counter

On reset the PC is loaded with the address of the reset handler. Bit[0] of the PC is always 0 because instruction fetches are always halfword aligned.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.4 Program status register

The program status register consists of three individual status registers:

- *Application Processor Status Register* (APSR), containing the condition code flags
- *Interrupt Processor Status Register* (IPSR), containing the *Interrupt Service Routine* (ISR) number of the current exception.
- *Execution Processor Status Register* (EPSR), containing the Thumb state bit.

You can access these registers individually or as a combination of any two or all three registers using the `MSR` and `MRS` instructions.

Figure 2-2 shows the bit assignments of the *APSR*, *IPSR*, and *EPSR* (xPSR) registers.

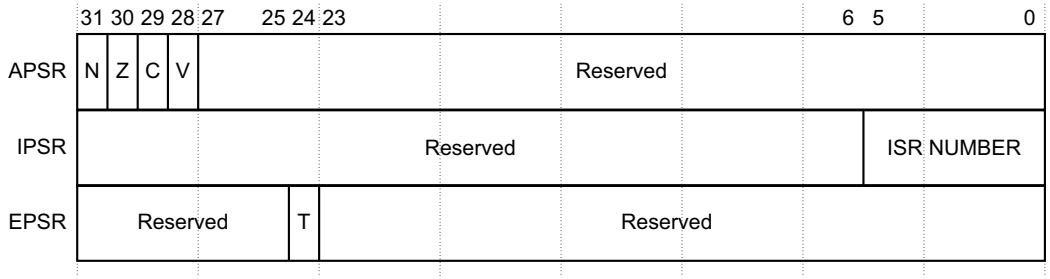

Figure 2-2 Bit assignments for the program status registers

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.4.1 Application Processor Status Register

Table 2-1 lists the bit assignments of the APSR.

Table 2-1 APSR bit assignments

Field	Name	Description
31	N	Negative or less than flag: 1 = result negative or less than 0 = result positive or greater than
30	Z	Zero flag: 1 = result of zero 0 = nonzero result
29	C	Carry or borrow flag: 1 = carry true or borrow false 0 = carry false or borrow true
28	V	Overflow flag: 1 = overflow 0 = no overflow
[27:0]	-	Reserved.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.4.2 Interrupt Processor Status Register

Table 2-2 lists the bit assignments of the IPSR.

Table 2-2 IPSR bit assignments

Field	Name	Description
[31:6]	-	Reserved
[5:0]	ISR_ NUMBER	This is the number of current exception: Thread mode = 0 NMI = 2 HardFault = 3 Supervisor Call = 11 PendSV = 14 SysTick = 15 IRQ0 = 16 … IRQ31 = 47 For more information, see *Exceptions* on page 2-12.

2.4.3 Execution Processor Status Register

Only a debugger on a debug version of the processor can access the EPSR. Attempts to read the EPSR directly through application code using the `MSR` instruction always returns zero. Attempts to write the EPSR using the MSR instruction in application code are ignored.

Table 2-3 lists the bit assignments of the EPSR.

Table 2-3 EPSR bit assignments

Field	Name	Description
[31:25]	-	Reserved
[24]	T	Indicates that the processor is in Thumb state. The T-bit is set according to the reset vector when the processor comes out of reset. Because the processor only executes Thumb instructions, the execution of an instruction with the EPSR T-bit clear causes a HardFault. This ensures that attempts to switch to ARM state fail in a predictable way.
[23:0]	-	Reserved

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.4.4 Special-Purpose Priority Mask Register

The *Special-Purpose Priority Mask Register* (SPPMR) is used for priority boosting, that is, raising the execution priority to prevent exceptions with configurable priority from activating inside a particular section of code.

Figure 2-3 shows the bit assignments of the SPPMR.

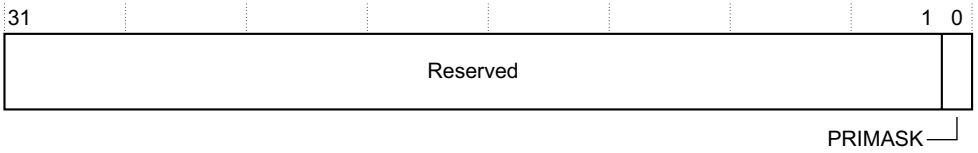

Figure 2-3 SPPMR bit assignments

Table 2-4 lists the bit assignments of the SPPMR.

Table 2-4 SPPMR bit assignments

Field	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	When set, raises execution priority to 0. This is higher priority than all configurable-priority exceptions and no configurable-priority exceptions can be activated, except when the exception escalates to HardFault. Set to 0 on reset.

You can access the Special-Purpose Priority Mask Register using the `MSR` and `MRS` instructions. You can also use the `CPS` instruction to set or clear PRIMASK.

2.4.5 Special-Purpose Control Register

The *Special Purpose Control Register* (SPCR) identifies the currently active stack pointer. By default, Thread mode uses the main stack pointer. You can switch Thread mode to use the process stack by writing to the active stack pointer control bit using the `MSR` instruction.

Note

Handler mode always uses the main stack, so explicit writes to the active stack pointer bit of the CONTROL register when in handler mode are ignored. The CONTROL register is updated as part of the exception entry and return mechanisms.

Figure 2-4 on page 2-10 shows the bit assignments of the SPCR.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

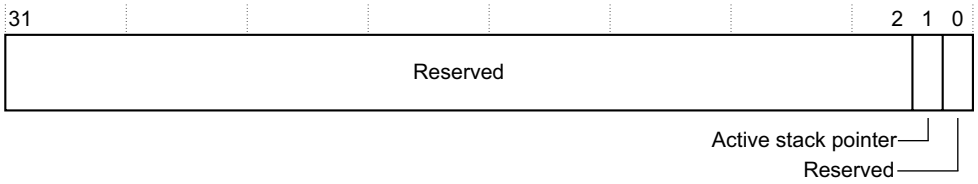

Figure 2-4 SPCR bit assignments

Table 2-5 lists bit assignments of the **SPCR**.

Table 2-5 SPCR bit assignments

Field	Name	Function
[31:2]	-	Reserved
[1]	Active stack pointer	Defines the stack to use: 0 = SP_main is used as the current stack 1 = For Thread mode, SP_process is used for the current stack[a].
[0]	-	Reserved

a. Attempts to set this bit from Handler mode are ignored.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.5 Data types

The processor supports the following data types:

- 32-bit words
- 16-bit halfwords
- 8-bit bytes.

Unless otherwise stated the core can access all regions of the memory map, including the code region, with all data types. To support this, the system, including memories, must support subword writes without corrupting neighboring bytes in that word.

The processor manages all memory accesses as little-endian.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.6 Exceptions

The processor and the *Nested Vectored Interrupt Controller* (NVIC) prioritize and handle all exceptions. All exceptions except for reset are handled in Handler mode. The following features enable efficient, low latency exception handling:

- Automatic state saving and restoring for all exceptions other than reset. The processor pushes state registers on the stack when entering the exception, and pops them when exiting the exception with no instruction overhead.
- Automatic reading of the vector table entry that contains the exception handler address.

  Note

  Vector table entries are ARM or Thumb interworking compatible values. Bit [0] of the vector value is loaded into the EPSR T-bit on exception entry. Bit [0] of the vector table entry must be set to 1 because the processor only executes instructions in Thumb state. Creating a table entry with bit [0] clear generates a HardFault exception on the first instruction of the handler corresponding to this vector.

- Closely-coupled interface between the processor and the NVIC to enable efficient processing of interrupts and processing of late-arriving interrupts with higher priority.
- Configurable number of interrupts, 1, 8 or 32.
- Fixed number of interrupt priorities, at two bits, four levels.
- Separate stacks for Handler and Thread modes.
- Exception control transfer using the calling conventions of the C/C++ standard *ARM Architecture Procedure Call Standard* (AAPCS). For more information, see the *Application Binary Interface for the ARM Architecture (The Base Standard)*.
- Priority masking to support critical regions.
- Automatic handling of level or pulse interrupts.

Note

- The number of interrupts are configured when the processor is instantiated. Software can choose to enable a subset of the configured number of hardware interrupts.
- For examples of software exception handlers, see Appendix A .

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.6.1 Exception types

The types of exceptions that exist in the processor are:

HardFault A HardFault is an exception that occurs as a result of an error during instruction execution or exception processing. Faults are reported synchronously to the instruction that caused them.

Faults are considered fatal, that is, no fault status information is provided to assist recovery.

Interrupt An interrupt is an exception that is triggered from an external source, for example, a system peripheral. All interrupts are asynchronous to the instruction stream. Typically interrupts are used by other peripherals within the system that want to communicate with the processor. Interrupts can be pended by software.

NMI *NonMaskable Interrupt* (NMI). This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. Like an interrupt, an NMI is issued by an external peripheral. It might indicate a critical system condition, such as certain watchdog functions. An NMI can be pended by software.

PendSV PendSV is an interrupt driven request for system service, pended by software.

Reset Reset is a special form of exception. When reset is asserted, the operation of the processor is stopped, potentially at an arbitrary point within an instruction. When reset is de-asserted execution is restarted from the address provided by the reset entry in the vector table.

SVCall *SuperVisor call* (SVCall). An exception that is explicitly caused by the `SVC` instruction. Application code can use a supervisor call to make a system (service) call to an underlying operating system. The `SVC` instruction enables the application to issue a system call for an operating system service and executes in program order with respect to the application.

SysTick An exception generated by the system timer.

Note

A pending, or pended, exception is one where the exception event has been generated but the processor has not yet taken the corresponding exception entry.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 2-6 lists the exception types with their exception number and priority.

The exception number is the word offset of the exception vector from the start of the vector table. The vector table is a table of exception handler addresses, always located at address `0x00000000`.

In the Priority column of the table, lower numbers indicate higher priority. The table also shows how the exception types are activated, synchronously or asynchronously. For more information on priorities, see *Exception priority* on page 2-15

Table 2-6 Exception types

Exception number	Exception type	Priority	Description	Activated
-	-	-	Stack top is loaded from first entry of vector table on reset.	-
1	Reset	-3, the highest	Invoked on power up and warm reset. On first instruction, drops to lowest priority, Thread mode.	Asynchronous
2	NMI	-2	This exception type cannot be: • masked or prevented from activation by any other exception • pre-empted by any other exception other than Reset.	Asynchronous
3	HardFault	-1	All classes of Fault.	Synchronous
4-10	-	-	Reserved.	-
11	SVCall	Configurable	SVCall using the `SVC` instruction.	Synchronous
12-13	-	-	Reserved.	-
14	PendSV	Configurable	Pendable request for system service. This is only pended by software.	Asynchronous
15	System Timer, SysTick	Configurable	System timer has reached zero.	Asynchronous
16-47	External Interrupt, **IRQ[0]**-**IRQ[31]**	Configurable	Asserted from outside the processor.	Asynchronous

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.6.2 Exception priority

The NVIC supports software-assigned priority levels for external interrupts, SVCall, PendSV, and SysTick. You can assign a positive or zero priority level from 0 to 3 to an interrupt by writing to the two-bit IP_N field in an Interrupt Priority Register, see *Interrupt Priority Registers* on page 5-7. Priority level -3 is the highest priority, and priority level 3 is the lowest. For example, if you assign priority level 1 to **IRQ[0]** and priority level 0 to **IRQ[31]**, then **IRQ[31]** has priority over **IRQ[0]**.

Note

Reset, NMI, and HardFault have fixed priorities that cannot be changed by software. They always have higher priority than the other exceptions.

When multiple exceptions have the same priority number, the pending exception with the lowest exception number takes precedence. For example, if both **IRQ[0]** and **IRQ[1]** are priority level 1, then **IRQ[0]** has precedence over **IRQ[1]**.

The processor starts to execute from the address given in the exception's vector table entry. An exception handler is pre-empted if an exception occurs that has a higher priority. If an exception occurs with the same priority as the exception handler, the handler is not pre-empted, irrespective of the exception number.

For information on system exception prioritizing, see *System handler priority registers* on page 6-8.

2.6.3 Exception entry

An exception causes the corresponding exception to become pending. When there is a pending exception of higher priority than the priority the processor is currently executing, exception entry occurs. During exception entry the corresponding exception becomes unpended unless another higher priority exception occurs, and is entered instead.

When the processor takes an exception, it automatically pushes the following eight registers to the current stack:

- xPSR
- `ReturnAddress()`
- LR
- R12
- R3
- R2
- R1
- R0

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

The SP is decremented by eight words on the completion of the stack push. Doubleword alignment of the stack pointer is enforced when stacking commences. Bit[2] of the stack pointer is saved as Bit[9] of the stacked xPSR.

An EXC_RETURN value is written to the LR indicating which stack pointer was used to stack the context and whether the processor was in Thread mode or Handler mode before the entry occurred.

The processor then enters handler mode and switches to the main stack pointer for subsequent stacking operations.

2.6.4 Exception exit

Exception exit occurs when one of the following instructions executed in Handler mode loads a EXC_RETURN value that was written to the LR on entry to the handler into the PC:

- POP that includes loading the PC
- BX with any register.

Table 2-7 lists the `EXC_RETURN[3:0]` values with a description of the exception return behavior.

The EXC_RETURN value has bits [31:24] set to `0xF` to indicate to the processor that the exception is complete, and the processor initiates the exception exit sequence.

The processor selects the main or process stack pointer according to bits [3:0] of the EXC_RETURN value, see Table 2-7, and if indicated by these bits, switches to Thread mode. The processor pops the eight registers from this stack. The stack pointer is adjusted according to bit [9] of the popped xPSR value to match the alignment before the exception handler was entered.

The interrupt return value, EXC_RETURN, is passed as a data field in the LR. This means exception functions can be C/C++ functions, and do not require a wrapper written in assembler.

Table 2-7 Exception return behavior

EXC_RETURN[3:0]	Description
0bXXX0	Reserved.
0b0001	Return to Handler mode. Exception return gets state from the main stack. Execution uses SP_Main after return.
0b0011	Reserved.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 2-7 Exception return behavior (continued)

EXC_RETURN[3:0]	Description
0b01X1	Reserved.
0b1001	Return to Thread mode. Exception return gets state from the main stack. Execution uses SP_Main after return.
0b1101	Return to Thread mode. Exception return gets state from the process stack. Execution uses SP_Process after return.
0b1X11	Reserved.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.7 LOCKUP

The processor has a LOCKUP state that is entered when a HardFault occurs while executing the NMI or HardFault handlers. While the processor is in LOCKUP no more instruction execution occurs. The processor remains in the LOCKUP state until:

- it is reset
- halting debug mode is entered
- an NMI occurs.

Note

If LOCKUP state occurs from the NMI handler a subsequent NMI does not cause the processor to leave LOCKUP state.

A debugger can identify debug state entry from LOCKUP when the PC reads as `0xFFFFFFFE` and the IPSR reads as `0x2` or `0x3`.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

2.8 Memory model

This section describes the memory model of the processor that includes:

- memory map
- memory types
- transaction ordering.

2.8.1 Processor memory map

Figure 2-5 shows the fixed memory map of the processor.

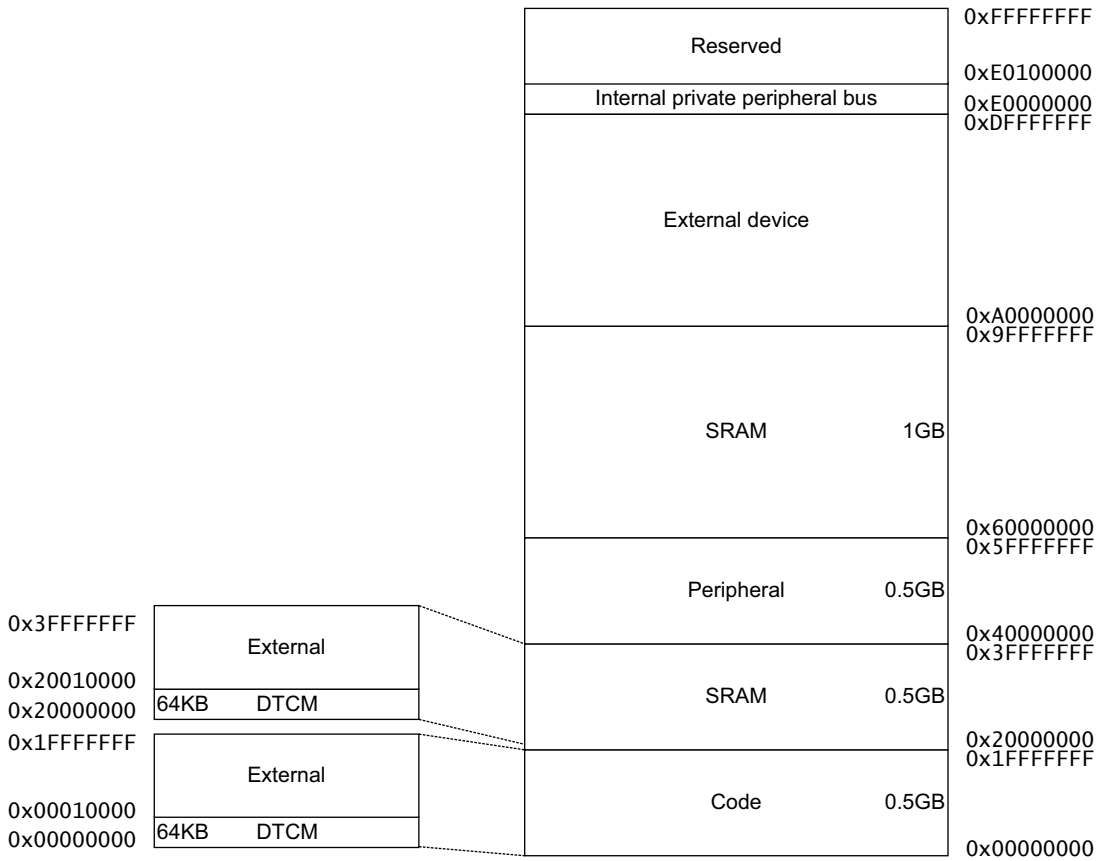

Figure 2-5 Processor memory map

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 2-8 lists processor interfaces that are used when different memory map regions are addressed by both or either instruction and data accesses.

Table 2-8 Processor memory regions

Address range	Memory region	Memory type	Interface
0x00000000-0x1FFFFFFF	Code	Normal	Instruction and data accesses to this memory region appear on the ITCM or the external Avalon-MM interface depending on: • the address of the access • the size of the configured ITCM. The ITCM starts at 0x00000000 and addresses up to the size of the ITCM appear on the internal ITCM interface only. Accesses beyond the ITCM size appear on the external Avalon-MM interface
0x20000000-0x3FFFFFFF	SRAM	Normal	Data accesses to this memory region appear on the DTCM or the external Avalon-MM interface depending on: • the address of the access • the size of the configured DTCM. The DTCM starts at 0x20000000 and data accesses to addresses up to the size of the DTCM appear on the internal DTCM interface only. Data accesses beyond the DTCM size appear on the external Avalon-MM interface. The space occupied by the DTCM is marked as *Execute Never* (XN). If the processor attempts to execute code from this region, the instruction fetches are prevented and a HardFault exception is generated. Instruction fetches are permitted to the area resolving to external memory.
0x40000000-0x5FFFFFFF	Peripheral	Device	Data accesses to this region appear on the external Avalon-MM interface. Instruction accesses are prevented by the processor hardware and cause a HardFault exception.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 2-8 Processor memory regions (continued)

Address range	Memory region	Memory type	Interface
0x60000000- 0x9FFFFFFF	SRAM	Normal	Instruction and data accesses to this region appear on the external Avalon-MM interface.
0xA0000000- 0xDFFFFFFF	External device	Device	Data accesses to this region appear on the external Avalon-MM interface. Instruction accesses are prevented by the processor hardware and cause a HardFault exception.
0xE0000000- 0xE00FFFFF	Private Peripheral Bus	Strongly Ordered	Accesses to this region are to the processor's internal *Private Peripheral Bus* (PPB) containing the NVIC, and when implemented, optional debug support. This region is marked as Execute Never (XN). If the processor attempts to execute code from this region, the instruction fetches are prevented and a HardFault exception is generated. The PPB address space only supports word (aligned) accesses.

When designing the SoC, place the peripherals in a region of the address space with an appropriate memory type to ensure the processor accesses memory-mapped peripheral registers in the required order.

2.8.2 Memory types and memory transactions

This section describes the memory transactions associated with different types of memory.

The order that the instructions are executed by the processor does not guarantee that the corresponding memory transactions occur in the same order.

The processor can re-order some memory transactions to improve efficiency, providing the behavior of the instruction sequence is unaffected.

The memory types are:

Normal Also referred to as weakly ordered, the processor can speculatively read or re-order transactions for efficiency.

Strongly Ordered The processor must preserve transaction order relative to all other transactions.

Device The processor must preserve transaction order relative to other transactions to device memory.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

You must enforce memory transaction ordering:

- in a system where the re-ordering is observable by other system components, for example, other processors or DMA controllers.
- where a memory transaction must be completed to ensure any indirect effect is recognizable by subsequent instructions. For example, when changing the priority of an interrupt or modifying instructions in memory that are to be executed.

The processor uses barrier instructions to enforce memory transaction and instruction ordering. The barrier instructions are:

`DMB` The *Data Memory Barrier* (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.

`DSB` The *Data Synchronization Barrier* (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.

`ISB` The *Instruction Synchronization Barrier* (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

Note

To ensure that a store to a PPB register has the appropriate effect on a particular instruction, you must ensure that the instruction is preceded by a `DSB`, `ISB` instruction sequence.

For examples of how to use barrier instructions to enforce memory transaction and instruction ordering, see *Using memory barrier instructions for memory transactions* on page A-4.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 3
Processor Instantiation and Configuration

This chapter describes how to instantiate and configure the Cortex-M1 processor for use in your own system design. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.1 About processor instantiation and configuration

The processor is available for instantiation and configuration within Altera SOPC Builder. The SOPC Builder configuration window is used to select the instantiation and configuration options required for the processor.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.2 Processor instantiation

Figure 3-1 shows the SOPC Builder project window.

To instantiate the Cortex-M1 processor from the SOPC Builder project window:

1. Select the **System Contents** tab to display the SOPC Builder component library.
2. Select the entry **ARM Cortex-M1 Processor** in the SOPC Builder component library.
3. Click on the **Add...** button underneath the SOPC Builder component library.

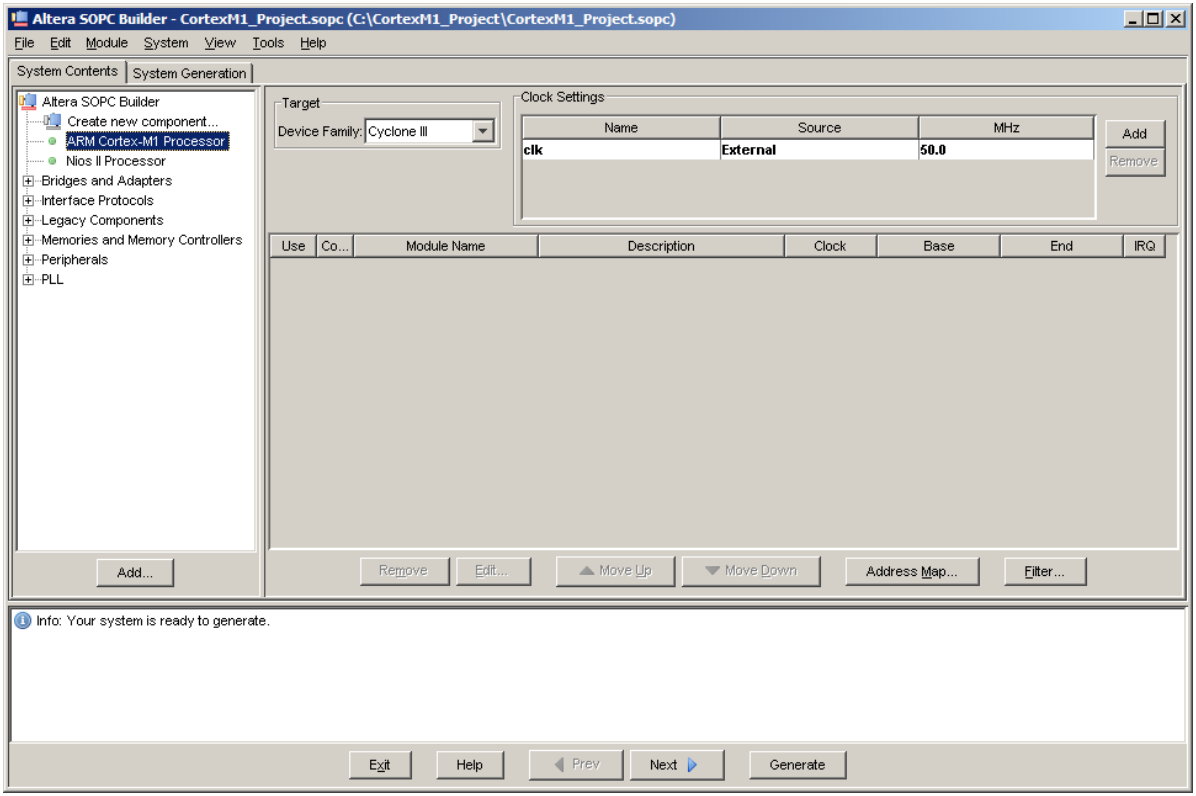

Figure 3-1 SOPC Builder project window

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.3 Processor configuration

Figure 3-2 shows the configuration dialog box that appears when a Cortex-M1 processor is instantiated. You can use this dialog box to adjust the configurable parameters of the processor.

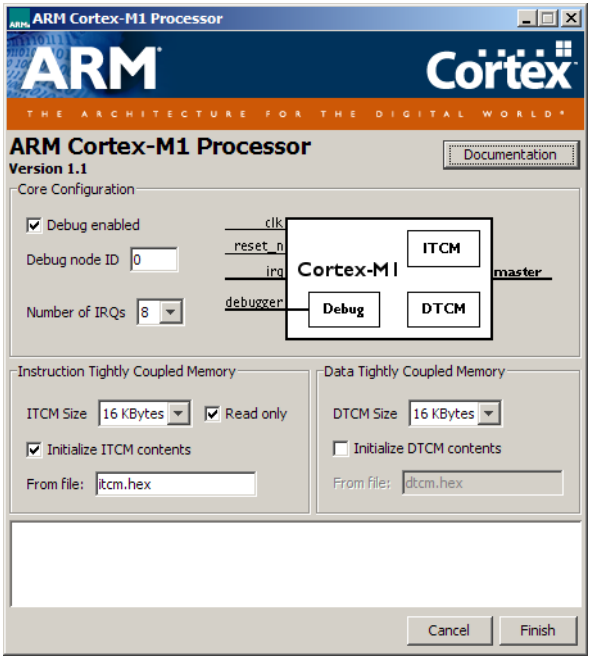

Figure 3-2 Configuration dialog box

Table 3-1 on page 3-5 lists the supported configuration options and the supported values.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 3-1 Processor configuration

Option	Description	Default
Core configuration		
Debug enabled	Tickbox to enable debug logic in the processor	Set
Debug node ID	A unique number to identify a debug core in a multiprocessor system. A unique number between 0 and 255 must be assigned to each Cortex-M1 debug processor in the SOPC Builder design. This option is only available when the Debug enabled option is set.	0
Number of IRQs	Number of interrupts available on the SOPC Builder component interface: 1, 8, or 32.	8
Instruction Tightly Coupled Memory		
ITCM size	Size of ITCM. The supported sizes are: 0KB (No TCM) 1KB 2KB 4KB 8KB 16KB 32KB 64KB.	16KB
Read only	The address locations comprising the ITCM are read only. For example, setting the ITCM to read-only can prevent the processor from modifying code inadvertently. The debugger access to the ITCM is always Read/Write regardless of the setting of this configuration. This option is not available if the ITCM size is set to No ITCM.	Set
Initialize ITCM contents	Tickbox to enable ITCM initialization from the file. This option is not available if the ITCM size is set to No ITCM.	Set
From file	Initialization file specifies the initial contents of the ITCM. This option is not available if the ITCM size is set to No ITCM.	`itcm.hex`

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 3-1 Processor configuration (continued)

Option	Description	Default
Data Tightly Coupled Memory		
DTCM size	Size of DTCM. The supported sizes are: 0KB (No TCM) 1KB 2KB 4KB 8KB 16KB 32KB 64KB.	16KB
Data TCM initialization	Tickbox to enable DTCM initialization from the file. This option is not available if the DTCM size is set to No DTCM.	Not set
From file	Initialization file specifies the initial contents of the DTCM. This option is not available if the DTCM size is set to No DTCM.	`dtcm.hex`

Note

If TCMs are included, these can be initialized using an Intel HEX format image. By default, the ITCM is initialized using the file `itcm.hex` and the DTCM is initialized using the file `dtcm.hex`. These files must be in the hexadecimal format for memory initialization required by Quartus II software. For more information, see the Quartus II documentation.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.4 Configuration considerations

ARM recommends that you consider device utilization in your design because the processor TCMs utilize on-chip memory resources.

To determine how much on-chip memory resource is available, and how much is required for other components in your system, see the datasheets that apply to your target device family.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.5 System connectivity

This section contains the following sections:

- *Avalon interfaces*
- *Exported signals*.

3.5.1 Avalon interfaces

When you have instantiated the Cortex-M1 processor within SOPC Builder, the Avalon-MM master and Interrupts interfaces are available for connection to peripherals and memories in the system.

Avalon-MM master interface

Do not place Avalon peripherals or external memory in the TCM address regions because TCM accesses are internal to the processor.

Figure 2-5 on page 2-19 shows the recommended region of the processor memory map where Avalon peripherals can be placed.

Interrupts interface

The processor uses a software prioritization scheme for interrupts. This means the Avalon System Interconnect Fabric assigns no priority to the interrupts from different sources.

You must ensure that software running on the processor assigns priorities to different interrupt lines by programming the NVIC. This is because interrupts that come from the Avalon fabric have no priority levels associated with them.

The *Altera Coporation, Avalon Memory-Mapped Interface Specification* defines interrupts to be level sensitive. The Cortex-M1 NVIC supports both level and pulse interrupts, so care must be taken that peripherals in the system do not pulse the interrupt pin unless an interrupt is requested.

3.5.2 Exported signals

Exported signals are signals that you must connect manually to the outside of the SOPC Builder system. Several signals are exported through the SOPC Builder system that must be connected to the top level of your design. Table 3-2 on page 3-9 lists the signals exported from the processor.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

All signals, except **DBGRESETn**, are sampled or generated synchronously with respect to the main processor clock. You must ensure that any logic connected to these signals is connected to the same clock as the processor. For example, if a PLL is used within the system to generate the processor clock, the exported version of this clock must be used to clock the external logic.

Table 3-2 Signals exported from the processor

Name	Direction	Description
DBGRESETn[a]	Input	Active LOW debug reset. This causes a reset of the debug logic without resetting the processor. This signal is present but not connected internally for the non-debug configuration.
EDBGRQ[a]	Input	External debug request. Indicates that external logic is signaling a debug event. This signal is present but not connected internally for the non-debug configuration. Ensure that this signal is tied LOW if you do not use the debug configuration.
HALTED[a]	Output	Asserted when the core is in halting debug state. This signal is always LOW for a non-debug configuration.
LOCKUP	Output	When HIGH indicates that the core is in the LOCKUP state. This signal can be connected to a watchdog that resets the system, for example.
NMI	Input	Non-maskable interrupt.

a. This signal is present in both debug and non-debug configurations of the component to avoid having to change the system when you switch between debug and non-debug configurations.

3.5.3 Resets and clocks

The processor has a clock and a reset input that are connected automatically by SOPC Builder to a clock and an appropriate reset signal within the system. The signals that comprise the Avalon-MM bus master interface, the interrupt interface, and the exported signals are all synchronous relative to this clock. All of the logic within the processor except the debug logic is reset by this reset signal.

The **DBGRESETn** signal appears as an input to the SOPC system and must be driven LOW after power is applied to the system to reset the debug logic. You can connect this input to the same source as the main **reset_n** input into the SOPC system.

Note

When **DBGRESETn** is driven LOW, all logic within the processor is reset so it must not be driven LOW without also driving **reset_n** low to reset the rest of the system at the same time.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

3.6 Simulation considerations

SOPC Builder can automatically generate a testbench to simulate SOPC designs in the ModelSim-Altera software. The Cortex-M1 processor uses several exported signals, as described in *Exported signals* on page 3-8. These signals are not automatically connected in the SOPC Builder testbench.

During simulation of the Cortex-M1 processor in the SOPC Builder testbench the exported inputs are set to default values, see Table 3-3. If you want to drive these signals from other custom logic, you can add custom code to the generated testbench. Driving the exported input signals from custom logic overrides the default values in Table 3-3.

Table 3-3 shows the default exported input signal values in the SOPC Builder testbench.

Table 3-3 Default exported input signals

Signal	Default value
DBGRESETn	1'b1
EDBGRQ	1'b0
NMI	1'b0

The Cortex-M1 processor also uses exported output signals. These signals are not connected to any external logic in the default testbench, and can be used to drive any custom logic that is added to the testbench.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 4
System Timer

This section describes the system timer and the registers it uses. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

4.1 About the system timer

The processor has a 24-bit system timer that the NVIC controller can use. This means that an RTOS does not have to rely on external timers.

The system timer consists of a control and status register to:

- configure its clock
- enable the counter
- enable the SysTick exception
- determine counter status.

When the system timer is enabled, it counts down from the reload value to zero, reload (wrap) to the value in the SysTick Reload Value Register on the next clock edge, then decrements on subsequent clocks.

Note

- If the core is in Debug state the counter does not decrement.
- The timer is clocked by the core clock.

Table 4-1 shows a summary of the system timer and control registers.

Table 4-1 System timer registers

Name of register	Type	Address	Reset value	See
SysTick Control and Status Register	R/W	0xE000E010	0x00000004	page 4-3
SysTick Reload Value Register	R/W	0xE000E014	0x00000000	page 4-4
SysTick Current Value Register	R/W clear	0xE000E018	0x00000000	page 4-5
SysTick Calibration Value Register	RO	0xE000E01C	0x80000000	page 4-6

Note

All registers are only accessible using word sized transfers. Attempting to write a halfword or byte can cause corruption of the register bits.

For examples of how to use the system timer, see *Examples of using the system timer* on page A-5.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

4.2 System timer registers

The sections that follow describe the system timer registers.

4.2.1 SysTick Control and Status Register

Use the SysTick Control and Status Register to enable the SysTick features.

The register address, access type, and reset value are:

Address `0xE000E010`

Access Read/write

Reset value `0x00000004`

Figure 4-1 shows the bit assignments of the SysTick Control and Status Register.

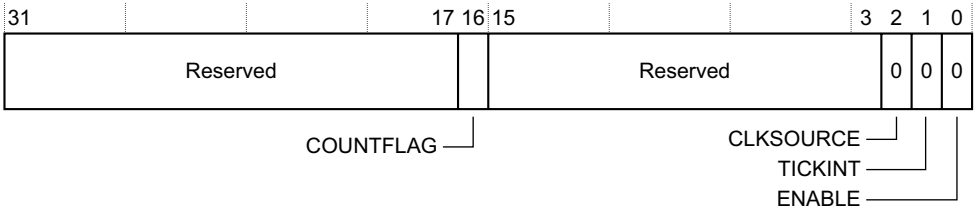

Figure 4-1 SysTick Control and Status Register bit assignments

Table 4-2 lists the bit assignments of the SysTick Control and Status Register.

Table 4-2 SysTick Control and Status Register bit assignments

Bits	Field	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read. Clears on read by application or debugger.
[15:3]	-	Reserved.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 4-2 SysTick Control and Status Register bit assignments (continued)

Bits	Field	Function
[2]	CLKSOURCE	Always reads as one: 1 = core clock. Indicates that SysTick uses the processor clock.
[1]	TICKINT	1 = counting down to zero pends the SysTick handler. 0 = counting down to zero does not pend the SysTick handler. Software can use COUNTFLAG to determine if the SysTick handler has ever counted to zero.
[0]	ENABLE	1 = counter operates in a multi-shot way. That is, counter loads with the Reload value and then begins counting down. On reaching 0, it sets the COUNTFLAG to 1 and optionally pends the SysTick handler, based on TICKINT. It then loads the Reload value again, and begins counting. 0 = counter disabled.

4.2.2 SysTick Reload Value Register

Use the SysTick Reload Value Register to specify the start value to load into the SysTick Current Value Register when the counter reaches 0. It can be any value in range `0x00000001`-`0x00FFFFFF`. A start value of 0 is possible, but has no effect because the SysTick interrupt and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value can be calculated according to its use. For example:

- A multi-shot timer has a SysTick interrupt RELOAD of N-1 to generate a timer period of N processor clock cycles. For example, if the SysTick interrupt is required every 100 clock pulses, 99 must be written into RELOAD.
- A single shot timer has a SysTick interrupt RELOAD of N to deliver a single SysTick interrupt after a delay of N processor clock cycles. For example, if a SysTick interrupt is next required after 400 clock pulses, you must write 400 into RELOAD.

The register address, access type, and reset value are:

Address `0xE000E014`

Access Read/write

Reset value `0x00000000`

Figure 4-2 on page 4-5 shows the bit assignments of the SysTick Reload Value Register.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

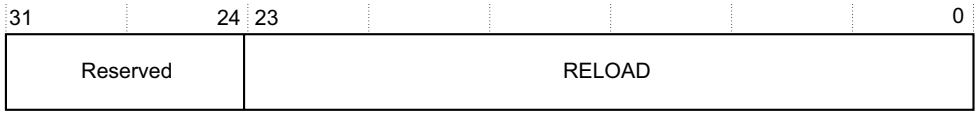

Figure 4-2 SysTick Reload Value Register bit assignments

Table 4-3 lists the bit assignments of the SysTick Reload Value Register.

Table 4-3 SysTick Reload Value Register bit assignments

Bits	Field	Function
[31:24]	-	Reserved
[23:0]	RELOAD	Value to load into the SysTick Current Value Register when the counter reaches 0

4.2.3 SysTick Current Value Register

Use the SysTick Current Value Register to find the current value of the SysTick counter.

The register address, access type, and reset value are:

Address `0xE000E018`
Access Read/write clear
Reset value `0x00000000`

Figure 4-3 shows the bit assignments of the SysTick Current Value Register.

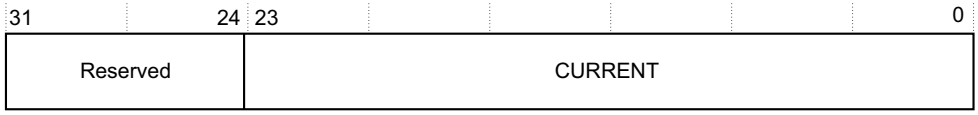

Figure 4-3 SysTick Current Value Register bit assignments

Table 4-4 lists the bit assignments of the SysTick Current Value Register.

Table 4-4 SysTick Current Value Register bit assignments

Bits	Field	Function
[31:24]	-	Reserved.
[23:0]	CURRENT	Reads return the current value of the SysTick counter. This register is write-clear. Writing to it with any value clears the register to 0. Clearing this register also clears the COUNTFLAG bit of the SysTick Control and Status Register.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

4.2.4 SysTick Calibration Value Register

The SysTick Calibration Value Register is used to indicate, if known, the information required to calibrate the system timer to a frequency of 100Hz. For this processor, the calibration information is not known. Instead, you must calculate the calibration value required from the frequency of the processor clock.

The register address, access type, and reset value are:

Address `0xE000E01C`

Access Read

Reset value `0x80000000`

Figure 4-4 shows the bit assignments of the SysTick Calibration Value Register.

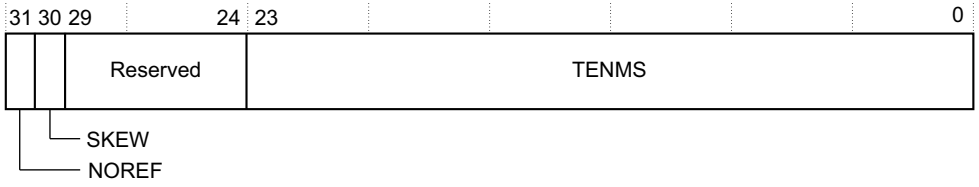

Figure 4-4 SysTick Calibration Value Register bit assignments

Table 4-5 lists the bit assignments of the SysTick Calibration Value Register.

Table 4-5 SysTick Calibration Value Register bit assignments

Bits	Field	Function
[31]	NOREF	Reads as one. Indicates that no separate reference clock is provided.
[30]	SKEW	Reads as zero. Calibration value for the 10ms inexact timing is not known because TENMS is not known. This can affect its suitability as a software real time clock.
[29:24]	-	Reserved.
[23:0]	TENMS	Reads as zero. Indicates calibration value is not known.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 5
Nested Vectored Interrupt Controller

This section describes the *Nested Vectored Interrupt Controller* (NVIC) and the registers it uses. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

5.1 About the NVIC

The core contains a tightly coupled NVIC that supports 1, 8 or 32 interrupts with four levels of priority. The processor state is automatically stacked on exception entry and unstacked on exception exit with no instruction overhead, which enables low latency of exception handling.

Table 5-1 shows a summary of the NVIC registers.

Table 5-1 NVIC registers

Name of register	Type	Address	Reset value	See
Interrupt Set Enable Register	R/W	0XE000E100	0x00000000	page 5-3
Interrupt Clear Enable Register	R/W	0XE000E180	0x00000000	page 5-4
Interrupt Set Pending Register	R/W	0XE000E200	0x00000000	page 5-5
Interrupt Clear Pending Register	R/W	0XE000E280	0x00000000	page 5-6
Priority 0 Register	R/W	0XE000E400	0x00000000	page 5-7
Priority 1 Register	R/W	0XE000E404		
Priority 2 Register	R/W	0XE000E408		
Priority 3 Register	R/W	0XE000E40C		
Priority 4 Register	R/W	0XE000E410		
Priority 5 Register	R/W	0XE000E414		
Priority 6 Register	R/W	0XE000E418		
Priority 7 Register	R/W	0XE000E41C		

Note

All registers are only accessible using word sized transfers. Attempting to write a halfword or byte can cause corruption of the register bits.

For an example of how to configure an external interrupt, see *Example of how to configure and enable an external interrupt* on page A-8.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

5.2 NVIC register descriptions

The sections that follow describe the NVIC registers.

5.2.1 Interrupt Set-Enable Register

Use the Interrupt Set-Enable Register to:

- enable interrupts
- determine which interrupts are currently enabled.

Each bit in the register corresponds to one of 32 interrupts. Setting a bit in the Interrupt Set-Enable Register enables the corresponding interrupt.

When the enable bit of a pending interrupt is set, the processor activates the interrupt based on its priority. When the enable bit is clear, asserting its interrupt signal pends the interrupt, but it is not possible to activate the interrupt, regardless of its priority.

Clear the enable state by writing a `0x1` to the corresponding bit in the Interrupt Clear-Enable Register (see *Interrupt Clear-Enable Register* on page 5-4). This also clears the corresponding bit in the Interrupt Set-Enable Register.

The register address, access type, and reset value are:

Address `0xE000E100`

Access Read/write

Reset value `0x00000000`

Figure 5-1 shows the bit assignments of the Interrupt Set-Enable Register.

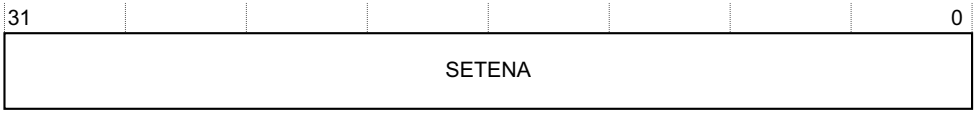

Figure 5-1 Interrupt Set-Enable Register bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 5-2 lists the bit assignments of the Interrupt Set-Enable Register.

Table 5-2 Interrupt Set-Enable Register bit assignments

Bits	Field	Function
[31:0]	SETENA	Interrupt set enable bits. For writes: 1 = enable interrupt 0 = no effect. For reads: 1 = interrupt enabled 0 = interrupt disabled Writing 0 to a SETENA bit has no effect. Reading the bit returns its current enable state. Reset clears the SETENA fields.

5.2.2 Interrupt Clear-Enable Register

Use the Interrupt Clear-Enable Register to:

- disable interrupts
- determine which interrupts are currently enabled.

Each bit in the register corresponds to one of 32 interrupts. Setting an Interrupt Clear-Enable Register bit disables the corresponding interrupt.

Note

Writing 1 to an Interrupt Clear-Enable Register bit does not affect currently active interrupts. It only prevents new activations.

The register address, access type, and reset value are:

Address `0xE000E180`

Access Read/write

Reset value `0x00000000`

Figure 5-2 shows the bit assignments of the Interrupt Clear-Enable Register.

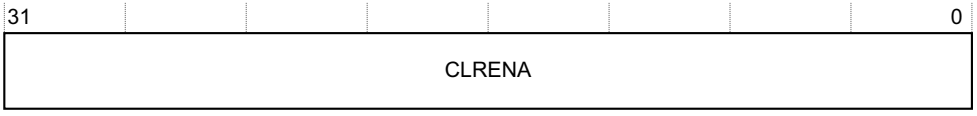

Figure 5-2 Interrupt Clear-Enable Register bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 5-3 lists the bit assignments of the Interrupt Clear-Enable Register.

Table 5-3 Interrupt Clear-Enable Register bit assignments

Bits	Field	Function
[31:0]	CLRENA	Interrupt clear-enable bits. For writes: 1 = disable interrupt 0 = no effect. For reads: 1 = interrupt enabled 0 = interrupt disabled. Writing 0 to a CLRENA bit has no effect. Reading the bit returns its current enable state. Reset clears the CLRENA field.

5.2.3 Interrupt Set-Pending Register

Use the Interrupt Set-Pending Register to:

- force interrupts into the pending state
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Set-Pending Register bit pends the corresponding interrupt. Writing a `0x0` to a pending bit has no effect on the pending state of the corresponding interrupt.

Clear an interrupt pending bit by writing a `0x1` to the corresponding bit in the Interrupt Clear-Pending Register, see *Interrupt Clear-Pending Register* on page 5-6.

Note

Writing to the Interrupt Set-Pending Register has no effect on an interrupt that is already pending.

The register address, access type, and reset value are:

Address `0xE000E200`
Access Read/write
Reset value `0x00000000`

Figure 5-3 on page 5-6 shows the bit assignments of the Interrupt Set-Pending Register.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

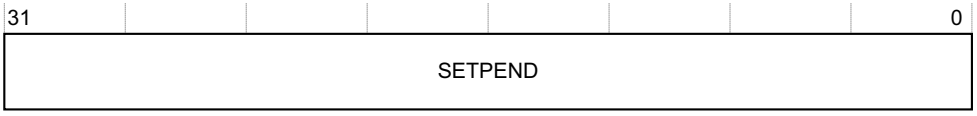

Figure 5-3 Interrupt Set-Pending Register bit assignments

Table 5-4 lists the bit assignments of the Interrupt Set-Pending Register.

Table 5-4 Interrupt Set-Pending Register bit assignments

Bits	Field	Function
[31:0]	SETPEND	Interrupt set-pending bits. For writes: 1 = pend interrupt 0 = no effect. For reads: 1 = interrupt is pending 0 = interrupt is not pending.

5.2.4 Interrupt Clear-Pending Register

Use the Interrupt Clear-Pending Register to:

- clear pending interrupts
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Clear-Pending Register bit clears the pending state of the corresponding interrupt.

Note

Writing to the Interrupt Clear-Pending Register has no effect on an interrupt that is active unless it is also pending.

The register address, access type, and reset value are:

Address `0xE000E280`

Access Read/write

Reset value `0x00000000`

Figure 5-4 on page 5-7 shows the bit assignments of the Interrupt Clear-Pending Register.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

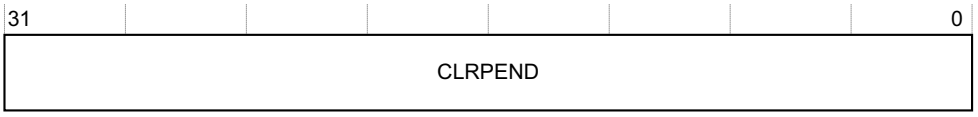

Figure 5-4 Interrupt Clear-Pending Register bit assignments

Table 5-5 lists the bit assignments of the Interrupt Clear-Pending Registers.

Table 5-5 Interrupt Clear-Pending Registers bit assignments

Bits	Field	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. For writes: 1 = clear pending interrupt 0 = no effect. For reads: 1 = interrupt is pending 0 = interrupt is not pending.

5.2.5 Interrupt Priority Registers

Use the Interrupt Priority Registers to assign a priority from 0 to 3 to each of the available interrupts. 0 is the highest priority, and 3 is the lowest.

The two bits of priority are stored in bits [7:6] of each byte.

The register address, access type, and reset value are:

Address `0xE000E400-0xE000E41C`

Access Read/write

Reset value `0x00000000`

Figure 5-5 on page 5-8 shows the bit assignments of Interrupt Priority Registers 0-7.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

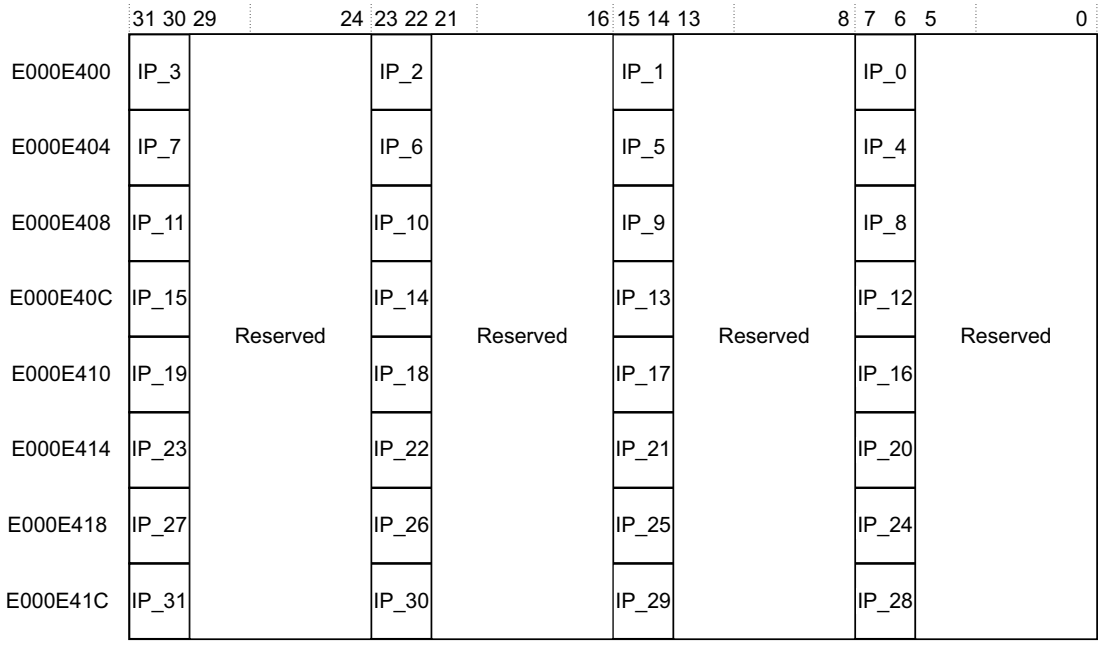

Figure 5-5 Interrupt Priority Registers 0-7 bit assignments

Figure 5-5 shows fields for 32 interrupts using Interrupt Priority Registers 0-7. If your implementation uses fewer interrupts, all unused registers are Reserved.

Table 5-6 lists the bit assignments of the Interrupt Priority Registers.

Table 5-6 Interrupt Priority Registers 0-31 bit assignments

Bits	Field	Function
[7:6]	IP_*n*	Priority of interrupt *n*

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

5.3 Level versus pulse interrupts

The processor supports both level and pulse interrupts. A level interrupt is held asserted until it is cleared by the ISR accessing the device. A pulse interrupt is a variant of an edge model. The interrupt signal is sampled synchronously on the rising edge of the processor clock. The processor recognizes a pulse when the input is observed LOW and then HIGH on two consecutive rising edges of the processor clock.

For level interrupts, if the signal is not deasserted before the return from the interrupt routine, the interrupt repends and re-activates. This is particularly useful for FIFO and buffer-based devices because it ensures that they drain either by a single ISR or by repeated invocations, with no extra work. This means that the device holds the interrupt signal asserted until the device is empty.

A pulse interrupt must be asserted for at least one processor clock cycle to enable the NVIC to observe it.

A pulse interrupt can be reasserted during the ISR so that the interrupt can be pended and active at the same time. The application design must ensure that a second pulse does not arrive before the interrupt caused by the first pulse is activated. If the second pulse arrives before the interrupt is activated, the second pulse has no effect because it is already pended. When the ISR is activated, the pend bit is cleared. If the interrupt asserts again when the ISR is activated, the NVIC latches the pend bit again.

Pulse interrupts are mainly used for external signals and for rate or repeat signals.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

5.4 Resampling level interrupts

An ISR can detect that no more interrupts occurred during interrupt processing to avoid the overhead of ISR exit and entry. This information is available in the set and clear pending registers, see *Interrupt Set-Pending Register* on page 5-5 and *Interrupt Clear-Pending Register* on page 5-6.

For pulse interrupts, a bit that is set to 1 indicates that another interrupt has arrived since the ISR started.

If the level interrupt is guaranteed to have been cleared and then asserted, the status bit read from the Interrupt Pending Registers is set to 1, as for pulse interrupts.

For level interrupts, where the line might remain HIGH continuously from ISR entry, write 1 to the appropriate bit of the:

- Interrupt Set-Pending Register
- Interrupt Clear-Pending Register.

The Interrupt Clear-Pending Register is not cleared if the interrupt line is HIGH, and can be read again to determine the status.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Chapter 6
System Control Block

This section describes the system control block and the registers it uses. It contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

6.1 About the system control block

The system control block provides:

- Exception enables.
- Setting or clearing exceptions to/from the pending state.
- Exception status (Inactive, Pending, or Active). Inactive is when an exception is neither Pending nor Active.
- Priority setting (for configurable system exceptions)

It also provides the exception number of the currently executing code and highest pending exception.

Table 6-1 gives a summary of the system configuration and control registers implemented in this processor.

Table 6-1 Summary of the system control registers

Name of register	Type	Address	Reset value	See
CPUID Base Register	RO	0xE000ED00	0x410CC210	page 6-3
Interrupt Control State Register	-[a]	0xE000ED04	0x00000000	page 6-4
Application Interrupt and Reset Control Register	-[b]	0xE000ED0C	0xFA050000	page 6-6
Configuration and Control Register	R/W	0xE000ED14	0x00000208	page 6-7
System Handler Priority Register 2	R/W	0xE000ED1C	0x00000000	page 6-8
System Handler Priority Register 3	R/W	0xE000ED20	0x00000000	page 6-8
System Handler Control and State Register	R/W	0xE000ED24	0x00000000	page 6-10

a. Access type depends on the individual bit. For more information see Table 6-3 on page 6-5
b. Access type depends on the individual bit. For more information see Table 6-4 on page 6-7

Note

All registers are only accessible using word sized transfers. Attempting to write a halfword or byte can cause corruption of the register bits.

For an example of how to configure and generate a PendSV, see *Example of how to configure and schedule a PendSV exception* on page A-9.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

6.2 System control block registers

The sections that follow describe the system control block registers.

6.2.1 CPUID Base Register

Read the CPUID Base Register to determine:

- the ID number of the processor core
- the version number of the processor core
- the implementation details of the processor core.

The register address, access type, and reset value are:

Address `0xE000ED00`
Access Read-only
Reset value `0x410CC210`

Figure 6-1 shows the bit assignments of the CPUID Base Register.

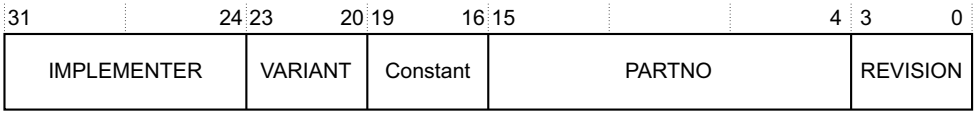

Figure 6-1 CPUID Base Register bit assignments

Table 6-2 lists the bit assignments of the CPUID Base Register.

Table 6-2 CPUID Base Register bit assignments

Bits	Field	Function
[31:24]	IMPLEMENTER	Implementer code: `0x41` = ARM
[23:20]	VARIANT	Implementation defined variant number: `0x0` for r0p1
[19:16]	Constant	Reads as `0xC`
[15:4]	PARTNO	Number of processor within family: `0xC21`
[3:0]	REVISION	Implementation defined revision number: `0x1` = r0p1

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

6.2.2 Interrupt Control State Register

Use the Interrupt Control State Register to:

- set a pending *Non-Maskable Interrupt* (NMI)
- set or clear a pending PendSV
- set or clear a pending SysTick
- check for pending exceptions
- check the vector number of the highest priority pended exception
- check the vector number of the active exception.

The register address, access type, and reset value are:

Address `0xE000ED04`

Access Access type depends on the individual bit. For more information see Table 6-3 on page 6-5.

Reset value `0x00000000`

Figure 6-2 shows the bit assignments of the Interrupt Control State Register.

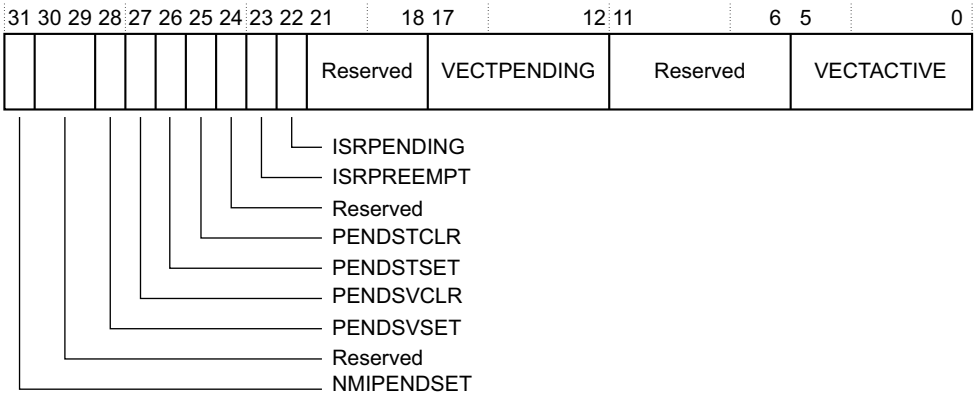

Figure 6-2 Interrupt Control State Register bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 6-3 lists the bit assignments of the Interrupt Control State Register.

Table 6-3 Interrupt Control State Register bit assignments

Bits	Field	Type	Function
[31]	NMIPENDSET	R/W	On writes: 1 = set pending NMI 0 = no effect. NMIPENDSET pends and activates an NMI. Because NMI is the highest-priority interrupt, it takes effect as soon as it registers unless the processor is locked at a priority of -2. On reads, this bit returns the pending state of NMI.
[30:29]	-	-	Reserved.
[28]	PENDSVSET	R/W	On writes: 1 = set pending PendSV 0 = no effect. On reads this bit returns the pending state of PendSV.
[27]	PENDSVCLR	WO	On writes: 1 = clear pending PendSV 0 = no effect.
[26]	PENDSTSET	R/W	On writes: 1 = set pending SysTick 0 = no effect. On reads this bit returns the pending state of SysTick.
[25]	PENDSTCLR	WO	On writes: 1 = clear pending SysTick 0 = no effect.
[24]	-	-	Reserved.
[23]	ISRPREEMPT[a]	RO	The system can only access this bit when the core is halted. It indicates that a pending interrupt is to be taken in the next running cycle: 1 = a pending exception is serviced on exit from the debug halt state 0 = a pending exception is not serviced.
[22]	ISRPENDING[a]	RO	External interrupt pending flag, where: 1 = interrupt pending 0 = interrupt not pending.
[21:18]	-	-	Reserved.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 6-3 Interrupt Control State Register bit assignments (continued)

Bits	Field	Type	Function
[17:12]	VECTPENDING	RO	Indicates the exception number for the highest priority pending enabled exception: 0 = no pending exceptions Non zero = An enabled exception is pending. This field includes the effect of memory-mapped enable and mask registers. It does not include the PRIMASK special-purpose register qualifier.
[11:6]	-	-	Reserved.
[5:0]	VECTACTIVE[a]	RO	Active exception number field: 0 = Thread mode Non zero = the exception number[b] of the currently active exception. Reset clears the VECTACTIVE field.

a. Debug Extension only, otherwise Reserved.
b. This is the same value as IPSR bits [5:0].

6.2.3 Application Interrupt and Reset Control Register

Use the Application Interrupt and Reset Control Register to:

- determine data endianness
- clear all active state information for debug or to recover from a hard failure
- request a system reset.

The register address, access type, and reset value are:

Address `0xE000ED0C`

Access Access type depends on the individual bit. For more information see Table 6-4 on page 6-7

Reset value `0xFA050000`

Figure 6-3 shows the bit assignments of the Application Interrupt and Reset Control Register.

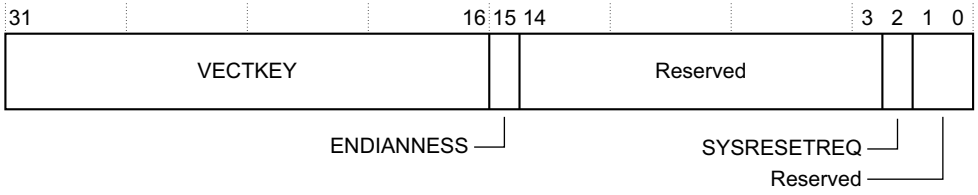

Figure 6-3 Application Interrupt and Reset Control Register bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 6-4 lists the bit assignments of the Application Interrupt and Reset Control Register.

Table 6-4 Application Interrupt and Reset Control Register bit assignments

Bits	Field	Type	Function
[31:16]	VECTKEY	WO	Register key. To write to other parts of this register, you must ensure `0x5FA` is written into the VECTKEY field.
[15]	ENDIANNESS	RO	Data endianness bit. Always reads as zero. The endian configuration implemented is little-endian.
[14:3]	-	-	Reserved.
[2]	SYSRESETREQ	WO	Writing 1 to this bit causes a signal to the outer system to be asserted to request a reset. Intended to force a system reset of all major components except for debug. The debugger does not lose contact with the device.
[1]	VECTCLRACTIVE	WO	Clears all active state information for fixed and configurable exceptions. This bit: • is self-clearing • can only be set by a debugger when the core is halted • When set: — clears all active exception status of the processor — forces a return to Thread mode — forces an IPSR of 0. A debugger must re-initialize the stack.
[0]	-	-	Reserved.

6.2.4 Configuration and Control Register

The Configuration and Control Register permanently enables stack alignment and causes unaligned accesses to result in a HardFault.

The register address, access type, and reset value are:

Address `0xE000ED14`

Access Read-only

Reset value `0x00000208`

Figure 6-4 on page 6-8 shows the bit assignments of the Configuration and Control Register.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

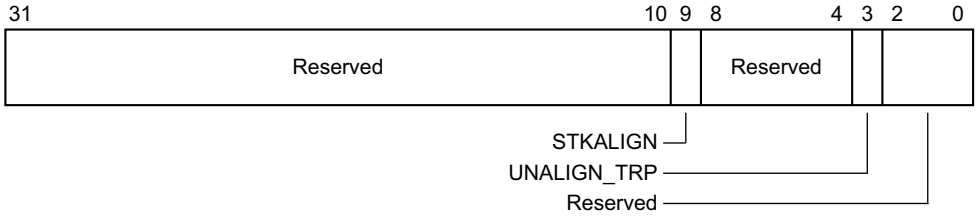

Figure 6-4 Configuration and Control Register bit assignments

Table 6-5 lists the bit assignments of the Configuration and Control Register.

Table 6-5 Configuration and Control Register bit assignments

Bits	Field	Type	Function
[31:10]	-	-	Reserved.
[9]	STKALIGN	RO	Fixed at 1. On exception entry, all exceptions are entered with 8-byte stack alignment and the context to restore the exception is saved. The SP is restored on the associated exception return.
[8:4]	-	-	Reserved.
[3]	UNALIGN_TRP	RO	Fixed at 1. Indicates that all unaligned accesses result in a HardFault.
[2:0]	-	-	Reserved.

6.2.5 System handler priority registers

System handlers are a special class of exception handler that can have their priority set to any of the priority levels.

There are two system handler priority registers for prioritizing the following system handlers:

- SVCall, see *System Handler Priority Register 2*
- SysTick, see *System Handler Priority Register 3* on page 6-9
- PendSV, see *System Handler Priority Register 3* on page 6-9.

PendSV and SVCall are permanently enabled. You can enable or disable SysTick by writing to the SysTick Control and Status Register.

System Handler Priority Register 2

The register address, access type, and reset value is:

Address `0xE000ED1C`

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Access Read/write
Reset value 0x00000000

Figure 6-5 shows the bit assignments of the System Handler Priority Register 2.

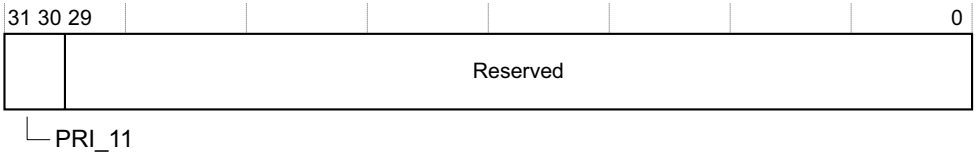

Figure 6-5 System Handler Priority Register 2 bit assignments

Table 6-6 lists the bit assignments for the System Handler Priority Register 2.

Table 6-6 System Handler Priority Register 2 bit assignments

Bits	Field	Function
[31:30]	PRI_11	Priority of system handler 11, SVCall
[29:0]	-	Reserved

System Handler Priority Register 3

The register address, access type, and reset value is:
Address 0xE000ED20
Access Read/write
Reset value 0x00000000

Figure 6-6 shows the bit assignments of the System Handler Priority Register 3.

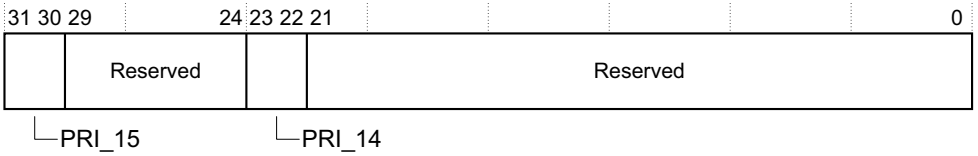

Figure 6-6 System Handler Priority Register 3 bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 6-7 lists the bit assignments of the System Handler Priority Registers.

Table 6-7 System Handler Priority Register 3 bit assignments

Bits	Field	Function
[31:30]	PRI_15	Priority of system handler 15, SysTick
[29:24]	-	Reserved
[23:22]	PRI_14	Priority of system handler 14, PendSV
[21:0]	-	Reserved

6.2.6 System Handler Control and State Register

Use the System Handler Control and State Register to determine or clear the pending status of SVCall.

The register address, access type, and reset value are:

Address `0xE000ED24`

Access Read/write

Reset value `0x00000000`

Figure 6-7 shows the bit assignments of the System Handler Control and State Register.

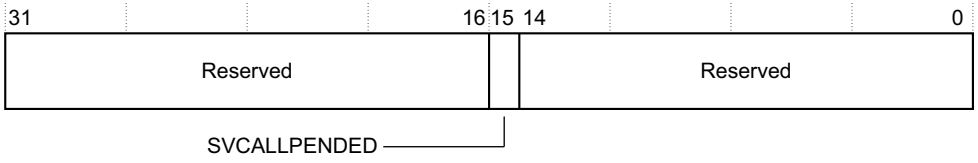

Figure 6-7 System Handler Control and State Register bit assignments

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Table 6-8 lists the bit assignments of the System Handler Control Register and State Register.

Table 6-8 System Handler Control and State Register bit assignments

Bits	Field	Function
[31:16]	-	Reserved
[15]	SVCALLPENDED	Reads as 1 if SVCall is pended. If written to: 1 = Set pending SVCall 0 = Clear pending SVCall
[14:0]	-	Reserved

Note

This register is only accessible as part of debug and not through the processor memory map.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Appendix A
Embedded Software Examples

This appendix describes C code examples that you can use and adapt in your own FPGA design. This appendix contains the following sections:

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.1 About embedded software examples

The examples in this appendix are provided as a starting point from where you can develop your own embedded software. You can use and adapt these examples for your own FPGA design.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.2 C declarations

This section describes the C declarations that are used in the examples described in this appendix.

The C declarations are:

```
/* C Declarations to support example code */
typedef unsigned int   UINT32;

/* Access 32-bit memory-mapped registers */
#define GET_UINT32(address, pData) (*(pData) = *((volatile UINT32 *)(address)))
#define PUT_UINT32(address, data)  (*((volatile UINT32 *)(address)) = (data))

/* Barrier operations defined using compiler intrinsics */
#define DataMemoryBarrier()                __dmb(0xF)        /* DMB */
#define DataSynchronizationBarrier()       __dsb(0xF)        /* DSB */
#define InstructionSynchronizationBarrier() __isb(0xF)       /* ISB */

/* Global interrupt enable operations on PRIMASK using compiler intrinsics */
#define DisableInterrupts()                __disable_irq()  /* CPSID I */
#define EnableInterrupts()                 __enable_irq()   /* CPSIE I */

/* SCS and NVIC register addresses */
#define CORTEXM1_SYSTICK_CSR          0xE000E010
#define CORTEXM1_SYSTICK_RELOAD       0xE000E014
#define CORTEXM1_SYSTICK_CURRENT      0xE000E018

#define CORTEXM1_NVIC_IRQ_SET_ENABLE  0xE000E100
#define CORTEXM1_NVIC_IRQ_CLR_ENABLE  0xE000E180
#define CORTEXM1_NVIC_IRQ_SET_PENDING 0xE000E200
#define CORTEXM1_NVIC_IRQ_CLR_PENDING 0xE000E280

#define CORTEXM1_SCB_ICS              0xE000ED04
#define CORTEXM1_SCB_SHPR3            0xE000ED20
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.3 Using memory barrier instructions for memory transactions

This section provides examples of how to use memory barrier instructions to:

- disable IRQ0, see Example A-1
- Initiate DMA transfer, Example A-2.

Example A-1 Disable IRQ0

```
/* Disable IRQ0 */
/* ... enters this code with PRIMASK set */
/* Clear IRQ0 enable bit in Interrupt Clear Enable register */
PUT_UINT32(CORTEXM1_NVIC_IRQ_CLR_ENABLE,  (1<<0));

/* Ensure store completes */
DataSynchronizationBarrier();

/* Ensure effect clearing enable bit occurs before next instruction */
InstructionSynchronizationBarrier();

/* Globally enable interrupts by clearing PRIMASK */
EnableInterrupts();
```

Example A-2 Initiate DMA transfer

```
/* Initiate DMA transfer */
/* Generate data to DMA */
GenerateData(data_buf);

/* Ensure all stores to data_buf complete before initiating DMA transfer */
DataMemoryBarrier();

/* Program DMA controller to start transfer */
StartDMATransfer(data_buf);
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.4 Examples of using the system timer

This section provides examples of how to use the system timer as:

- a periodic timer, see Example A-3
- an alarm timer, see Example A-4 on page A-6
- a simple timer, see Example A-5 on page A-7.

Example A-3 System Timer as a periodic timer

```
/* System Timer as periodic timer */

/* Frequency of processor clock for System Timer in Hz*/
#define PROCESSOR_FREQ                  50000000

/* Initialize System Timer as periodic timer */
void InitializePeriodicTimer(UINT32 frequency)
{
  /* Disable timer counter and interrupt */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x0);

  /* Clear current counter and COUNTFLAG */
  PUT_UINT32(CORTEXM1_SYSTICK_CURRENT, 0x0);

  /* Set reload register according to required frequency in Hz */
  PUT_UINT32(CORTEXM1_SYSTICK_RELOAD, (PROCESSOR_FREQ / frequency) - 1);

  /* Enable timer counter and interrupt */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x3);

  return;
}
/* Exception handler for SysTick interrupt */
void SysTick_Periodic_Handler(void) __irq
{
  UINT32 csr;

  /* Read the SysTick Control and Status register to determine and */
  /* cear COUNTFLAG */
  GET_UINT32(CORTEXM1_SYSTICK_CSR, &csr);

 /* Check COUNTFLAG and perform periodic processing if set */
  if ((csr & (1<<16)) != 0) {
     /* ... periodic processing */
  }
    return;
}
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Example A-4 System Timer as an alarm timer

```
/* System Timer as alarm timer */

/* Frequency of processor clock for System Timer in Hz*/
#define PROCESSOR_FREQ                 50000000

/* Initialize System Timer as one-shot alarm timer */

void InitializeAlarmTimer(UINT32 delay)
{
  /* Disable timer counter and interrupt */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x0);

  /* Clear current counter and COUNTFLAG */
  PUT_UINT32(CORTEXM1_SYSTICK_CURRENT, 0);

  /* Set reload register according to required delay in mS */
  PUT_UINT32(CORTEXM1_SYSTICK_RELOAD, delay * (PROCESSOR_FREQ / 1000));

  /* Enable timer counter and interrupt */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x3);

  return;
}

/* Exception handler for SysTick interrupt */
void SysTick_Alarm_Handler(void) __irq
{
  UINT32 csr;

  /* Read SysTick Control and Status register to determine COUNTFLAG */
  GET_UINT32(CORTEXM1_SYSTICK_CSR, &csr);

  /* Check COUNTFLAG and perform alarm processing if set */
  if ((csr & (1<<16)) != 0) {
    /* Disable timer counter and interrupt */
    PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x0);

    /* Ensure no SysTick interrupt is left pending */
    PUT_UINT32(CORTEXM1_SCB_ICSR, (1<<25));

    /* ... alarm processing */

  }
    return;
}
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Example A-5 System Timer as a simple timer

```
/* Simple timer */

/* Frequency of processor clock for System Timer in Hz*/
#define PROCESSOR_FREQ                  50000000

/* Maximum SysTick count (24 bits) */
#define SYSTICK_MAX_COUNT               0xFFFFFF

  UINT32 current;
  UINT32 ticks;

  /* Disable timer counter and interrupt */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x0);

  /* Clear current counter and COUNTFLAG */
  PUT_UINT32(CORTEXM1_SYSTICK_CURRENT, 0);

  /* Set reload register to maximum  */
  PUT_UINT32(CORTEXM1_SYSTICK_RELOAD, SYSTICK_MAX_COUNT);

  /* Enable timer counter */
  PUT_UINT32(CORTEXM1_SYSTICK_CSR, 0x1);

  /* ... code to be timed here */
  GET_UINT32(CORTEXM1_SYSTICK_CURRENT, &current);

  ticks = SYSTICK_MAX_COUNT - current;

  printf("Time = %d ticks or %d uS\n", ticks, ticks / (PROCESSOR_FREQ/1000000));
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.5 Example of how to configure and enable an external interrupt

This section provides an example of how to configure and enable external interrupt.

Example A-6 Configure and enable an external interrupt

```
/* Configure and Enable External Interrupt */

/* Configure and enable an external interrupt */

void ConfigureIRQ(UINT32 irq, UINT32 priority)
{
  /* Must be called with the interrupt disabled */

  UINT32 ipri;

  /* Calculate register offset for Interrupt Priority Register */
  UINT32 reg_offset = 4 * (irq / 4);

  /* Calculate bit offset within Interrupt Priority Register */
  UINT32 bit_offset = 8 * (irq % 4) + 6;

  /* All accesses to NVIC registers must be 32-bits so read */
  /* the Interrupt Priority Register to preserve other fields */
  GET_UINT32(CORTEXM1_NVIC_IRQ_PRI_BASE + reg_offset, &ipri);

  /* Mask out previous priority */
  ipri &= ~(0x3 << bit_offset);

  /* Set new priority */
  ipri |= (priority << bit_offset);

  /* Update Interrupt Priority Register with new priority */
  PUT_UINT32(CORTEXM1_NVIC_IRQ_PRI_BASE + reg_offset, ipri);

  /* Ensure any previous interrupt is un-pended to avoid */
  /* a spurious interrupt exception entry when it is enabled */
  PUT_UINT32(CORTEXM1_NVIC_IRQ_CLR_PENDING,  1<<irq);

  /* Set bit in Interrupt Set Enable Register to enable IRQ */
  PUT_UINT32(CORTEXM1_NVIC_IRQ_SET_ENABLE,  1<<irq);

  /* Ensure all stores complete */
  DataSynchronizationBarrier();

  return;
}
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

A.6 Example of how to configure and schedule a PendSV exception

This section provides an example of how to configure and schedule a PendSV exception.

Example A-7 Configure and schedule a PendSV exception

```
/* Configure priority of PendSV exception */

void ConfigurePendSV(UINT32 priority)
{
  UINT32 syspri;

  /* The PendSV exception is always enabled so set PRIMASK */
  /* to prevent it from occurring while being configured */
  DisableInterrupts();

  /* All accesses to SCS registers must be 32-bits so read */
  /* the System Handler Priority Register 3 to preserve other fields */
  GET_UINT32(CORTEXM1_SCB_SHPR3, &syspri);

  /* Mask out previous priority */
  syspri &= ~(0x3 << 22);

  /* Set new priority */
  syspri |= (priority << 22);

  /* Update System Handler Priority Register 3 with new priority */
  PUT_UINT32(CORTEXM1_SCB_SHPR3, syspri);

  /* Ensure the effect of the priority change occurs before */
  /* clearing PRIMASK to ensure that future PendSV exceptions */
  /* are taken at the new priority */
  DataSynchronizationBarrier();
  InstructionSynchronizationBarrier();

  EnableInterrupts();

  return;
}


/* Exception handler for IRQ1 for time-critical processing */

void IRQ1_Handler(void) __irq
{
  /* ... code executing at interrupt priority to handle */
  /* time-critical data transfers to the peripheral */
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

```
  /* Schedule a PendSV exception to handle other low-priority */
  /* operations while permitting other high priority exceptions */
  /* to be handled */
  PUT_UINT32(CORTEXM1_SCB_ICSR, 1<<28);

  return;
}



/* Exception handler for PendSV for low-priority processing */

void PendSV_Handler(void) __irq
{
  /* ... code to handle low-priority operations scheduled */
  /* from a high priority interrupt handler */

    return;
}



/* Initialization code in main() */

/* Configure PendSV with priority 0x3 (lowest priority) */
ConfigurePendSV(0x3);

/* Configure IRQ1 with priority 0x0 (higher than PendSV) */
ConfigureIRQ(0x1, 0x0);
```


Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Glossary

This glossary describes some of the terms used in technical documents from ARM Limited.

Aligned A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

Architecture The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6-M architecture.

ARM instruction An instruction of the ARM *Instruction Set Architecture* (ISA). These cannot be executed by the processor.

ARM state The processor state in which the processor executes the instructions of the ARM ISA. The processor only operates in Thumb state, never in ARM state.

Base register A register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the address that is sent to memory.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Big-endian Byte ordering scheme in which bytes of decreasing significance in a data word are stored at increasing addresses in memory.

See also Little-endian and Endianness.

Big-endian memory Memory in which:

- a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

See also Little-endian memory.

Breakpoint A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints can be removed after the program is successfully tested.

See also Watchpoint.

Byte An 8-bit data item.

Context The environment that each process operates in for a multitasking operating system.

Core A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

Core reset *See* Warm reset.

Debugger A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Endianness Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian.

Exception An error or event which can cause the processor to suspend the currently executing instruction stream and execute a specific exception handler or interrupt service routine. The exception could be an external interrupt or NMI, or it could be a fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs, normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt service routine to deal with the exception.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Exception handler
: *See* Interrupt service routine.

Exception vector
: *See* Interrupt vector.

Halfword
: A 16-bit data item.

Halt mode
: One of two mutually exclusive debug modes. In halt mode all processor execution halts when a breakpoint or watchpoint is encountered. All processor state, coprocessor state, memory and input/output locations can be examined and altered by the JTAG interface.

  See also Monitor debug-mode.

Host
: A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

Implementation-defined
: The behavior is not architecturally defined, but is defined and documented by individual implementations.

Interrupt service routine
: A program that control of the processor is passed to when an interrupt occurs.

Interrupt vector
: One of a number of fixed addresses in low memory that contains the first instruction of the corresponding interrupt service routine.

Little-endian
: Byte ordering scheme in which bytes of increasing significance in a data word are stored at increasing addresses in memory.

  See also Big-endian and Endianness.

Little-endian memory
: Memory in which:

  - a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address
  - a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

  See also Big-endian memory.

Load/store architecture
: A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

Macrocell
: A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Multi-layer An interconnect scheme similar to a cross-bar switch. Each master on the interconnect has a direct link to each slave, The link is not shared with other masters. This enables each master to process transfers in parallel with other masters. Contention only occurs in a multi-layer interconnect at a payload destination, typically the slave.

Private Peripheral Bus
Memory space at `0xE0000000` to `0xE00FFFFF`.

Processor A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Reserved A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

SBO *See* Should Be One.

SBZ *See* Should Be Zero.

Should Be One (SBO)
Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.

Should Be Zero (SBZ)
Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.

System memory map Address space at `0x00000000` to `0xFFFFFFFF`.

Thumb instruction A halfword that specifies an operation for an ARM processor in Thumb state to perform. Thumb instructions must be halfword-aligned.

Thumb state A processor that is executing Thumb (16-bit) halfword aligned instructions is operating in Thumb state.

Unaligned A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

Unpredictable For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Watchpoint A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Warm reset Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

Word A 32-bit data item.

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2007, 2008 ARM Limited. All rights reserved.
Non-Confidential