Cortex®-R4 Cycle Model

Version 9.0.0

User Guide

Non-Confidential

ARM®

ARM DUI 1005B (ID100716)

Cortex-R4 Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
April 2016	A	Non-Confidential	Restamp with Release 8.1.0. r1p3.
November 2016	B	Non-Confidential	Release 9.0.0.

Non-Confidential Proprietary Notice

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.

Product Status

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

Web Address

http://www.arm.com

Contents

Chapter 1.
Using the Cycle Model Component in SoC Designer Plus

Cortex-R4 and Cortex-R4F Functionality 12
Implemented Hardware Features 12
Unsupported Hardware Features 13
Features Additional to the Hardware 13
Adding and Configuring the SoC Designer Plus Component 14
SoC Designer Plus Component Files 14
Adding the Cycle Model to the Component Library 15
Adding the Component to the SoC Designer Canvas 15
Available Component ESL Ports 15
Setting Component Parameters 17
Debug Features 21
Register Information 21
Run To Debug Point 27
Memory Information 28
Disassembly View 28
Available Profiling Data 28
Hardware Profiling 28
Software Profiling 30

Preface

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

About This Guide

This guide provides all the information needed to configure and use the ARM Cortex-R4 or Cortex-R4F Cycle Model in SoC Designer Plus.

Audience

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

- SoC Designer Plus
- 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 Plus:

- *SoC Designer Plus Installation Guide*
- *SoC Designer Plus User Guide*
- *SoC Designer Plus Standard Model Library Reference Manual*
- *SoC Designer Plus AXIv2 Protocol Bundle User Guide*

External publications

The following publications provide reference information about ARM® products:

- *Cortex-R4 and Cortex-R4F Technical Reference Manual*
- *Cortex-R4 and Cortex-R4F Configuration and Sign-Off Guide*
- *AMBA Specification*
- *AMBA 3 APB Protocol Specification*
- *AMBA AXI Protocol 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 Carbon Model Studio (or *Carbon compiler*) from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Carbon 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 Plus, 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 Component in SoC Designer Plus

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

- Cortex-R4 and Cortex-R4F Functionality
- Adding and Configuring the SoC Designer Plus Component
- Available Component ESL Ports
- Setting Component Parameters
- Debug Features
- Available Profiling Data

1.1 Cortex-R4 and Cortex-R4F Functionality

The Cortex-R4 processor is a mid-range CPU for use in deeply-embedded systems. It includes the following functionality:

- a Data Processing unit (DPU) consisting of a five stage in-order pipeline which interfaces directly to the Prefetch unit and Load-Store Unit components of the Level 1 Memory System
- an Interrupt Controller (IC) with support for legacy ARM interrupts
- single and double precision floating point support (*Cortex-R4F only*)
- Level 1 memory system includes a three stage prefetch unit (PU) including a static branch predictor and call-return stack. The LSU is also three-stage and includes a store queue for efficient access to the Tightly Coupled Memory (TCM) ports
- AXI high-speed Advanced Microprocessor Bus Architecture (AMBA) L2 master and slave interfaces
- a Harvard cache with support for variable cache size configuration
- a single AXI Master port which supports multiple concurrent AXI requests from each of the two cache systems using the access ID fields in the AXI interface

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:

- Implemented Hardware Features
- Unsupported Hardware Features
- Features Additional to the Hardware

1.1.1 Implemented Hardware Features

Most hardware features have been implemented. Some functionality and register pin differences are listed in the next section.

See the *ARM Cortex-R4 and Cortex-R4F Technical Reference Manual* for more information.

1.1.2 Unsupported Hardware Features

The following features of the Cortex-R4 hardware are not implemented in the Cycle Model:

- The Cycle Model does not support any of the ETM features of the design.
- The Cycle Model does not support dual core functionality - a second core for redundancy.
- The Cycle Model does not support the debug functionality described in the v7 architecture specification. It does support a full compliant debug interface. The debug registers available in CP14 are supported where appropriate for software compliance. For example, the debug identification and status/control registers return the appropriate values when read from software or from the debug register interface.
- The following registers are not available to be read / written via debug transactions — for example, in the SoC Designer Plus Registers window, or by accessing them directly from your debugger:
  - Cache Registers
  - Micro Architectural Registers
  - Integration Test Registers
  - Perf Registers: The Software Increment register (CP15_SWINCR) is not supported

  The functionality of these registers, however, does exist and can be accessed by software running on the virtual platform.

1.1.3 Features Additional to the Hardware

The following features that are implemented in the Cortex-R4 Cycle Model do not exist in the Cortex-R4 hardware. These features have been added to the Cycle Model for enhanced usability.

- The component supports positive and negative level *irq* and *fiq* signal. This is configurable using the *negLogic* parameter (see Table 1-3 on page 17).
- 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” on page 27 for more information.

1.2 Adding and Configuring the SoC Designer Plus Component

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

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

1.2.1 SoC Designer Plus Component Files

The component files are the final output from the Model Studio compile and are the input to SoC Designer Plus. 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 Plus. 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 Plus Component Files

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

For more information on SoC Designer Canvas, see the *SoC Designer Plus User Guide*.

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. Depending on your configuration, ports may differ slightly from those listed in Table 1-2.

1.3 Available Component ESL Ports

Table 1-2 describes the ESL ports that are exposed in SoC Designer Plus. Some ports may or may not appear depending on the version of the ARM hardware. See the *ARM Cortex-R4 and Cortex-R4F Technical Reference Manual* for more information.

Table 1-2 ESL Component Ports

ESL Port	Description	Direction	Type
ACLKENM	Clock enable for the AXI master port. By default it is set to 1 (true) through a component parameter. If another component is connected through this port, it will override the default value.	Input	Signal slave
ACLKENS	Clock enable for the AXI slave port. By default it is set to 1 (true) through a component parameter. If another component is connected through this port, it will override the default value.	Input	Signal slave
DBGEN	Invasive debug enable.	Input	Signal slave
DBGRESTART	External restart request	Input	Signal slave
EDBGRQ	External debug request	Input	Signal slave
axi_s	AXI ACP (Accelerator Coherency Port) Slave port. (Available only when it has been enabled in the configuration file.)	Input	AXI Transaction slave
fiq [1]	Cortex-R4 processor private FIQ request input lines: 0 = do not activate fast interrupt 1 = activate fast interrupt	Input	Signal slave
irq [1]	Cortex-R4 processor normal IRQ request input lines: 0 = do not activate interrupt 1 = activate interrupt	Input	Signal slave
irqaddr	Address of the IRQ. This signal must be stable when IRQADDRV is asserted.	Input	Signal slave
clk_in	Input clock.	Input	Clock slave
DBGACK	Debug acknowledge signal.	Output	Signal master
DBGRESTARTED	Used with DBGRESTART to move between Debug and Normal state.	Output	Signal master
IRQADDRV	Indicates that IRQADDR is valid.	Output	Signal master

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Direction	Type
STANBYWFI	Indicates that the processor is in Standby mode and the processor clock is stopped. You can use this signal for TCMs RAM clock gating.	Output	Signal master
axi_m	AXI Master port.	Output	AXI Transaction master
extSemi	Semihosting can be enabled by connecting this port to the SoC Designer Plus semihost component contained in the SoC Designer Plus Standard Model Library (v3.0 or greater).	Output	Transaction master

1. For these interrupt ports, the active high/low setting is controlled by the `negLogic` component parameter. The default is active high.

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 ACLKENM, ACLKENS, and DBGEN ports. 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.

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 **Edit Parameters...**. 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 (for example, *cr4cell_defs.v*).

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]
ACLKENM	Clock enable for the AXI master port.	true, false	true	Yes
ACLKENS	Clock enable for the AXI slave port.	true, false	true	Yes
Align Waveforms	When *true*, waveforms dumped by the component are aligned with the SoC Designer Plus simulation time. The reset sequence, however, is not included in the dumped data. When *false*, the reset sequence is dumped to the waveform data, however, the component time is not aligned with SoC Designer Plus time.	true, false	true	No
axi_m Enable Debug Messages	Whether debug messages are logged for the AXI master port.	true, false	false	Yes
axi_s axi_size0	These parameters are obsolete and should be left at their default values.[2]	0x0 – 0x100000000	0x100000000	No
axi_s axi_size[1-5] [3]		0x0 – 0x100000000	0x0	No
axi_s axi_start[0-5]		0x0 – 0xffffffff	0x00000000	No
axi_s Enable Debug Messages	Whether debug messages are logged for the slave port.	true, false	false	Yes
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
CFGATCMSZ	Selects the ATCM region size. The size is in KB.	0x0 = 0 0x3 = 4 0x4 = 8 0x5 = 16 0x6 = 32 0x7 = 64 0x8 = 128 0x9 = 256 0xA = 512 0xB = 1024 0xC = 2048 0xD = 4096 0xE = 8192	0xE (8MB)	No
CFGATCREAD-WAIT	ATCM Read Delay Cycles	0-15	0	No
CFGATCWRITE-WAIT	ATCM Write Delay Cycles	0-15	0	No
CFGB0TCREAD WAIT	BTCM0 Read Delay Cycles	0-15	0	No
CFGB0TCWRITE WAIT	BTCM0 Read Delay Cycles	0-15	0	No
CFGB1TCREAD WAIT	BTCM1 Read Delay Cycles	0-15	0	No
CFGB1TCWRITE WAIT	BTCM1 Read Delay Cycles	0-15	0	No
CFGBTCMSZ	Selects the BTCM region size.	Same values as *CFGATCMSZ*	0xE (8MB)	No
CFGEE	Post-reset EE-bit value. When *true*, indicates data big-endian mode.	true, false	false	No
CFGIE	Instruction side endianness. When *true*, indicates instruction big-endian mode.	true, false	false	No
CFGNMFI	Enables non-maskable fast interrupts (FIQ).	true, false	false	No
DBGEN	Debug enable	true, false	false	No
Dump Waveforms	Whether SoC Designer Plus dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	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 value.	N/A.	false	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
ENTCM1IF	Enable B1TCM interface. If this parameter is set to *false*, only the B0TCM port will be used by the core.	true, false	false	No
ERRENRAM	Enables external error checking on the TCM ports on reset.	0x0 = All disabled 0x1 = ATCM enabled 0x2 = B0TCM enabled 0x4 = B1TCM enabled 0x7 = All enabled	0x0	No
INITRAMA	When *true*, enables ATCM at reset.	true, false	false	No
INITRAMB	When *true*, enables BTCM at reset.	true, false	false	No
LOCZRAMA	Determines the initial base address of the ATCM or BTCM.	0x0 = BTCM initial address 0, ATCM initial address is implementation defined 0x1 = ATCM initial address 0, BTCM initial address is implementation defined	0x0	No
negLogic	Sets IRQ/FIQ assertion to use negative logic. Default of *false* means 0=off and 1=on. *True* means 0=on and 1=off.	true, false	false	Yes
PARECCENRAM	Enables parity or ECC checking on each TCM interface.	0x0 = All disabled 0x1 = ATCM enabled 0x2 = B0TCM enabled 0x4 = B1TCM enabled 0x7 = All enabled	0x0	No
PARLVRAM	Selects between odd and even parity for caches, TCMs, and buses. (*True* indicates even parity).	true, false	false	No
PC Tracing File	This parameter is obsolete and should be left at its default value.	N/A	N/A	No
RMWENRAM	Enables read-modify-write on each TCM interface.	0x00 = ATCM and BTCM disabled 0x01 = ATCM enabled 0x10 = BTCM enabled 0x11 = ATCM and BTCM enabled	0x00	No

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Runtime[1]
SLBTCMSB	Use most significant bit of BTCM address to select B0TCM. Use bit[3] of the BTCM address if this parameter is *false*. Note: This parameter is ignored if *ENTCM1IF* is *false*.	true, false	false	No
TEINIT	When *false*, set the exception handling state to ARM mode. When *true*, set the exception handling state to Thumb mode.	true, false	false	No
VINITHI	When *true*, enables 'HiVecs' (High Exception Vector Mode).	true, false	false	No
Waveform File[4]	Name of the waveform file.	*string*	carbon_CORTEXR4.fsdb	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. ARM recommends using the Memory Map Editor (MME) in SoC Designer Plus, which provides centralized viewing and management of the memory regions available to the components in a system. For information about migrating existing systems to use the MME, refer to Chapter 9 of the *SoC Designer Plus User Guide*.
3. The size of a memory region depends on the "axi_s axi_start[M]" and "axi_s axi_size[M]" parameters. The end address is calculated as StartAddr +Size -1. The size of the memory region must not exceed the value of 0x100000000. If the sum of StartAddr+Size is greater than 0x100000000, the size of the memory region is reduced to the difference: 0x100000000-StartAddr.
4. When enabled, SoC Designer Plus 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-R4 Cycle Model has a debug interface (CADI) that allows you to view, manipulate, and control the registers and memory. A view can be accessed in SoC Designer Plus by right clicking on the Cycle Model and choosing the appropriate menu entry.

The following topics are discussed in this section:

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

1.5.1 Register Information

The Cortex-R4 Cycle Model has many sets of registers that are accessible via the debug interface. Registers are grouped into sets according to functional area. Note that some register tabs will not be present if that functionality is not built into the Cycle Model.

- Core Registers
- Control Registers
- Direct Memory Access Registers
- Debug Registers
- Single-Precision Floating-Point Registers
- Double-Precision Floating-Point Registers
- ID Registers
- Jazelle Registers
- Memory Protection Unit Registers
- Performance Monitoring Unit Registers
- Tightly Coupled Memory Control Registers

See the *ARM Cortex-R4F Technical Reference Manual* for detailed descriptions of these registers.

1.5.1.1 Core Registers

The Core group contains the ARM architectural registers.

Table 1-4 Core Registers

Name	Description	Type
R0	R0 register	read-write [1]
R1	R1 register	read-write [1]
R2	R2 register	read-write [1]
R3	R3 register	read-write [1]
R4	R4 register	read-write [1]
R5	R5 register	read-write [1]
R6	R6 register	read-write [1]
R7	R7 register	read-write [1]
R8	R8 register	read-write [1]
R9	R9 register	read-write [1]
R10	R10 register	read-write [1]
R11	R11 register	read-write [1]
R12	R12 register	read-write [1]
R13	R13/Stack Pointer register	read-write [1]
R14	R14/Link register	read-write [1]
R15	R15/Program Counter register	read-write
CPSR	Current Program Status register	read-only
Mode	Current Processor Mode register	read-only
R8_USR	User Mode Register 8	read-write [1]
R9_USR	User Mode Register 9	read-write [1]
R10_USR	User Mode Register 10	read-write [1]
R11_USR	User Mode Register 11	read-write [1]
R12_USR	User Mode Register 12	read-write [1]
R13_USR	User Mode Register 13	read-write [1]
R14_USR	User Mode Register 14	read-write [1]
R13_IRQ	IRQ Mode Register 13	read-write [1]
R14_IRQ	IRQ Mode Register 14	read-write [1]
SPSR_IRQ	Saved Program Status Register (IRQ)	read-only
R8_FIQ	FIQ Mode Register 8	read-write [1]
R9_FIQ	FIQ Mode Register 9	read-write [1]

Table 1-4 Core Registers (continued)

Name	Description	Type
R10_FIQ	FIQ Mode Register 10	read-write [1]
R11_FIQ	FIQ Mode Register 11	read-write [1]
R12_FIQ	FIQ Mode Register 12	read-write [1]
R13_FIQ	FIQ Mode Register 13	read-write [1]
R14_FIQ	FIQ Mode Register 14	read-write [1]
SPSR_FIQ	Saved Program Status Register (FIQ)	read-only
R13_SVC	SVC Mode Register 13	read-write [1]
R14_SVC	SVC Mode Register 14	read-write [1]
SPSR_SVC	Saved Program Status Register (SVC)	read-only
R13_ABT	ABT Mode Register 13	read-write [1]
R14_ABT	ABT Mode Register 14	read-write [1]
SPSR_ABT	Saved Program Status Register (ABT)	read-only
R13_UND	UND Mode Register 13	read-write [1]
R14_UND	UND Mode Register 14	read-write [1]
SPSR_UND	Saved Program Status Register (UND)	read-only

1. Writeable at debug point only.

1.5.1.2 Control Registers

The Control group is used for system control and configuration using the CP15 Control registers.

Table 1-5 Control Registers

Name	Description	Type
CP15_CONTROL	System Control register	read-write
CP15_AUXILIARY_CONTROL	Auxiliary Control register	read-write
CP15_SECONDARY_AUXILIARY_CONTROL	Secondary Auxiliary Control register	read-write
CP15_COPROCESSOR_ACCESS	Coprocessor Access register	read-write
CP15_DFSR	Data Fault Status register	read-write
CP15_ADFSR	Auxiliary Data Fault Status register	read-write
CP15_IFSR	Instruction Fault Status register	read-write
CP15_AIFSR	Auxiliary Instruction Fault Status register	read-write
CP15_DFAR	Data Fault Address register	read-write
CP15_IFAR	Instruction Fault Address register	read-write
CP15_CFLR	Correctable Fault Location register	read-write

1.5.1.3 Direct Memory Access Registers

The DMA group provides AXI slave port configuration registers.

Table 1-6 DMA Registers

Name	Description	Type
CP15_SLAVE_PORT_CTL	Slave Port Control register	read-write

1.5.1.4 Debug Registers

The Debug group provides access to the current debug state.

Table 1-7 Debug Registers

Name	Description	Type
CP14_DIDR	Debug ID register	read-only
CP14_DRAR	Debug ROM Address register	read-only
CP14_DSAR	Debug Self Address Offset register	read-only
CP14_DSCR	Debug Status and Control register	read-only

1.5.1.5 Single-Precision Floating-Point Registers

The FP Single group contains FP single-precision floating-point number registers.

Table 1-8 FP Single Registers

Name	Description	Type
S0 - S31	Standard Single-precision FP Registers 0 through 31	read-write
FPSCR_1	Floating-Point Status and Control register	read-write
FBEXC_1	Floating-Point Exception register	read-write
FPSID_1	Floating-Point System ID register	read-only
MVFR0_1	Media and VFP Feature 0 register	read-only
MVFR1_1	Media and VFP Feature 1 register	read-only

1.5.1.6 Double-Precision Floating-Point Registers

The FP Double group contains FP double-precision floating-point number registers.

Table 1-9 FP Double Registers

Name	Description	Type
D0 - D15	Standard Double-precision FP Registers 0 through 15	read-write
FPSCR	Floating-Point Status and Control register	read-write
FBEXC	Floating-Point Exception register	read-write
FPSID	Floating-Point System ID register	read-only
MVFR0	Media and VFP Feature 0 register	read-only
MVFR1	Media and VFP Feature 1 register	read-only

1.5.1.7 ID Registers

The ID group contains registers that identify the device.

Table 1-10 ID Registers

Name	Description	Type
CP15_MAIN_ID	Main ID register	read-only
CP15_CACHE_TYPE	Cache Type register	read-only
CP15_TCM_TYPE	TCM Type register	read-only
CP15_MPU_TYPE	MPU Type register	read-only
CP15_MULTIPROCESSOR_ID	Multiprocessor ID register	read-only
CP15_PFR0	Processor Feature register 0	read-only
CP15_PFR1	Processor Feature register 1	read-only
CP15_DFR0	Debug Feature register 0	read-only
CP15_AFR1	Auxiliary Feature register 0	read-only
CP15_MMFR0	Memory Model Feature register 0	read-only
CP15_MMFR1	Memory Model Feature register 1	read-only
CP15_MMFR2	Memory Model Feature register 2	read-only
CP15_MMFR3	Memory Model Feature register 3	read-only
CP15_ISAR0	Instruction Set Attributes Register 0	read-only
CP15_ISAR1	Instruction Set Attributes Register 1	read-only
CP15_ISAR2	Instruction Set Attributes Register 2	read-only
CP15_ISAR3	Instruction Set Attributes Register 3	read-only
CP15_ISAR4	Instruction Set Attributes Register 4	read-only
CP15_ISAR5	Instruction Set Attributes Register 5	read-only
CP15_CURRENT_CACHE_SIZE_ID	Current Cache Size ID register	read-only
CP15_CURRENT_CACHE_LEVEL_ID	Current Cache Level ID register	read-only
CP15_CACHE_SIZE_SELECTION	Cache Size Selection register	read-write
CP15_BUILD_CONFIGURATION_1	Build Options 1 register	read-only
CP15_BUILD_CONFIGURATION_2	Build Options 2 register	read-only
CP15_FCSE_PID	FSCE PID register	RAZ, ignore writes
CP15_CTX_ID	Context ID register	read-write
CP15_TPID_RW	User R/W Thread and Process ID register	read-write
CP15_TPID_RO	User R/O Thread and Process ID register	read-write
CP15_TPID_PO	Privileged Only Thread and Process ID register	read-write

1.5.1.8 Jazelle Registers

The Jazelle group provides architecturally defined Jazelle registers.

Table 1-11 Jazelle Registers

Name	Description	Type
CP14_JID	Jazelle ID register	read-only
CP14_JOSCTL	Jazelle OS Control register	read-only
CP14_JCONF	Jazelle Main Configuration register	read-only

1.5.1.9 Memory Protection Unit Registers

The MPU group provides the Memory Protection Unit registers.

Table 1-12 MPU Registers

Name	Description	Type
CP15_MPU_REGION_BASE_ ADDRESS	MPU Region Base Address register	read-write
CP15_MPU_REGION_SIZE_ AND_ENABLE	MPU Region Size and Enable register	read-write
CP15_MPU_REGION_ACCESS_ CONTROL	MPU Region Access Control register	read-write
CP15_MPU_MEMORY_REGION _NUMBER	MPU Memory Region Number register	read-write

1.5.1.10 Performance Monitoring Unit Registers

The Perf group contains the Performance Monitoring Unit related registers.

Table 1-13 Perf Registers

Name	Description	Type
CP15_PMNC	Performance Monitor Control register	read-write
CP15_CNTENS	Count Enable Set register	read-write
CP15_CNTENC	Count Enable Clear register	read-write
CP15_FLAG	Overflow Flag Status register	read-write
CP15_PMNXSEL	Performance Counter Selection register	read-write
CP15_CCNT	Cycle Count register	read-write
CP15_EVTSEL	Event Select register	read-write
CP15_PMC	Performance Monitor Count register	read-write
CP15_USEREN	User Enable register	read-write
CP15_INTENS	Interrupt Enable Set register	read-write
CP15_INTENC	Interrupt Enable Clear register	read-write
CP15_IRQ_ENABLE_SET	nVAL IRQ Enable Set register	read-write

Table 1-13 Perf Registers (continued)

Name	Description	Type
CP15_FIQ_ENABLE_SET	nVAL FIQ Enable Set register	read-write
CP15_DBG_REQUEST_ENABLE_SET	nVAL Debug Request Enable Set register	read-write
CP15_RESET_ENABLE_SET	nVAL Reset Enable Set register	read-write
CP15_IRQ_ENABLE_CLEAR	nVAL IRQ Enable Clear register	read-write
CP15_FIQ_ENABLE_CLEAR	nVAL FIQ Enable Clear register	read-write
CP15_RESET_ENABLE_CLEAR	nVAL Reset Enable Clear register	read-write
CP15_DBG_REQUEST_ENABLE_CLEAR	nVAL Debug Request Enable Clear register	read-write

1.5.1.11 Tightly Coupled Memory Control Registers

The TCM Ctrl group provides Tightly Coupled Memory control and configuration registers.

Table 1-14 TCM Ctrl Registers

Name	Description	Type
CP15_BTCM_REGION	BTCM Region register	read-write
CP15_ATCM_REGION	ATCM Region register	read-write
CP15_TCM_SELECTION	TCM Selection register	read-write

1.5.2 Run To Debug Point

The “run to debug point” feature has been added to enhance Cycle Model debugging. The Cortex-R4 processor is a dual issue out of order completion machine. This means that while the processor is running it does not present a coherent programmer’s view state; instructions in the pipeline may be in different execution states.

This feature forces the processor into a coherent state called “run to 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 Plus, 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 (*Run to Debuggable Point*) for the component within SoC Designer Plus Simulator. It is also available in the disassembler view.

1.5.3 Memory Information

In SoC Designer Plus, the Cortex-R4 memory spaces are selectable using the Space: pulldown menu in the Memory view, and the Memory space pulldown menu in the Disassembly view (see the *SoC Designer Plus User Guide* for more information). Each memory space represents a different view of memory using a page table.

1.5.4 Disassembly View

The Cycle Model supports a disassembly view of a program running on the Cortex-R4 Cycle Model in SoC Designer Simulator. To display the disassembly view in the SoC Designer Simulator, right-click on the Cortex-R4 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 via the Debug menu in the SoC Designer Simulator. Both hardware and software based profiling is available.

1.6.1 Hardware Profiling

Hardware profiling is broken down into eight streams per processor core. They are the Application, I-Cache, D-Cache, Memory, Interrupts, TCM, Pipeline, and AXI streams. The buckets supported by each of these streams are shown in Table 1-15:

Table 1-15 Cortex-R4 Profiling Events

Stream	Buckets	X axis	Y axis
Application	Software Increment	Cycle	Application
	Change to Context ID		
	Software change to PC		
	Divide instruction executed		
	PLD instruction initiates linefill		
	PLD instruction fails to generate Linefill		
	Double precision floating point operation executed		
	Processor livelock due to hard errors or exception at exception vector		

Table 1-15 Cortex-R4 Profiling Events (continued)

Stream	Buckets	X axis	Y axis
I-Cache	ICache Linefill	Cycle	Instruction Cache
	Instruction cache tag RAM parity/ECC error (correctable)		
	Instruction cache data RAM parity/ECC error (correctable)		
	ICache Access		
D-Cache	DCache Linefill	Cycle	Data Cache
	DCache Access		
	Data Read Operations		
	Data Write Operations		
	DCache eviction writeback		
	Data cache tag or dirty RAM parity or correctable ECC error		
	Data cache data RAM parity or correctable ECC error		
	Store buffer Merge		
	Data cache data RAM fatal ECC error		
	Data cache tag/dirty RAM fatal ECC error		
Memory	Unaligned Load/Store	Cycle	Memory
	External memory request		
	Data Memory Barrier		
	Non-Cacheable external request		
Interrupts	CSPR FIQ Masked	Cycle	Interrupts
	CSPR IRQ Masked		
TCM	ITCM error	Cycle	TCM
	DTCM error		
	ATCM parity or multi-bit ECC error		
	B0TCM parity or multi-bit ECC error		
	B1TCM parity or multi-bit ECC error		
	ATCM single-bit ECC error		
	B0TCM single-bit ECC error		
	B1TCM single-bit ECC error		
	LSU TCM single-bit ECC error		
	PFU TCM single-bit ECC error		

Table 1-15 Cortex-R4 Profiling Events (continued)

Stream	Buckets	X axis	Y axis
Pipeline	Instructions executed	Cycle	Pipeline
	Exceptions taken		
	Exception returns		
	Dual issue case A (branch)		
	Dual issue case B1, B2, F2 (load/store), F2D		
	Dual issue other		
	Dual issued pair of instructions architecturally executed		
	Direct branch		
	Predicted return operations		
	Branch mispredicted		
	Branch predictable		
	Stall caused by divide		
	Stall IBuf		
	Stall Ilock		
	Stall LSU		
	Stall buffer stall		
	Stall queue stall		
AXI	TCM Parity or multi-bit error	Cycle	AXI
	TCM single-bit ECC error		

1.6.2 Software Profiling

Software-based profiling is provided by SoC Designer Plus. Profiling information is also available in the SoC Designer Profiler. See the user guide for SoC Designer Plus 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**

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.