Cortex®-M4 Cycle Model

Version 9.1.0

User Guide

Non-Confidential

Cortex-M4 Cycle Model
User Guide

Copyright © 2017 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
April 2016	A	Non-Confidential	Restamp; Release 8.1.0. r0p1.
November 2016	B	Non-Confidential	Release 9.0.0.
February 2017	C	Non-Confidential	Release 9.1.0

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

A Cycle Model component is a library developed from ARM intellectual property (IP) that is generated through Cycle Model Studio. The Cycle Model then can be used within a virtual platform tool, for example, SoC Designer.

About This Guide

This guide provides all the information needed to configure and use the Cortex-M4 Cycle Model in SoC Designer.

Audience

This guide is intended for experienced hardware and software developers who create components for use with SoC Designer. You should be familiar with the following products and technology:

- SoC Designer
- Hardware design verification
- Verilog or SystemVerilog programming language

Conventions

This guide uses the following conventions:

Convention	Description	Example
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[text]	Square brackets [ ] indicate optional text.	`$CARBON_HOME/bin/modelstudio [ <filename> ]`
[text1 | text2]	The vertical bar | indicates "OR," meaning that you can supply text1 or text 2.	`$CARBON_HOME/bin/modelstudio [<name>.symtab.db | <name>.ccfg ]`

Also note the following references:

- References to C code implicitly apply to C++ as well.
- File names ending in .cc, .cpp, or .cxx indicate a C++ source file.

Further reading

The following publications provide information that relate directly to SoC Designer:

- *SoC Designer Installation Guide*
- *SoC Designer User Guide*
- *SoC Designer Standard Component Library Reference Manual*
- *SoC Designer AHBv2 Protocol Bundle User Guide*

The following publications provide reference information about ARM® products:

- *Cortex-M4 Technical Reference Manual*
- *AMBA 3 AHB-Lite Overview*
- *AMBA Specification (Rev 2.0)*
- *AMBA AHB Transaction Level Modeling Specification*
- *Architecture Reference Manual*

See http://infocenter.arm.com/help/index.jsp for access to ARM documentation.

The following publications provide additional information on simulation:

- IEEE 1666™ SystemC Language Reference Manual, (IEEE Standards Association)
- SPIRIT User Guide, Revision 1.2, SPIRIT Consortium.

Glossary

AMBA	*Advanced Microcontroller Bus Architecture*. The ARM open standard on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC).
AHB	*Advanced High-performance Bus*. A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.
APB	*Advanced Peripheral Bus*. A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports.
AXI	*Advanced eXtensible Interface*. A bus protocol that is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Cycle Model	A software object created by the Cycle Model Studio (or *Cycle Model compiler*) from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Cycle Model Studio	Graphical tool for generating, validating, and executing hardware-accurate software models. It creates a Cycle Model, and it also takes a Cycle Model as input and generates a component that can be used in SoC Designer, Platform Architect, or Accellera SystemC for simulation.
CASI	*ESL API Simulation Interface*, is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	*ESL API Debug Interface*, enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	*ESL API Profiling Interface*, enables collecting historical data from a component and displaying the results in various formats.
Component	Building blocks used to create simulated systems. Components are connected together with unidirectional transaction-level or signal-level connections.
ESL	*Electronic System Level*. A type of design and verification methodology that models the behavior of an entire system using a high-level language such as C or C++.
HDL	*Hardware Description Language*. A language for formal description of electronic circuits, for example, Verilog.
RTL	*Register Transfer Level*. A high-level hardware description language (HDL) for defining digital circuits.
SoC Designer	High-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug as well as architectural exploration.
SystemC	SystemC is a single, unified design and verification language that enables verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design.
Transactor	*Transaction adaptors*. You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.

Chapter 1
Using the Cycle Model in SoC Designer

This chapter describes the functionality of the Cycle Model component, and how to use it in SoC Designer. It contains the following sections:

- Cortex-M4 Functionality
- Adding and Configuring the SoC Designer Component
- Available Component ESL Ports
- Setting Component Parameters
- Debug Features
- Available Profiling Data

1.1 Cortex-M4 Functionality

The Cortex-M4 processor is a low-power processor that features low gate count, low interrupt latency, and low-cost debug. It is intended for deeply embedded applications that require fast interrupt response features. The processor implements the ARMv7-M architecture.

This section provides a summary of the functionality of the Cycle Model compared to that of the hardware, and the performance and accuracy of the Cycle Model. For details of the functionality of the hardware that the Cycle Model simulates, see the *Cortex-M4 Technical Reference Manual*.

- Fully Functional and Accurate Features
- Unsupported Hardware Features
- Features Additional to the Hardware

1.1.1 Fully Functional and Accurate Features

The following features of the Cortex-M4 hardware are fully implemented in the Cortex-M4 Cycle Model:

- Cortex-M4 Integer Core
- NVIC – Nested Vectored Interrupt Controller
- WIC – Wakeup Interrupt Controller Interface Support[1] (optional)
- AHB-Lite: ICode, DCode, and System Bus Interfaces
- APB v3.0 interface for accessing the external Private Peripheral Bus
- FPB – Flash Patch and Debug
- DWT – Debug Watchpoint and Trace
- MPU – Memory Protection Unit (optional)
- Bit-Banding (optional)
- Floating Point Unit (optional)

1. The Wakeup Interrupt Controller is not included in this Cycle Model. Support is for the interface only.

1.1.2 Unsupported Hardware Features

The following features of the Cortex-M4 hardware are not implemented in the Cortex-M4 Cycle Model:

- SW/JTAG-DP
- ITM
- ETM
- TPIU
- AHB-AP
- RTL clock-gating - The Cortex-M4 core supports architectural clock-gating, which is controlled by setting the CLKGATE_PRESENT parameter to 1 in the default.conf configuration file before creating the Cycle Model. The clock-gating option can be set when configuring the Cycle Model, but the Cycle Model *does not* currently support RTL-clock gating.
- ARMv7-M ROM table
- WIC Module
- The following register is not available to be read / written via debug transactions — for example, in the SoC Designer Registers window, or by accessing them directly from a debugger:
  - System Control: STIR (Software Trigger Interrupt Register 0xE000EF00)

  The functionality of this register, however, does exist and can be accessed by software running on the virtual platform.
- System Control: ICSR (Interrupt Control and State Register) is read-only via debug transactions even though some bits are writable via software running on the virtual platform.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the Cortex-M4 Cycle Model to enhance usability do not exist in the Cortex-M4 hardware:

- Semihosting Support. Semihosting enables the target application to communicate with the host operating system. This is used for external time synchronization, file handling operations, console input/output, and similar functionality.
- Debug and Profiling. For more information about debug and profiling features, refer to the sections Debug Features and Available Profiling Data, respectively.
- The “run to debug point” feature has been added. This feature forces the debugger to advance the processor to the debug state instead of having the Cycle Model get into a non-debuggable state. See “Run To Debug Point Feature” on page 36 for more information.

1.2 Adding and Configuring the SoC Designer Component

The following topics briefly describe how to use the component. See the *SoC Designer User Guide* (ARM DUI0956) for more information.

- SoC Designer Component Files
- Adding the Cycle Model to the Component Library
- Adding the Component to the SoC Designer Canvas

1.2.1 SoC Designer Component Files

The component files are the final output from the Cycle Model Studio compile and are the input to SoC Designer. There are two versions of the component; an optimized *release* version for normal operation, and a *debug* version.

On Linux, the *debug* version of the component is compiled without optimizations and includes debug symbols for use with gdb. The *release* version is compiled without debug information and is optimized for performance.

On Windows, the *debug* version of the component is compiled referencing the debug runtime libraries so it can be linked with the debug version of SoC Designer. The *release* version is compiled referencing the release runtime library. Both release and debug versions generate debug symbols for use with the Visual C++ debugger on Windows.

The provided component files are listed below:

Table 1-1 SoC Designer Component Files

Platform	File	Description
Linux	maxlib.lib<*model_name*>.conf	SoC Designer configuration file
	lib<*component_name*>.mx.so	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.so	SoC Designer component debug file
Windows	maxlib.lib<*model_name*>.windows.conf	SoC Designer configuration file
	lib<*component_name*>.mx.dll	SoC Designer component runtime file
	lib<*component_name*>.mx_DBG.dll	SoC Designer component debug file

Additionally, this User Guide PDF file is provided with the component.

1.2.2 Adding the Cycle Model to the Component Library

The compiled Cycle Model component is provided as a configuration file (*.conf*). To make the component available in the Component Window in SoC Designer Canvas, perform the following steps:

1. Launch SoC Designer Canvas.
2. From the *File* menu, select **Preferences**.
3. Click on **Component Library** in the list on the left.
4. Under the *Additional Component Configuration Files* window, click **Add**.
5. Browse to the location where the SoC Designer Cycle Model is located and select the component configuration file:
   – `maxlib.lib<model_name>.conf` (for Linux)
   – `maxlib.lib<model_name>.windows.conf` (for Windows)
6. Click **OK**.
7. To save the preferences permanently, click the **OK & Save** button.

The component is now available from the SoC Designer *Component Window.*

1.2.3 Adding the Component to the SoC Designer Canvas

Locate the component in the *Component Window* and drag it out to the Canvas. The component's appearance may vary depending on your specific device configuration.

Additional ports are provided depending on the model RTL configuration file, *default.conf*, used to create the Cycle Model.

1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports that are exposed in SoC Designer. See the *Cortex-M4 Technical Reference Manual* for more information.

All pins that are not listed in this table have been either tied or disconnected for performance reasons.

Note: *Some ESL component port values can be set using a component parameter. This includes the BIGEND port. In those cases, the parameter value will be used whenever the ESL port is not connected. If the port is connected, the connection value takes precedence over the parameter value.*

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
clk-in	Input Clock port. This port must be explicitly connected to a clock master.	Input	Clock Generator
AUXFAULT	Auxiliary fault status information. It is the input to AFSR (Auxiliary Fault Status Register in NVIC), where *value* = fault number (0-31).	Input	Signal slave
BIGEND	This port indicates the endianness; where 1=big endian and 0=little endian. It changes the *BIGEND* component parameter value. Note that this configuration is only latched during core reset.	Input	Signal slave
BRCHSTAT	Indicates the branch status information of the opcode currently in decode.	Output	Signal master
CURRPRI	Indicates what priority interrupt, or base boost, is being used now. CURRPRI represents the preemption priority; it does not indicate secondary priority.	Output	Signal master
EXREQD	Exclusive Request. EXREQD is an address phase control signal that indicates if the access is because of a LDREX or STREX.	Output	Signal master
EXREQS	Exclusive Request. EXREQS is an address phase control signal that indicates if the access is because of a LDREX or STREX.	Output	Signal master
EXRESPD	Exclusive Response. EXRESPD is a data phase response like HRESPD, but is only valid for exclusive accesses and indicates the success or failure of an exclusive operation.	Input	Signal slave
EXRESPS	Exclusive Response. EXRESPS is a data phase response like HRESPS, but is only valid for exclusive accesses and indicates the success or failure of an exclusive operation.	Input	Signal slave
HALTED	HALTED is asserted while the core is in debug.	Output	Signal master

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
IRQ	This port connects to external interrupt signals. It can be anywhere from 1 to 240 bits wide based on the configuration used to create the Cycle Model. The value must indicate the interrupt number [NumIRQ..0] and the *extValue must indicate whether the IRQ line is asserted (*extValue=1) or deasserted (*extValue=0).	Input	Signal slave
LOCKUP	Indicates that the core is locked up.	Output	Signal master
MEMATTRD	Memory attributes.	Output	Signal master
MEMATTRI	Memory attributes.	Output	Signal master
MEMATTRS	Memory attributes.	Output	Signal master
NMI	Non-maskable interrupt input to the NVIC; where 1 is used to assert NMI, and 0 is used to deassert NMI request.	Input	Signal slave
RST	This port is the core input reset. Note that the value is active high (instead of active low reset used in the hardware).	Input	Signal slave
RXEV	Causes a wakeup from a WFE instruction.	Input	Signal slave
SLEEPDEEP	Indication of core going into SLEEPDEEP mode; where 1 is used when going into SLEEPDEEP, and 0 is used when the core is waked up.	Output	Signal master
SLEEPHOLDACK	Acknowledges signal for SLEEPHOLDREQ that the core will be held in sleep mode.	Output	Signal master
SLEEPHOLDREQ	Request to extend sleep mode.	Input	Signal slave
SLEEPING	Indication that the core is going into SLEEP mode (because of WFE/WFI). The value 1 is used when the core goes into SLEEP mode, and 0 when the core is waken up.	Output	Signal master
STCALIB	System Tick calibration register.	Input	Signal slave
STCLK	System Tick Clock. See "Clock Ports" on page 19 for more information.	Input	Signal slave
SYSRESETREQ	System reset request.	Output	Signal master
TXEV	Event transmitted as a result of SEV instruction.	Output	Signal master
WICCLEAR	Indicates that the WIC must clear its mask.	Output	Signal master
WICDSACK	Active LOW acknowledge that deep sleep is WIC-based deep sleep.	Output	Signal master
WICDSREQ	Active LOW request for deep sleep to be WIC-based deep sleep.	Input	Signal slave
WICLOAD	Indicates that the WIC must reload its mask from WICMASKISR,WICMASKNMI and WIC-MASKRXEV.	Output	Signal master

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
WICMASKISR	Interrupt sensitivity mask used for WIC wake-up detection.	Output	Signal master
WICMASKMON	WIC DBG MON sensitivity.	Output	Signal master
WICMASKNMI	NMI sensitivity mask used for WIC wake-up detection.	Output	Signal master
WICMASKRXEV	RXEV sensitivity for WIC wake-up detection.	Output	Signal master
extSemi	Semihosting can be enabled by connecting this port to the SoC Designer semihost component, contained in the SoC Designer Standard Model Library (v3.0 or greater).	Output	Transaction master
ext_ppb	Private Peripheral Bus Interface. This bus master port implements the APB (v3.0) interface on the Cortex-M4 for accessing peripherals mapped in the external Private Peripheral Bus (PPB) region.	Output	APB Transaction master
mem_D	DCode Interface. See "AHB-Lite Transaction Master Ports" on page 18 for more information.	Output	AHB-Lite Transaction master
mem_I	ICode Interface. See "AHB-Lite Transaction Master Ports" on page 18 for more information.	Output	AHB-Lite Transaction master
mem_S	System Bus Interface. See "AHB-Lite Transaction Master Ports" on page 18 for more information.	Output	AHB-Lite Transaction master

1.3.1 Transaction Ports

1.3.1.1 AHB-Lite Transaction Master Ports

The *mem_I*, *mem_D*, and *mem_S* transaction master ports implement the AMBA AHB-Lite interface for the ICode, DCode, and System bus, respectively. These transaction master ports should be connected to AHBv2 slaves using either an MxAHBv2 bus component (where one side is an AHB Lite Master and the other side is an AHB Lite Slave) or a PL301 in between. See the *SoC Designer AHBv2 Protocol Bundle User Guide* for more information.

1.3.1.2 ext_ppb Bus Master Port

The *ext_ppb* bus master port implements the APB v3.0 interface on the Cortex-M4 for accessing peripherals mapped in the external Private Peripheral Bus (PPB) region. Data accesses to an address mapped to the external PPB space (0xE0040000 to 0xE00FFFFF) goes through this port.

Note: Address range seen by the ext_ppb port is from 0x40000 to 0xFEFFF (as opposed to 0xE00FFFFF to 0xE00FEFFF), i.e., the upper 12 bits are unused. Consequently, when defining the address map for peripheral components in the external peripheral space, the upper 12 bits of base address should be set to zero.

1.3.2 Clock Ports

clk_in is the clock port used to clock the core. The *STCLK* signal port can be used to clock the system tick timer. Note that the CLKSOURCE bit in the SYST_CSR register of the NVIC has to be set to '1' if the internal core clock is used to clock the system tick timer, or '0' if an external clock source is used. The reset value of CLKSOURCE bit is '0'.

1.4 Setting Component Parameters

You can change the settings of all the component parameters in SoC Designer Canvas, and of some of the parameters in SoC Designer Simulator. To modify the component's parameters :

1. In the Canvas, right-click on the component and select **Component Information**. You can also double-click the component. The *Edit Parameters* dialog box appears.

   The list of available parameters will be slightly different depending on the settings that you enabled in the configuration file (*default.conf*) when creating the component.

2. In the *Parameters* window, double-click the **Value** field of the parameter that you want to modify.
3. If it is a text field, type a new value in the *Value* field. If a menu choice is offered, select the desired option. The parameters are described in Table 1-3.

Table 1-3 Component Parameters

Name	Description	Allowed Values	Default Value	Runtime[1]
Align Waveforms	When set to *true*, waveforms dumped from the component are aligned with the SoC Designer simulation time. The reset sequence, however, is not included in the dumped data. When set to *false*, the reset sequence is dumped to the waveform data, however, the component time is not aligned with the SoC Designer time.	true, false	true	No
BIGEND	When set to *true*, configures the processor in big endian mode. Otherwise it works in little endian mode (default).	true, false	false	Yes
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	No
DNOTITRANS	When set to *true*, it disallows transactions on the I and D interfaces at the same time.	true, false	false	Yes
Dump Waveforms	Determines whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	Determines whether debug messages are logged for the component.	true, false	false	Yes
Enable PC Tracing	This parameter is obsolete and should be left at its default setting.	N/A	false	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
ext_ppb Enable Debug Messages	Determines whether debug messages are logged for the *ext_ppb* port.	true, false	false	Yes
ext_ppb PReady Default High	The transfer is extended if PREADY is held low during an access phase.	true, false	true	Yes
FPUDISABLE	Disables the FPU if present.	true, false	false	No
mem_D Align Data	Determines whether halfword and byte transactions will align data to the transaction size for this port. By default, data is not aligned.	true, false	false	No
mem_D Big Endian	Determines whether AHB data is treated as big endian for this port. By default, data is not sent as big endian.	true, false	false	No
mem_D Enable Debug Messages	Determines whether debug messages are logged for the *mem_D* port.	true, false	false	Yes
mem_I Align Data	Determines whether halfword and byte transactions will align data to the transaction size for this port. By default, data is not aligned.	true, false	false	No
mem_I Big Endian	Determines whether AHB data is treated as big endian for this port. By default, data is not sent as big endian.	true, false	false	No
mem_I Enable Debug Messages	Determines whether debug messages are logged for the *mem_I* port.	true, false	false	Yes
mem_S Align Data	Determines whether halfword and byte transactions will align data to the transaction size for this port. By default, data is not aligned.	true, false	false	No
mem_S Big Endian	Determines whether AHB data is treated as big endian for this port. By default, data is not sent as big endian.	true, false	false	No
mem_S Enable Debug Messages	Determines whether debug messages are logged for the *mem_S* port.	true, false	false	Yes
MPUDISABLE	Disables the MPU if present.	true, false	false	No
PC Tracing File	This parameter is obsolete and should be left at its default setting.	N/A	N/A	No

STKALIGNINIT	Configures the value of stack alignment.
	True = 8-byte alignment
	False = 4-byte alignment
		true, false	false	No

Waveform File [2]	Name of the waveform file.	*string*	CortexM4.VCD	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
Waveform Format	The format of the waveform dump file.	VCD, FSDB	VCD	No
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	No

1. *Yes* means the parameter can be dynamically changed during simulation, *No* means it can be changed only when building the system, *Reset* means it can be changed during simulation, but its new value will be taken into account only at the next reset.
2. When enabled, SoC Designer writes accumulated waveforms to the waveform file in the following situations: when the waveform buffer fills, when validation is paused and when validation finishes, and at the end of each validation run.

1.5 Debug Features

The Cortex-M4 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate, and control the registers and memory, and display disassembly for programs running on the Cycle Model in the SoC Designer Simulator or any debugger that supports CADI. A view can be accessed in SoC Designer Simulator by right clicking on the Cycle Model and choosing the appropriate menu entry.

- Register Information
- Run To Debug Point Feature
- Memory Information
- Disassembly View

1.5.1 Register Information

The Cortex-M4 Cycle Model has many sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area.

- Core Registers
- VFP Registers
- System Control Registers
- ID Registers
- MPU Registers
- SysTick Registers
- Debug Registers
- NVIC Registers
- FPB Registers
- DWT Registers

See the *Cortex-M4 Technical Reference Manual* for detailed descriptions of these registers.

1.5.1.1 Core Registers

The Core group contains the ARM Architectural registers.

Note that all read-write registers are writeable at debuggable point only. Otherwise, a warning is printed.

Table 1-4 Core Registers

Name	Description	Type
R0	R0 register	read-write
R1	R1 register	read-write
R2	R2 register	read-write
R3	R3 register	read-write
R4	R4 register	read-write
R5	R5 register	read-write

Table 1-4 Core Registers (continued)

Name	Description	Type
R6	R6 register	read-write
R7	R7 register	read-write
R8	R8 register	read-write
R9	R9 register	read-write
R10	R10 register	read-write
R11	R11 register	read-write
R12	R12 register	read-write
R13	R13/Stack Pointer (SP) register	read-write
R13_MAIN	R13_MAIN_MSP register	read-write
R13_PROCESS	R13_PROCESS_PSP register	read-write
R14	R14/Link Register (LR)	read-write
R15	R15/PC (Program Counter) Register	read-write
XPSR	Program Status Register	read-write[1]
PRIMASK	PRIMASK	read-write
BASEPRI	BASEPRI	read-only
FAULTMASK	FAULTMASK	read-write
CONTROL	CONTROL	read-write

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

1.5.1.2 NVIC Registers

The NVIC group provides access to the interrupt controller state.

Table 1-5 NVIC Registers

Name	Description	Type
NVIC_ISER0	NVIC Interrupt Set-Enable Register 0 (31_0) 0xE000E100	read-write
NVIC_ISER1[1]	NVIC Interrupt Set-Enable Register 1 (63_32) 0xE000E104	read-write
NVIC_ISER2[1]	NVIC Interrupt Set-Enable Register 2 (95_64) 0xE000E108	read-write
NVIC_ISER3[1]	NVIC Interrupt Set-Enable Register 3 (127_96) 0xE000E10C	read-write
NVIC_ISER4[1]	NVIC Interrupt Set-Enable Register 4 (159_128) 0xE000E110	read-write
NVIC_ISER5[1]	NVIC Interrupt Set-Enable Register 5 (191_160) 0xE000E114	read-write

Table 1-5 NVIC Registers (continued)

Name	Description	Type
NVIC_ISER6[1]	NVIC Interrupt Set-Enable Register 6 (223_192) 0xE000E118	read-write
NVIC_ISER7[1]	NVIC Interrupt Set-Enable Register 7 (239_224) 0xE000E11C	read-write
NVIC_ICER0	NVIC Interrupt Clear-Enable Register 0 (31_0) 0xE000E180	read-write
NVIC_ICER1[1]	NVIC Interrupt Clear-Enable Register 1 (63_32) 0xE000E184	read-write
NVIC_ICER2[1]	NVIC Interrupt Clear-Enable Register 2 (95_64) 0xE000E188	read-write
NVIC_ICER3[1]	NVIC Interrupt Clear-Enable Register 3 (127_96) 0xE000E18C	read-write
NVIC_ICER4[1]	NVIC Interrupt Clear-Enable Register 4 (159_128) 0xE000E190	read-write
NVIC_ICER5[1]	NVIC Interrupt Clear-Enable Register 5 (191_160) 0xE000E194	read-write
NVIC_ICER6[1]	NVIC Interrupt Clear-Enable Register 6 (223_192) 0xE000E198	read-write
NVIC_ICER7[1]	NVIC Interrupt Clear-Enable Register 7 (239_224) 0xE000E19C	read-write
NVIC_ISPR0	NVIC Interrupt Set-Pending Register 0 (31_0) 0xE000E200	read-write
NVIC_ISPR1[1]	NVIC Interrupt Set-Pending Register 1 (63_32) 0xE000E204	read-write
NVIC_ISPR2[1]	NVIC Interrupt Set-Pending Register 2 (95_64) 0xE000E208	read-write
NVIC_ISPR3[1]	NVIC Interrupt Set-Pending Register 3 (127_96) 0xE000E20C	read-write
NVIC_ISPR4[1]	NVIC Interrupt Set-Pending Register 4 (159_128) 0xE000E210	read-write
NVIC_ISPR5[1]	NVIC Interrupt Set-Pending Register 5 (191_160) 0xE000E214	read-write
NVIC_ISPR6[1]	NVIC Interrupt Set-Pending Register 6 (223_192) 0xE000E218	read-write
NVIC_ISPR7[1]	NVIC Interrupt Set-Pending Register 7 (239_224) 0xE000E21C	read-write
NVIC_ICPR0	NVIC Interrupt Clear-Pending Register 0 (31_0) 0xE000E280	read-write
NVIC_ICPR1[1]	NVIC Interrupt Clear-Pending Register 1 (63_32) 0xE000E284	read-write

Table 1-5 NVIC Registers (continued)

Name	Description	Type
NVIC_ICPR2[1]	NVIC Interrupt Clear-Pending Register 2 (95_64) 0xE000E288	read-write
NVIC_ICPR3[1]	NVIC Interrupt Clear-Pending Register 3 (127_96) 0xE000E28C	read-write
NVIC_ICPR4[1]	NVIC Interrupt Clear-Pending Register 4 (159_128) 0xE000E290	read-write
NVIC_ICPR5[1]	NVIC Interrupt Clear-Pending Register 5 (191_160) 0xE000E294	read-write
NVIC_ICPR6[1]	NVIC Interrupt Clear-Pending Register 6 (223_192) 0xE000E298	read-write
NVIC_ICPR7[1]	NVIC Interrupt Clear-Pending Register 7 (239_224) 0xE000E29C	read-write
NVIC_IABR0	NVIC Interrupt Active Bit Register 0 (31_0) 0xE000E300	read-only
NVIC_IABR1[1]	NVIC Interrupt Active Bit Register 1 (63_32) 0xE000E304	read-only
NVIC_IABR2[1]	NVIC Interrupt Active Bit Register 2 (95_64) 0xE000E308	read-only
NVIC_IABR3[1]	NVIC Interrupt Active Bit Register 3 (127_96) 0xE000E30C	read-only
NVIC_IABR4[1]	NVIC Interrupt Active Bit Register 4 (159_128) 0xE000E310	read-only
NVIC_IABR5[1]	NVIC Interrupt Active Bit Register 5 (191_160) 0xE000E314	read-only
NVIC_IABR6[1]	NVIC Interrupt Active Bit Register 6 (223_192) 0xE000E318	read-only
NVIC_IABR7[1]	NVIC Interrupt Active Bit Register 7 (239_224) 0xE000E31C	read-only
NVIC_IPR0	Interrupt Priority Register 0 (0-3)	read-write
NVIC_IPR1	Interrupt Priority Register 1 (4-7)	read-write
NVIC_IPR2	Interrupt Priority Register 2 (8-11)	read-write
NVIC_IPR3	Interrupt Priority Register 3 (12-15)	read-write
NVIC_IPR4	Interrupt Priority Register 4 (16-19)	read-write
NVIC_IPR5	Interrupt Priority Register 5 (20-23)	read-write
NVIC_IPR6	Interrupt Priority Register 6 (24-27)	read-write
NVIC_IPR7	Interrupt Priority Register 7 (28-31)	read-write
NVIC_IPR8[1]	Interrupt Priority Register 8 (32-35)	read-write
NVIC_IPR9[1]	Interrupt Priority Register 9 (36-39)	read-write

Table 1-5 NVIC Registers (continued)

Name	Description	Type
NVIC_IPR10[1]	Interrupt Priority Register 10 (40-43)	read-write
NVIC_IPR11[1]	Interrupt Priority Register 11 (44-47)	read-write
NVIC_IPR12[1]	Interrupt Priority Register 12 (48-51)	read-write
NVIC_IPR13[1]	Interrupt Priority Register 13 (52-55)	read-write
NVIC_IPR14[1]	Interrupt Priority Register 14 (56-59)	read-write
NVIC_IPR15[1]	Interrupt Priority Register 15 (60-63)	read-write
NVIC_IPR16[1]	Interrupt Priority Register 16 (64-67)	read-write
NVIC_IPR17[1]	Interrupt Priority Register 17 (68-71)	read-write
NVIC_IPR18[1]	Interrupt Priority Register 18 (72-75)	read-write
NVIC_IPR19[1]	Interrupt Priority Register 19 (76-79)	read-write
NVIC_IPR20[1]	Interrupt Priority Register 20 (80-83)	read-write
NVIC_IPR21[1]	Interrupt Priority Register 21 (84-87)	read-write
NVIC_IPR22[1]	Interrupt Priority Register 22 (88-91)	read-write
NVIC_IPR23[1]	Interrupt Priority Register 23 (92-95)	read-write
NVIC_IPR24[1]	Interrupt Priority Register 24 (96-99)	read-write
NVIC_IPR25[1]	Interrupt Priority Register 25 (100-103)	read-write
NVIC_IPR26[1]	Interrupt Priority Register 26 (104-107)	read-write
NVIC_IPR27[1]	Interrupt Priority Register 27 (108-111)	read-write
NVIC_IPR28[1]	Interrupt Priority Register 28 (112-115)	read-write
NVIC_IPR29[1]	Interrupt Priority Register 29 (116-119)	read-write
NVIC_IPR30[1]	Interrupt Priority Register 30 (120-123)	read-write
NVIC_IPR31[1]	Interrupt Priority Register 31 (124-127)	read-write
NVIC_IPR32[1]	Interrupt Priority Register 32 (128-131)	read-write
NVIC_IPR33[1]	Interrupt Priority Register 33 (132-135)	read-write
NVIC_IPR34[1]	Interrupt Priority Register 34 (136-139)	read-write
NVIC_IPR35[1]	Interrupt Priority Register 35 (140-143)	read-write
NVIC_IPR36[1]	Interrupt Priority Register 36 (144-147)	read-write
NVIC_IPR37[1]	Interrupt Priority Register 37 (148-151)	read-write
NVIC_IPR38[1]	Interrupt Priority Register 38 (152-155)	read-write

Table 1-5 NVIC Registers (continued)

Name	Description	Type
NVIC_IPR39[1]	Interrupt Priority Register 39 (156-159)	read-write
NVIC_IPR40[1]	Interrupt Priority Register 40 (160-163)	read-write
NVIC_IPR41[1]	Interrupt Priority Register 41 (164-167)	read-write
NVIC_IPR42[1]	Interrupt Priority Register 42 (168-171)	read-write
NVIC_IPR43[1]	Interrupt Priority Register 43 (172-175)	read-write
NVIC_IPR44[1]	Interrupt Priority Register 44 (176-179)	read-write
NVIC_IPR45[1]	Interrupt Priority Register 45 (180-183)	read-write
NVIC_IPR46[1]	Interrupt Priority Register 46 (184-187)	read-write
NVIC_IPR47[1]	Interrupt Priority Register 47 (188-191)	read-write
NVIC_IPR48[1]	Interrupt Priority Register 48 (192-195)	read-write
NVIC_IPR49[1]	Interrupt Priority Register 49 (196-199)	read-write
NVIC_IPR50[1]	Interrupt Priority Register 50 (200-203)	read-write
NVIC_IPR51[1]	Interrupt Priority Register 51 (204-207)	read-write
NVIC_IPR52[1]	Interrupt Priority Register 52 (208-211)	read-write
NVIC_IPR53[1]	Interrupt Priority Register 53 (212-215)	read-write
NVIC_IPR54[1]	Interrupt Priority Register 54 (216-219)	read-write
NVIC_IPR55[1]	Interrupt Priority Register 55 (220-223)	read-write
NVIC_IPR56[1]	Interrupt Priority Register 56 (224-227)	read-write
NVIC_IPR57[1]	Interrupt Priority Register 57 (228-231)	read-write
NVIC_IPR58[1]	Interrupt Priority Register 58 (232-235)	read-write
NVIC_IPR59[1]	Interrupt Priority Register 59 (236-239)	read-write

1. This register is available only if it was defined in the configuration when the Cycle Model was built.

1.5.1.3 ID Registers

The ID registers hold information on what ARM features were compiled into the Cycle Model, what version numbers were used, and so on.

Table 1-6 ID Registers

Name	Description	Type
CPUID	CPUID Base Register - 0xE000ED00	read-only
ID_PFR0	Processor Feature Register 0 - 0xE000ED40	read-only
ID_PFR1	Processor Feature Register 1 - 0xE000ED44	read-only
ID_DFR0	Debug Feature Register 0 - 0xE000ED48	read-only
ID_AFR0	Auxiliary Feature Register 0 - 0xE000ED4C	read-only
ID_MMFR0	Memory Model Feature Register 0 - 0xE000ED50	read-only
ID_MMFR1	Memory Model Feature Register 1 - 0xE000ED54	read-only
ID_MMFR2	Memory Model Feature Register 2 - 0xE000ED58	read-only
ID_MMFR3	Memory Model Feature Register 3 - 0xE000ED5C	read-only
ID_ISAR0	ISA Feature Register 0 - 0xE000ED60	read-only
ID_ISAR1	ISA Feature Register 1 - 0xE000ED64	read-only
ID_ISAR2	ISA Feature Register 2 - 0xE000ED68	read-only
ID_ISAR3	ISA Feature Register 3 - 0xE000ED6C	read-only
ID_ISAR4	ISA Feature Register 4 - 0xE000ED70	read-only
PID0	Peripheral ID Register 0 - 0xE000EFE0	read-only
PID1	Peripheral ID Register 1 - 0xE000EFE4	read-only
PID2	Peripheral ID Register 2 - 0xE000EFE8	read-only
PID3	Peripheral ID Register 3 - 0xE000EFEC	read-only
PID4	Peripheral ID Register 4 - 0xE000EFD0	read-only
PID5	Peripheral ID Register 5 - 0xE000EFD4	read-only
PID6	Peripheral ID Register 6 - 0xE000EFD8	read-only
PID7	Peripheral ID Register 7 - 0xE000EFDC	read-only
CID0	Component ID Register 0	read-only
CID1	Component ID Register 1	read-only
CID2	Component ID Register 2	read-only
CID3	Component ID Register 3	read-only

1.5.1.4 System Control Registers

The system control registers provide miscellaneous configuration and status information.

Table 1-7 System Control Registers

Name	Description	Type
ICTR	Interrupt Controller Type register 0xE000E004	read-only
ACTLR	Auxiliary Control Register - 0xE000E008	read-write
ICSR	Interrupt Control and State Register - 0xE000ED04	read-only
VTOR	Vector Table Offset register 0xE000ED08	read-write
AIRCR	Application Interrupt Reset Control register 0xE000ED0C	read-write
SCR	System Control register 0xE000ED10	read-write
CCR	Config Control register 0xE000ED14	read-write
SHPR1	System Handlers 4-7Priority Register 1	read-write
SHPR2	System Handlers 8-11Priority Register 2	read-write
SHPR3	System Handlers 12-15 Priority Register 3	read-write
SHCSR	System Handler Control And State register 0xE000ED24	read-write[1]
CFSR	Configurable Fault Status Register 0xE000ED28	read-only
HFSR	HardFault Status register 0xE000ED2C	read-only
DFSR	Debug Status register 0xE000ED30	read-only
MMAR	Memory Manage Address register 0xE000ED34	read-only
BFAR	Bus Fault Address register 0xE000ED38	read-only
AFSR	Auxiliary Fault Status register 0xE000ED3C	read-only
CPACR	Coprocessor Access Control Register - 0xE000ED88	read-write

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

1.5.1.5 Debug Registers

The Debug group contains miscellaneous debug-related registers.

Table 1-8 Debug Registers

Name	Description	Type
DHCSR	Debug Halting Control and Status Register - 0xE000EDF0	read-write
DCRSR	Debug Core Register Selector Register - 0xE000EDF4	read-write[1]
DCRDR	Debug Core Register Data Register - 0xE000EDF8	read-write
DEMCR	Debug Exception and Monitor Control Register - 0xE000EDFC	read-write

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

1.5.1.6 MPU Registers

The MPU group contains registers for the Memory Protection Unit. It is present only if the MPU is enabled.

Table 1-9 MPU Registers

Name	Description	Type
MPU_TYPE	MPU Type register	read-only
MPU_CTRL	MPU Control register	read-write
MPU_RNR	MPU Region Number register	read-write
MPU_RBAR	MPU Base Address register	read-write
MPU_RASR	MPU Region Attribute register	read-write

1.5.1.7 SysTick Registers

The SysTick group contains information about ...

Table 1-10 SysTick Registers

Name	Description	Type
SYST_CSR	SysTick Control and Status Register- 0xE000E010	read-write[1]
SYST_RVR	SysTick Reload Value Register - 0xE000E014	read-write
SYST_CVR	SysTick Current Value Register - 0xE000E018	read-write
SYST_CALIB	SysTick Calibration Value Register - 0xE000E01C	read-only

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

1.5.1.8 FPB Registers

The FPB group contains registers pertaining to the hardware breakpoints.

Table 1-11 FPB Registers

Name	Description	Type
FP_CTRL	FP_CTRL register	read-write[1]
FP_REMAP	FP_REMAP register	read-write[1]
FP_COMP1	FlashPatch Comparator Register 1 - 0xE000200C	read-write
FP_COMP2	FlashPatch Comparator Register 2 - 0xE0002010	read-write
FP_COMP3	FlashPatch Comparator Register 3 - 0xE0002014	read-write
FP_COMP4	FlashPatch Comparator Register 4 - 0xE0002018	read-write
FP_COMP5	FlashPatch Comparator Register 5 - 0xE000201C	read-write
FP_COMP6	FlashPatch Comparator Register 6 - 0xE0002020	read-write
FP_COMP7	FlashPatch Comparator Register 7 - 0xE0002024	read-write
FPB_PID0	FPB Peripheral Identification Register 0 - 0xE0002FE0	read-only
FPB_PID1	FPB Peripheral Identification Register 1 - 0xE0002FE4	read-only
FPB_PID2	FPB Peripheral Identification Register 2 - 0xE0002FE8	read-only
FPB_PID3	FPB Peripheral Identification Register 3 - 0xE0002FEC	read-only
FPB_PID4	FPB Peripheral Identification Register 4 - 0xE0002FD0	read-only
FPB_PID5	FPB Peripheral Identification Register 5 - 0xE0002FD4	read-only
FPB_PID6	FPB Peripheral Identification Register 6 - 0xE0002FD8	read-only
FPB_PID7	FPB Peripheral Identification Register 7 - 0xE0002FDC	read-only
FPB_CID0	FPB Component Identification Register 0 - 0xE0002FF0	read-only
FPB_CID1	FPB Component Identification Register 1 - 0xE0002FF4	read-only

Table 1-11 FPB Registers (continued)

Name	Description	Type
FPB_CID2	FPB Component Identification Register 2 - 0xE0002FF8	read-only
FPB_CID3	FPB Component Identification Register 3 - 0xE0002FFC	read-only

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

1.5.1.9 DWT Registers

The DWT group contains registers pertaining to hardware watchpoints.

Table 1-12 DWT Registers

Name	Description	Type
DWT_CTRL	DWT Control Register - 0xE0001000	read-write[1]
DWT_CYCCNT	DWT Cycle Count Register - 0xE0001004	read-write
DWT_CPICNT	DWT CPI Count Register - 0xE0001008	read-write
DWT_EXECNT	DWT Exception Overhead Count Register - 0xE000100C	read-write
DWT_SLEEPCNT	DWT Sleep Count Register - 0xE0001010	read-write
DWT_LSUCNT	DWT LSU Count Register - 0xE0001014	read-write
DWT_FOLDCNT	DWT Folded-instruction Count Register - 0xE0001018	read-write
DWT_PCSR	DWT Program Counter Sample Register - 0xE000101C	read-only
DWT_COMP0	DWT Comparator Register 0 - 0xE0001020	read-write
DWT_MASK0	DWT Mask Register 0 - 0xE0001024	read-write
DWT_FUNCTION0	DWT Function Register 0 - 0xE0001028	read-write[1]
DWT_COMP1	DWT Comparator Register 1 - 0xE0001030	read-write
DWT_MASK1	DWT Mask Register 1 - 0xE0001034	read-write
DWT_FUNCTION1	DWT Function Register 1 - 0xE0001038	read-write[1]
DWT_COMP2	DWT Comparator Register 2 - 0xE0001040	read-write
DWT_MASK2	DWT Mask Register 2 - 0xE0001044	read-write
DWT_FUNCTION2	DWT Function Register 2 - 0xE0001048	read-write[1]
DWT_COMP3	DWT Comparator Register 3 - 0xE0001050	read-write
DWT_MASK3	DWT Mask Register 3 - 0xE0001054	read-write
DWT_FUNCTION3	DWT Function Register 3 - 0xE0001058	read-write[1]
DWT_ PID0	DWT Peripheral Identification Register 0 - 0xE0001FE0	read-only

Table 1-12 DWT Registers (continued)

Name	Description	Type
DWT_ PID1	DWT Peripheral Identification Register 1 - 0xE0001FE4	read-only
DWT_ PID2	DWT Peripheral Identification Register 2 - 0xE0001FE8	read-only
DWT_ PID3	DWT Peripheral Identification Register 3 - 0xE0001FEC	read-only
DWT_ PID4	DWT Peripheral Identification Register 4 - 0xE0001FD0	read-only
DWT_ PID5	DWT Peripheral Identification Register 5 - 0xE0001FD4	read-only
DWT_ PID6	DWT Peripheral Identification Register 6 - 0xE0001FD8	read-only
DWT_ PID7	DWT Peripheral Identification Register 7 - 0xE0001FDC	read-only
DWT_CID0	DWT Component Identification Register 0 - 0xE0001FF0	read-only
DWT_CID1	DWT Component Identification Register 1 - 0xE0001FF4	read-only
DWT_CID2	DWT Component Identification Register 2 - 0xE0001FF8	read-only
DWT_CID3	DWT Component Identification Register 3 - 0xE0001FFC	read-only

1. Contains fields that are read-only. For more information, refer to the ARM documentation on the register.

The values shown for the DWT registers will only be valid if the Cortex-M4 is configured with the DEBUG_LEVEL and TRACE_LEVEL values set to the highest value (3). These values are set in the *default.conf* file when the Cycle Model was generated. Also, the DWT must be enabled via the debug exception and monitor control register (TRCENA).

If any of these conditions are false, the values shown should not be considered valid.

1.5.1.10 VFP Registers

The Vector Floating Point (VFP) group contains registers for the optional Floating Point Processor Unit.

Table 1-13 Vector Floating Point (VFP) Registers

Name	Description	Type
S0	S0 Register	read-write
S1	S1 Register	read-write
S2	S2 Register	read-write
S3	S3 Register	read-write
S4	S4 Register	read-write
S5	S5 Register	read-write
S6	S6 Register	read-write
S7	S7 Register	read-write
S8	S8 Register	read-write
S9	S9 Register	read-write
S10	S10 Register	read-write
S11	S11 Register	read-write
S12	S12 Register	read-write
S13	S13 Register	read-write
S14	S14 Register	read-write
S15	S15 Register	read-write
S16	S16 Register	read-write
S17	S17 Register	read-write
S18	S18 Register	read-write
S19	S19 Register	read-write
S20	S20 Register	read-write
S21	S21 Register	read-write
S22	S22 Register	read-write
S23	S23 Register	read-write
S24	S24 Register	read-write
S25	S25 Register	read-write
S26	S26 Register	read-write
S27	S27 Register	read-write
S28	S28 Register	read-write
S29	S29 Register	read-write
S30	S30 Register	read-write

Table 1-13 Vector Floating Point (VFP) Registers (continued)

Name	Description	Type
S31	S31 Register	read-write
D0	D0 Register	read-write
D1	D1 Register	read-write
D2	D2 Register	read-write
D3	D3 Register	read-write
D4	D4 Register	read-write
D5	D5 Register	read-write
D6	D6 Register	read-write
D7	D7 Register	read-write
D8	D8 Register	read-write
D9	D9 Register	read-write
D10	D10 Register	read-write
D11	D11 Register	read-write
D12	D12 Register	read-write
D13	D13 Register	read-write
D14	D14 Register	read-write
D15	D15 Register	read-write
FPSCR	Floating Point Status and Control	read-write
FPCCR	Floating Point Context Control Register	read-write
FPCAR	Floating Point Context Access Register	read-write
FPDSCR	Floating Point Default Status Control Register	read-write
MVFR0	Media and VFP Feature Register 0	read-only
MVFR1	Media and VFP Feature Register 1	read-only

1.5.2 Run To Debug Point Feature

The “run to debug point” feature has been added to enhance Cycle Model debugging. This feature forces the processor into a coherent state called a “debug point”. When debugging, the Cycle Model is brought to the debug point automatically whenever a software breakpoint is hit (including single stepping). However, if a hardware breakpoint is reached, or the system is advanced by cycles within SoC Designer, the Cycle Model can get to a non-debuggable state. In this event, the *run to debug point* will advance the processor to the debug state. It does this by stalling the instruction within the decode stage and allowing all earlier instructions to complete. Once that has been accomplished, the Cycle Model will cause the system to stop simulating.

The run to debug point is available as a context menu item for the component within SoC Designer Simulator. It is also available in the disassembler view.

1.5.3 Memory Information

The SoC Designer Simulator memory space view gives a view of the memory as seen from a particular port. For example, if you specify the mem_D port (the data port), the view you see is the view of the memory as seen from that port.

Table 1-14 describes the available memory space views.

Table 1-14 Memory Spaces

Name	Description	Address Range	Access Size (bits)	Number of Blocks
memory	Main memory. This is the default memory space.	0x0:0xffffffff	8	1
mem_I	Physical Memory Space for port mem_I	0x0:0xffffffff	8	1
mem_D	Physical Memory Space for port mem_D	0x0:0xffffffff	8	1
mem_S	Physical Memory Space for port mem_S	0x0:0xffffffff	8	1
ext_ppb	Physical Memory Space for port ext_ppb	0x0:0xffffffff	8	1

1.5.4 Disassembly View

The Cortex-M4 Cycle Model supports a disassembly view of a program running on the Cycle Model in SoC Designer Simulator. To display the disassembly view in the SoC Designer Simulator, right-click on the Cortex-M4 Cycle Model and select **View Disassembly…** from the context menu.

All CADI windows support breakpoints – when double-clicking on the proper location a red dot will indicate that a breakpoint is currently active. To remove the breakpoints simply double-click on the same location again.

1.6 Available Profiling Data

Profiling data is enabled, and can be viewed using the Profiling Manager, which is accessible from the **Debug** menu in the SoC Designer Simulator. Hardware and software-based profiling are available for this Cycle Model.

1.6.1 Hardware Profiling

Hardware profiling includes the Core Events stream. The events supported by this stream are shown in Table 1-15.

Note: In the Cycle Model, the Hardware Profiling event counts increment cumulatively when enabled. This behavior differs from the behavior of the DWT counter registers, which wrap at 256.

Table 1-15 Cortex-M4 Profiling Events

Stream	Event Name	Description
Core Events	0x00_CLOCK_CYCLES	Cycle Count
	0x01_DWT_FOLDCNT	Folded Instruction Count
	0x02_DWT_LSUCNT	Load-Store Count
	0x03_DWT_SLEEPCNT	Sleep Overhead Count
	0x04_DWT_CPICNT	Instruction Cycle Count
	0x05_DWT_EXCCNT	Exception Overhead Count

1.6.2 Software Profiling

Software-based profiling is provided by SoC Designer. Profiling information is also available in the SoC Designer Profiler. See the *SoC Designer User Guide* (ARM DUI0956) or SoC Designer Profiler for more information.

Third Party Software Acknowledgement

ARM acknowledges and thanks the respective owners for the following software that is used by our product:

- **ELF (Executable and Linking Format) Tool Chain Product**