DynamIQ Cortex-A75 Cycle Model

Version 9.2.0

User Guide

Non-Confidential

ARM®

100993_0902_00 *(ID052417)*

Cortex-A75 Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
May 2017	0902-00	Non-Confidential	Release for v9.2.0

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

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 your 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 publication provides information that relates directly to SoC Designer:

- *SoC Designer User Guide*

The following publications provide reference information about ARM® products:

- *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*
- *AMBA AXI and ACE Protocol Specification, Issue E*
- *Large Physical Address Extensions Specification* (ARM Architecture Group)

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 from an RTL design. The Cycle Model contains a cycle- and register-accurate Cycle 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	A 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:

1.1 Cycle Model Functionality

In the multiprocessor configuration, up to eight processors are available in a cache-coherent cluster, under the control of a Snoop Control Unit (SCU), which maintains L1, L2 and L3 data cache coherency.

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:

- Cortex-A75 Cycle Model functionality
- Restrictions and Limitations
- Features Additional to the Hardware

1.1.1 Cortex-A75 Cycle Model functionality

This section describes:

- Supported Hardware Features
- Unsupported Hardware Features

1.1.1.1 Supported Hardware Features

The Cortex-A75 Cycle Model supports:

- Up to four processors.
- The AArch32 and AArch64 versions of the ARMv8.2-A architecture instruction set.
- Harvard Level 1 (L1) memory system with a Memory Management Unit (MMU).
- Level 2 (L2) and Level 3 (L3) memory system providing multiprocessor memory coherency.
- Per-core Generic Interrupt Controllers (GICs).
- A generic 64-bit timer per processor.
- Support for AMBA 4.0 AXI Coherency Extension (ACE).
- VFP Floating Point.
- Neon Advanced SIMD.
- Cryptography Engine.
- Optional AMBA 4.0 AXI Peripheral port.
- Hardware profiling.

1.1.1.2 Unsupported Hardware Features

The following features of the hardware are *not* implemented in the Cortex-A75 Cycle Model:

- CHI bus interface.
- Master data width can not be set to 256.
- ACP.
- SCU cache protection.
- Legacy v7 debug map.
- Embedded Logic Analyzer (ELA).
- ARM Processor Optimization Pack (POP) RAMs.
- CPU cache protection.
- L2 cache is not configurable per-core. All cores must be set to the same value.
- The registers listed in "Register Information" on page 27 are available to be read/written via debug transactions — for example, in the SoC Designer Registers window.

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

1.1.2 Restrictions and Limitations

Be aware of the following limitations with this release of your Cycle Model:

- Semihosting is not supported.
- Instruction single-step is not supported.
- Software profiling is not supported.
- Memory views do not include cache content.

1.1.3 Features Additional to the Hardware

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

- Support for positive- and negative-level *irq*, *virq*, *fiq,* and *vfiq* signals. This is configurable using the *negLogic* parameter (see Table 1-3 on page 20).
- 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 46 for more information.
- Waveform dumping using the waveform-related parameters described in Table 1-3 on page 20.
- Support for viewing memory spaces (see “Memory Information” on page 46).
- Support for viewing disassembly data (see “Disassembly View” on page 46).

1.2 Adding and Configuring the SoC Designer Component

The *SoC Designer User Guide* (ARM 100996) describes how to use the component. See that guide 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 in Table 1-1 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, use SoC Designer Canvas.

For more information on SoC Designer Canvas, see the SoC Designer *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 (see “Available Component ESL Ports” on page 17).

1.3 ESL Ports

This section describes the differences between the hardware pins and those on the Cycle Model. Certain hardware pins have been converted to init-time Cycle Model parameters.

- Available Component ESL Ports
- Reset Behavior and Ports
- Tied Pins
- Port Name and Polarity Differences

1.3.1 Available Component ESL Ports

Table 1-2 describes the Cycle Model transactors and special pins that are exposed in SoC Designer. See the corresponding *Technical Reference Manual* for information.

Note: Most ESL component port values can be set using a component parameter. In these cases, the parameter value is 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	Type
ACE5_Master_M0	ACE Master S2T when configured in ACE mode.	Transactor Master
ACE_Lite_ACE5Lite_Master_Peripheral	Master S2T when configured in ACE Lite mode.	Transactor Master
ATCLK	The clock for the ATB trace bus output from the cluster.	Clock Slave
AXI4Stream_master_PROCESSOR	AXI4 Stream Master Inteface.	Transactor Master
AXI4Stream_slave_DISTRIBUTOR	AXI4 Stream Slave Interface.	Transactor Slave
CORECLK[0-3]	The per-core clock for all CPU logic including Level 1 and Level 2 caches.	Clock Slave
extSemi_<*model*>_cpu0	Semihosting is not supported with this release of your Cycle Model.	Transactor Master
extSemi_<*model*>_cpu1		Transactor Master
extSemi_<*model*>_cpu2		Transactor Master
extSemi_<*model*>_cpu3		Transactor Master
GICCLK	Clock for the GIC interface.	Clock Slave
PCLK	Clock for the debug APB interface.	Clock Slave

Table 1-2 ESL Component Ports (continued)

ESL Port	Description	Type
PERIPHCLK	Clock for the timers, power management, and other miscellaneous logic.	Clock Slave
SCLK	Clock for the SCU/L3 and the AMBA interface.	Clock Slave
clk-in	This port is used internally. Leave unconnected.	Clock Slave

1.3.2 Reset Behavior and Ports

The Cycle Model is reset internally each time SoC Designer Simulator is initialized. This behavior is standard and can not be changed. To view the internal reset sequence, set the *Align Waveforms* parameter to False (see Setting Component Parameters), and this data appears in the waveform.

At simulation time zero and while simulation is running, you can generate a reset sequence. To do so, drive the reset pins on the component using external signals (for example, using the MxSigDriver component).

For information about reset pin names, bit ordering (for multiple cores), and required reset sequence, refer to the *Technical Reference Manual* for your IP.

1.3.3 Tied Pins

Certain pins in the hardware have been either tied or disconnected in the Cycle Model for performance reasons. These include:

- DBGEN (tied high)
- DFTCGEN: (tied low)
- DFTRSTDISABLE tied to 00 (2 bits)
- DFTRAMHOLD: (tied low)
- DFTMCPHOLD: (tied low)
- DFTCORECLKDISABLE: (tied low)
- NIDEN (tied high)
- SPIDEN (tied high)
- SPNIDEN (tied high)
- nMBISTRESET (tied high)
- MBISTREQ (tied low)

1.3.4 Port Name and Polarity Differences

The following ports are named differently than their counterparts in the RTL; ports in the RTL that begin with the prefix nCNT* are called CNT* in the Cycle Model:

- nCNTHPIRQ
- nCNTPNSIRQ
- nCNTPSIRQ
- nCNTVIRQ
- nCNTHVIRQ

In addition, these ports are Active Low in the RTL, but Active High in the Cycle Model. You can use the Cycle Model parameter negLogic to invert the polarity of these ports if necessary.

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.

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 component parameters are described in Table 1-3.

Table 1-3 Component Parameters

Name	Description	Allowed Values	Default Value	Init/ Runtime
AA64nAA32	Determines whether processor boots in 32-bit or 64-bit mode.	0 – Boots processor in 32-bit mode. 1 – Boots processor in 64-bit mode.	0	Init
ACE5_Master_M0 Enable Debug Messages	Enable or disable ACE_master port debug.	true, false	false	Runtime
ACE5_Master_M0 Protocol Variant	Variant of the ACE protocol to use on ACE_master_M0,	ACE	ACE	Init
ACE_Lite_ACE5_Lite_ Master_Peripheral Enable Debug Messages	Enables port debug on ACE_Lite_ACE5_Lite_Master	true, false	false	Runtime
ACE_Lite_ACE5_Lite_ Master_Peripheral Protocol Variant	Variant of the ACE protocol to use on ACE_Lite_ACE5_Lite_Master_Peripheral.	ACE_Lite	ACE_Lite	Init
ACLKENM	ACE Master Input Clock Enable.	0, 1	1	Runtime
ACLKENMP	AXI Master peripheral port enable.	0, 1	1	Runtime
ACVMIDEXTM0[3:0]	Additional VMID bits for DVM messages.	0x0 - 0xF	0	Runtime
ACWAKEUPM0	ACE snoop activity indicator.	0, 1	0	Runtime
AENDMP	End address for peripheral port address range.	0x100000 - 0x7FFFFFFFFF	0xd3100000	Init
AFVALIDMx	Fifo flush request. This signal is part of the ATB interface.	0, 1	0	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
Align Waveforms	When set to *true*, waveforms dumped by 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 SoC Designer time.	true, false	true	Init
ASTARTMP	Start address for peripheral port address range.	0x100000 - 0x7FFFFFFFFF	0xd3000000	Init
ATCLKQREQn	Indicates that the clock controller wants to gate the clock (active low). This signal is synchronised into the ATCLK domain before use.	0, 1	true	Init
ATREADYMx	ATB device ready.	0, 1	0	Runtime
AXI4Stream_master_ PROCESSOR Enable Debug Messages	Enables port debug on AXI4Stream_master_PROCESSOR.	true, false	false	Runtime
AXI4Stream_slave_ DISTRIBUTOR Enable Debug Messages	Enables port debug on AXI4Stream_slave_DISTRIBUTOR.	true, false	false	Runtime
BROADCASTCACHE-MAINT	Enable broadcasting of cache maintenance operations to downstream caches.	0 — Cache maintenance operations are not broadcast to downstream caches 1 — Cache maintenance operations are broadcast to downstream caches.	False	Init
BROADCASTCACHE-MAINTPOU	Enable broadcasting of cache maintenance operations to the point of unification.	0 — Cache maintenance operations DCCMVAU and DC CVAU are not broadcast to other clusters. This is more efficient if all other clusters are ARM Cortex processors. 1 = Cache maintenance operations DCCMVAU and DC CVAU are broadcast to other clusters.	False	Init

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
BROADCASTOUTER	Enable broadcasting of Shareable transactions.	0 — Shareable transactions are not broadcast externally. 1 — Shareable transactions are broadcast externally.	False	Init
BROADCASTPERSIST	Enable broadcasting of cache clean to the point of persistence operations.	0 — DC CVAP instructions are treated the same as DC CVAC 1 — DC CVVAP instructions sent as a clean to the point of persistence transaction externally.	False	Init
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	Init
CFGEND	Endianess configuration. 1-bit wide for UP, 4 bits wide for MP. Automatically kept in sync with CFGEND*n*.	integer	0	Init
CFGTE	Default exception handling state (ARM/Thumb). 1-bit wide for UP, 4 bits wide for MP. Automatically kept in sync with CFGTE*n*.	integer	0	Init
CLUSTERIDAFF2	Value read in ClusterID Affinity Level-2 field, MPIDR bits[23:16].	0x0 - 0x7F	0	Init
CLUSTERIDAFF3	Value read in ClusterID Affinity Level-3 field, MPIDR bits[39:32].	0x0 - 0x7F	0	Init
CLUSTERPREQ	Indicates that the power controller wants the cluster to move to a new power state. This signal is synchronised into multiple clock domains before use.	0, 1	0	Runtime
CLUSTERPSTATE	Power state that the power controller wishes the cluster to move to. This signal is synchronised into the SCLK domain before use.	0x0 - 0x3F	0x48	Init
CNTCLKEN	Counter clock enable. This clock enable must be inserted one cycle before the CNTVALUEB bus.	0, 1	1	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
COREPREQ*x*	Indicates that the power controller wants the referenced CPU to move to a new power state. This signal is synchronised into the PERIPHCLK domain before use.	0, 1	0	Runtime
COREPSTATE*x*	Power state that the power controller wishes the CPU to move to. This signal is synchronised into the PERIPHCLK domain before use.	0x0 - 0x1F	0x8	Init
CRYPTODISABLE	Disables the Cryptographic Extensions.	true, false	false	Init
DBGCONNECTED	A debugger is connected and so the DebugBlock should be accessed on boot. This signal is synchronised into the PCLK domain before use.	0, 1	0	Runtime
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	bool	false	Runtime
Enable Debug Messages	Whether debug messages are logged for the component.	bool	false	Runtime
Enable Fast Application Load	Controls fast debug access for application load. “True” setting loads multiple bytes at a time; “False” setting loads 1 byte at a time.	true/false	true	Init
EVENTIREQ	Event input request for processor wake-up from WFE state.	true, false	false	Runtime
EVENTOACK	Event output request acknowledge.	true, false	false	Runtime
Fast Application Load Support	Identifies that the component supports fast debug access for application load.	Not configurable	Yes	N/A
GICCDISABLE	Disables the GIC CPU interface logic and routes the legacy nIRQ, nFIQ, nVIRQ, and nVFIQ. Required to enable use of non-ARM interrupt controllers.	0, 1	0	Init
GICCLKQREQn	Indicates that the clock controller wants the gate the clock (active low). This signal is synchronised into the GICCLK domain before use.	true, false	true	Init
IRITWAKEUP	AXI4 stream protocol activity indicator.	0, 1	0	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
negLogic	If set to 1, inverts the values of the following ports (these ports are Active High by default): • fiq • irq • virq • vfiq • CNTHPIRQ • CNTPNSIRQ • CNTPSIRQ • CNTVIRQ	bool	false	Runtime
PADDRDC	APB address.	0x4 – 0x7FFFF	0	Runtime
PCLKQREQn	Indicates that the clock controller wants the gate the clock (active low). This signal is synchronised into the PCLK domain before use.	true, false	true	Init
PENABLEDC	APB transfer complete flag.	0, 1	0	Runtime
PMUSNAPSHOTREQ	Request for a snapshot of the PMU counters. This signal is synchronised into the PERIPHCLK domain before use.	0, 1	0	Runtime
PREADYCD	APB slave ready, used to extend a transfer.	0, 1	0	Runtime
PSELDC	DebugBlock to Cluster bus access.	0, 1	0	Runtime
PSLVERRCD	APB slave transfer error.	0, 1	0	Runtime
PWAKEUPDC	APB activity indicator.	0, 1	0	Runtime
PWDATADC	APB write data.	0x0 – 0xFFFF	0	Runtime
PWRITEDC	APB read/write indicator.	0x0 – 0xFFFF	0	Runtime
RREADYS	Read data ready.	0, 1	0	Runtime
RVBARADDR[0-3]	Reset Vector Base Address for executing in 64-bit state. Width depends on the widest PA of connected cores.	integer	0	Init
SCLKQREQn	Indicates that the clock controller wants the gate the clock (active low). This signal is synchronised into the SCLK domain before use.	true, false	true	Init
SYNCREQM[0-3]	ETM Signal. Synchronization request from trace sink.	0, 1	0	Runtime
SYSCOACKM0	The system may send snoops to the cluster.	0, 1	0	Runtime

Table 1-3 Component Parameters (continued)

Name	Description	Allowed Values	Default Value	Init/ Runtime
TSCLKEN	Timestamp clock enable. This clock enable is pipelined inside the processor and must be inserted one cycle before the TSVALUEB bus.	0, 1	0	Runtime
TSVALUEB	Timestamp in binary encoding.	0x0 – 0xFFFFFFFF	0	Runtime
VINITHI	Use high vector addresses. 1-bit wide for UP, 4 bits wide for MP. Automatically kept in sync with VINITHI*n*.	integer	0	Init
Waveform File[1]	Name of the waveform file.	*string*	arm_cm_<cycle_mode>.vcd	Init
Waveform Format	The format of the waveform dump file.	VCD, FSDB	VCD	Init
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	Init

1. 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 Cycle Model has a debug interface (CADI) that allows the user to view, manipulate, and control the registers and memory. You can access a view in SoC Designer 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

This section describes the registers supported with this release. For detailed descriptions of these registers, refer to the *ARM Technical Reference Manual* for your IP, or the *ARM Architecture Reference Manual ARMv8 (*for ARMv8-A architecture profile).

Note: Registers are accurate only at debuggable points. While SoC Designer grays out the register view when the processor is not at a debuggable point, values are still visible. Due to the speculative nature of the processor pipeline, these values are not guaranteed to be accurate.

In general, you can write to a register only at a debuggable point. If a value is deposited at any other point, it may not be correctly propagated.

This section includes the following subsections:

- Supported Cortex-A75 Registers

1.5.1.1 Supported Cortex-A75 Registers

Debug register writes to the following register groups, through the register view GUI or through CADI, are not supported in the following cases:

- All General Purpose registers
- SP_ELx (x can be 0-3 depending on the execution state)
- ELR_ELx (x can be 0-3 depending on the execution state)
- DLR_EL0

If debug register writes are not supported, an error message is displayed when a write is attempted. Other registers in the group may be writeable at that time.

To resolve this issue, stop at a point in the software where there is no implicit relationship between the register being written to and any other general purpose register. A relationship between registers may fit one of the following cases:

- Case 1: Registers deriving values from each other without change to either one. For example, stopping at `mov x0, x1` does not allow you to write to x0 or x1. However, if the next instruction is to add `x0, x0, #4`, and you stop after this instruction, then the relationship is broken and you can `debug write` to either location.
- Case 2: Registers not being initialized. At such a point, all registers are related to the same default value and therefore are related to each other. Once a register has been given some value, you can `debug write` to it, except in cases that violate Case 1.

This section describes the following registers:

- Cortex-A75 AArch64_Core Registers
- Cortex-A75 AArch64 System Registers
- Cortex-A75 AArch64 PERF Registers
- Cortex-A75 AArch64 RAS Registers
- Cortex-A75 Cluster AArch64 Registers

- Cortex-A75 AArch32_Core Registers
- Cortex-A75 AArch32 VFP/NEON Registers
- Cortex-A75 AArch32 Secure World Registers
- Cortex-A75 AArch32 Normal World Registers
- Cortex-A75 AArch32 ID Registers
- Cortex-A75 AArch32 Control Registers
- Cortex-A75 AArch32 VA to PA Registers
- Cortex-A75 AArch32 PERF Registers
- Cortex-A75 Cluster AArch32 Registers
- Cortex-A75 Cluster AArch32 PMU Registers
- Cortex-A75 Cluster AArch64 PMU Registers
- Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH32 Registers
- Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH64 Registers
- Cortex-A75 GICv3 CPU Interface Hypervisor AARCH32 Registers
- Cortex-A75 GICv3 CPU Interface Hypervisor AARCH64 Registers
- Cortex-A75 Internal State Registers

Cortex-A75 AArch64_Core Registers

Table 1-4 Cortex-A75 AArch64_Core Registers

Name	Access
CurrentEL	Read/Write
DAIF	Read/Write
SPSR_abt	Read/Write
SPSR_fiq	Read/Write
SPSR_irq	Read/Write
SPSR_und	Read/Write
SPSel	Read Only
X0-X30	Read/Write
PC	Read/Write
SP_EL[0-3]	Read/Write
SP	Read/Write
ELR_EL[1-3]	Read/Write
ELR_zero	Read Only
ELR	Read/Write
CPSR_nodbg	Read/Write
DSPSR	Read/Write
CPSR	Read/Write
SPSR_EL[1-3]	Read/Write
SPSR	Read/Write
PC_MEMSPACE	Read Only

Cortex-A75 AArch64 System Registers

Table 1-5 Cortex-A75 AArch64 System Registers

Name	Access
ACTLR_EL1	Read Only
ACTLR_EL[2,3]	Read/Write
AFSR[0,1]_EL[1-3]	Read Only
AIDR_EL1	Read Only
AMAIR_EL1	Read Only
CCSIDR_EL1	Read Only
CLIDR_EL1	Read Only
CNTKCTL_EL1	Read/Write
CNTVOFF_EL2	Read/Write

Table 1-5 Cortex-A75 AArch64 System Registers (continued)

Name	Access
CONTEXTIDR_EL[1,2]	Read/Write
CPACR_EL1	Read/Write
CSSELR_EL1	Read Only
CTR_EL0	Read Only
DACR32_EL2	Read/Write
DCZID_EL0	Read Only
ESR_EL[1-3]	Read/Write
FAR_EL[1-3]	Read/Write
HACR_EL2	Read Only
HCR_EL2	Read/Write
HPFAR_EL2	Read/Write
ID_AA64AFR[0,1]_EL1	Read Only
ID_AA64DFR[0,1]_EL1	Read Only
ID_AA64ISAR[0,1]_EL1	Read Only
ID_AA64MMFR[0-2]_EL1	Read Only
ID_AA64PFR[0,1]_EL1	Read Only
ID_AFR0_EL1	Read Only
ID_DFR0_EL1	Read Only
ID_ISAR[0-5]_EL1	Read Only
ID_MMFR[0-4]_EL1	Read Only
ID_PFR[0,1]_EL1	Read Only
ISR_EL1	Read/Write
MAIR_EL[1-3]	Read/Write
MIDR_EL1	Read Only
MPIDR_EL1	Read Only
MVFR[0-2]_EL1	Read Only
PAR_EL1	Read/Write
REVIDR_EL1	Read Only
RMR_EL3	Read/Write
RVBAR_EL3	Read Only
SCR_EL3	Read/Write
SCTLR_EL[1-3]	Read/Write
TCR_EL[1-3]	Read/Write
TPIDRRO_EL0	Read/Write

Table 1-5 Cortex-A75 AArch64 System Registers (continued)

Name	Access
TPIDR_EL[0-3]	Read/Write
TTBR[0,1]_EL[1-3]	Read/Write
VBAR_EL[1-3]	Read/Write
VMPIDR_EL2	Read/Write
VPIDR_EL2	Read/Write
VTCR_EL2	Read/Write
VTTBR_EL2	Read/Write

Cortex-A75 AArch64 PERF Registers

Table 1-6 Cortex-A75 AArch64 PERF Registers

Name	Access
PMCEID[0,1]_EL0	Read Only
PMCR_EL0	Read/Write
PMCNTENSET_EL0	Read/Write
PMCNTENCLR_EL0	Read/Write
PMOVSSET_EL0	Read/Write
PMOVSCLR_EL0	Read/Write
PMSELR_EL0	Read/Write
PMCCNTR_EL0	Read/Write
PMEVTYPER[0-5]_EL0	Read/Write
PMCCFILTR_EL0	Read/Write
PMXEVTYPER_EL0	Read/Write
PMEVCNTR[0-5]_EL0	Read/Write
PMXEVCNTR_EL0	Read/Write
PMUSERENR_EL0	Read/Write
PMINTENSET_EL1	Read/Write
PMINTENCLR_EL1	Read/Write

Cortex-A75 AArch64 RAS Registers

Table 1-7 Cortex-A75 AArch64 RAS Registers

Name	Access
DISR_EL1	Read/Write
VDISR_EL2	Read/Write
VSESR_EL2	Read/Write

Cortex-A75 Cluster AArch64 Registers

Table 1-8 Cortex-A75 Cluster AArch64 Registers

Name	Access
CLUSTERACPSID_EL1	Read/Write
CLUSTERACTLR_EL1	Read/Write
CLUSTERBUSQOS_EL1	Read/Write
CLUSTERCFR_EL1	Read Only
CLUSTERIDR_EL1	Read/Write
CLUSTERL3HIT_EL1	Read/Write
CLUSTERL3MISS_EL1	Read/Write
CLUSTERPARTCR_EL1	Read/Write
CLUSTERPWRCTLR_EL1	Read/Write
CLUSTERPWRDN_EL1	Read Only
CLUSTERPWRSTAT_EL1	Read/Write
CLUSTERREVIDR_EL1	Read/Write
CLUSTERSTASHSID_EL1	Read/Write
CLUSTERTHREADSID_EL1	Read Only

Cortex-A75 AArch32_Core Registers

Table 1-9 Cortex-A75 AArch32_Core Registers

Name	Access
SPSR	Read/Write
R0-R14	Read/Write
R[8-14]_fiq	Read/Write
R[8-14]_usr	Read/Write
R[13, 14]_irq	Read/Write
R[13, 14]_svc	Read/Write
R[13, 14]_und	Read/Write
R[13, 14]_abt	Read/Write
R[13, 14]_mon	Read/Write
R13_hyp	Read/Write
R15	Read/Write
DSPSR32	Read/Write
CPSR32	Read/Write
CPSR_mode	Read Only
SPSR _svc	Read/Write
SPSR_irq	Read/Write
SPSR_fiq	Read/Write
SPSR_und	Read/Write
SPSR_abt	Read/Write
SPSR_mon	Read/Write
SPSR_hyp	Read/Write
SPSR32	Read/Write

Cortex-A75 AArch32 VFP/NEON Registers

Table 1-10 Cortex-A75 AArch32 VFP/NEON Registers

Name	Access
FPSID	Read Only
MVFR[0,1]	Read Only

Cortex-A75 AArch32 Secure World Registers

Table 1-11 Cortex-A75 AArch32 Secure World Registers

Name	Access
S_ADFSR	Read Only
S_AIFSR	Read Only
S_CONTEXTIDR	Read/Write
S_CSSELR	Read/Write
S_DACR	Read/Write
S_DFAR	Read/Write
S_DFSR	Read/Write
S_ICC_CTLR	Read/Write
S_ICC_SRE	Read/Write
S_IFAR	Read/Write
S_IFSR	Read/Write
MVBAR	Read/Write
S_NMRR	Read/Write
S_PAR	Read/Write
S_PRRR	Read/Write
SCR	Read/Write
SDER	Read/Write
S_TPIDRPRW	Read/Write
S_TPIDRURO	Read/Write
S_TPIDRURW	Read/Write
S_TTBCR	Read/Write
S_VBAR	Read/Write
S_SCTLR	Read/Write
S_ACTLR	Read/Write
S_TTBR0	Read/Write
S_TTBR1	Read/Write
S_TTBR0_64	Read/Write
S_TTBR1_64	Read/Write

Cortex-A75 AArch32 Normal World Registers

Table 1-12 Cortex-A75 AArch32 Normal World Registers

Name	Access
N_ADFSR	Read Only
N_AIFSR	Read Only
N_CONTEXTIDR	Read/Write
N_CSSELR	Read/Write
N_DACR	Read/Write
N_DFAR	Read/Write
N_DFSR	Read/Write
N_ICC_CTLR	Read/Write
N_ICC_SRE	Read/Write
N_IFAR	Read/Write
N_IFSR	Read/Write
N_NMRR	Read/Write
N_PAR	Read/Write
N_PRRR	Read/Write
N_TPIDRPRW	Read/Write
N_TPIDRURO	Read/Write
N_TPIDRURW	Read/Write
N_TTBCR	Read/Write
N_VBAR	Read/Write
N_SCTLR	Read/Write
N_ACTLR	Read/Write
N_TTBR0	Read/Write
N_TTBR1	Read/Write
N_TTBR0_64	Read/Write
N_TTBR1_64	Read/Write

Cortex-A75 AArch32 ID Registers

Table 1-13 Cortex-A75 AArch32 ID Registers

Name	Access
CSSELR	Read/Write
MIDR	Read Only
CTR	Read Only
TCMTR	Read Only
TLBTR	Read Only
MPIDR	Read Only
REVIDR	Read Only
ID_PFR[0,1]	Read Only
ID_DFR0	Read Only
ID_AFR0	Read Only
ID_MMFR[0-4]	Read Only
ID_ISAR[0-5	Read Only
CSSIDR	Read Only
CLIDR	Read Only
AIDR	Read Only
JIDR	Read Only
JOSCR	Read Only
JMCR	Read Only

Cortex-A75 AArch32 Control Registers

Table 1-14 Cortex-A75 AArch32 Control Registers

Name	Access
ACTLR	Read/Write
ACTLR2	Read Only
ADFSR	Read/Write
AIFSR	Read/Write
CNTFRQ	Read/Write
CNTKCTL	Read/Write
CNTVOFF	Read/Write
CNTV_CVAL	Read/Write
CONTEXTIDR	Read/Write
CPACR	Read/Write
DACR	Read/Write
DFSR	Read/Write
FCSEIDR	Read Only
HACTLR	Read/Write
HADFSR	Read Only
HAIFSR	Read Only
HCR	Read/Write
HDCR	Read/Write
HDFAR	Read/Write
HIFAR	Read/Write
HMAIR0	Read/Write
HMAIR1	Read/Write
HPFAR	Read/Write
HSCTLR	Read/Write
HSR	Read/Write
HTPIDR	Read/Write
HTCR	Read/Write
HTTBR	Read/Write
HVBAR	Read/Write
ISR	Read/Write
NSACR	Read/Write
NMRR	Read/Write
PRRR	Read/Write

Table 1-14 Cortex-A75 AArch32 Control Registers (continued)

Name	Access
SCR	Read/Write
SCTLR	Read/Write
RVBAR	Read Only
TPIDRPRW	Read/Write
TPIDRURO	Read/Write
TPIDRURW	Read/Write
TTBCR	Read/Write
TTBR[0,1]	Read/Write
TTBR[0,1]_64	Read/Write
VBAR	Read/Write
VPIDR	Read/Write
VTCR	Read/Write
VTTBR	Read/Write

Cortex-A75 AArch32 VA to PA Registers

Table 1-15 Cortex-A75 AArch32 VA to PA Registers

Name	Access
PAR	Read/Write

Cortex-A75 AArch32 PERF Registers

Table 1-16 Cortex-A75 AArch32 PERF Registers

Name	Access
PMCEID[0-3]	Read Only
PMCNTENCLR	Read/Write
PMCNTENSET	Read/Write
PMCR	Read/Write
PMINTENCLR	Read/Write
PMINTENSET	Read/Write
PMOVSR	Read/Write
PMOVSSET	Read/Write
PMSELR	Read/Write
PMUSERENR	Read/Write

Cortex-A75 Cluster AArch32 Registers

Table 1-17 Cortex-A75 Cluster AArch32 Registers

Name	Access
CLUSTERACPSID	Read/Write
CLUSTERACTLR	Read/Write
CLUSTERBUSQOS	Read/Write
CLUSTERCFR	Read/Write
CLUSTERIDR	Read/Write
CLUSTERL3HIT	Read/Write
CLUSTERL3MISS	Read/Write
CLUSTERPARTCR	Read/Write
CLUSTERPWRCTLR	Read/Write
CLUSTERPWRDN	Read/Write
CLUSTERPWRSTAT	Read/Write
CLUSTERREVIDR	Read/Write
CLUSTERSTASHSID	Read/Write
CLUSTERTHREADSID	Read/Write

Cortex-A75 Cluster AArch32 PMU Registers

Table 1-18 Cortex-A75 Cluster AArch32 PMU Registers

Name	Access
CLUSTERPMCCNTR	Read/Write
CLUSTERPMCEID0	Read/Write
CLUSTERPMCEID1	Read/Write
CLUSTERPMCNTENSET	Read/Write
CLUSTERPMCR	Read/Write
CLUSTERPMOVSSET	Read/Write
CLUSTERPMXEVCNTR	Read/Write
CLUSTERPMXEVTYPER	Read/Write
PMEVCNTR0	Read/Write
PMEVCNTR1	Read/Write
PMEVCNTR2	Read/Write
PMEVCNTR3	Read/Write
PMEVCNTR4	Read/Write
PMEVCNTR5	Read/Write
CLUSTERPMMDCR	Read/Write

Cortex-A75 Cluster AArch64 PMU Registers

Table 1-19 Cortex-A75 Cluster AArch64 PMU Registers

Name	Access
CLUSTERPMCCNTR_EL1	Read/Write
CLUSTERPMCEID0_EL1	Read Only
CLUSTERPMCEID1_EL1	Read Only
CLUSTERPMCNTENSET_EL1	Read/Write
CLUSTERPMCR_EL1	Read/Write
CLUSTERPMMDCR_EL3	Read/Write
CLUSTERPMXEVCNTR_EL1	Read Only
CLUSTERPMXEVTYPER_EL1	Read Only
PMEVCNTR[0-5]_EL0	Read/Write
CLUSTERPMCR_EL0	Read/Write
CLUSTERPMCNTENSET_EL0	Read/Write
CLUSTERPMCNTENCLR_EL0	Read/Write
CLUSTERPMOVSR_EL0	Read/Write
CLUSTERPMSELR_EL0	Read/Write
CLUSTERPMCEID[0,1]_EL0	Read Only
CLUSTERPMCCNTR_EL0	Read/Write
CLUSTERPMINTENSET_EL1	Read/Write
CLUSTERPMINTENCLR_EL1	Read/Write
CLUSTERPMOVSSET_EL0	Read/Write
CLUSTERPMXEVCNTR_EL0	Read/Write
CLUSTERPMXEVTYPER_EL0	Read/Write
CLUSTERPMEVCNTR[0-5]_EL0	Read/Write
CLUSTERPMEVTYPER[0-5]_EL0	Read/Write

Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH32 Registers

Table 1-20 Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH32 Registers

Name	Access
ICC_AP0R0	Read/Write
ICC_BPR0	Read/Write
ICC_HPPIR0	Read Only
ICC_HPPIR1	Read Only
ICC_IAR0	Read Only
ICC_IAR1	Read Only
ICC_IGRPEN0	Read/Write
ICC_MCTLR	Read/Write
ICC_MGRPEN1	Read/Write
ICC_PMR	Read/Write
ICC_CTLR	Read/Write
ICC_SRE	Read/Write

Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH64 Registers

Table 1-21 Cortex-A75 GICv3 CPU Interface CPU/Virtual AARCH64 Registers

Name	Access
ICC_BPR0_EL1	Read/Write
N_ICC_CTLR_EL1	Read/Write
S_ICC_CTLR_EL1	Read/Write
ICC_CTLR_EL3	Read/Write
ICC_HPPIR0_EL1	Read Only
ICC_HPPIR1_EL1	Read Only
ICC_IAR0_EL1	Read Only
ICC_IAR1_EL1	Read Only
ICC_IGRPEN0_EL1	Read/Write
ICC_IGRPEN1_EL3	Read/Write
ICC_PMR_EL1	Read/Write
N_ICC_SRE_EL1	Read/Write
S_ICC_SRE_EL1	Read/Write
ICC_SRE_EL2	Read/Write
ICV_BPR0_EL1	Read/Write
ICV_BPR1_EL1	Read/Write
ICV_CTLR_EL1	Read/Write
ICV_HPPIR0_EL1	Read Only
ICV_HPPIR1_EL1	Read Only
ICV_IAR0_EL1	Read Only
ICV_IAR1_EL1	Read Only
ICV_IGRPEN0_EL1	Read/Write
ICV_PMR_EL1	Read/Write
ICV_RPR_EL1	Read Only
ICC_CTLR_EL1	Read/Write
ICC_SRE_EL1	Read/Write

Cortex-A75 GICv3 CPU Interface Hypervisor AARCH32 Registers

Table 1-22 Cortex-A75 GICv3 CPU Interface Hypervisor AARCH32 Registers

Name	Access
ICV_BPR0	Read/Write
ICV_BPR1	Read/Write
ICV_CTLR	Read/Write
ICV_HPPIR0	Read Only
ICV_HPPIR1	Read Only
ICV_IAR0	Read Only
ICV_IAR1	Read Only
ICV_IGRPEN0	Read/Write
ICV_IGRPEN1	Read/Write
ICV_PMR	Read/Write
ICV_RPR	Read Only

Cortex-A75 GICv3 CPU Interface Hypervisor AARCH64 Registers

Table 1-23 Cortex-A75 GICv3 CPU Interface Hypervisor AARCH64 Registers

Name	Access
ICH_AP0R0_EL2	Read/Write
ICH_AP1R0_EL2	Read/Write
ICH_EISR_EL2	Read Only
ICH_HCR_EL2	Read/Write
ICH_LR0_EL2	Read/Write
ICH_LR1_EL2	Read/Write
ICH_MISR_EL2	Read/Write
ICH_VMCR_EL2	Read/Write

Cortex-A75 Internal State Registers

Table 1-24 Cortex-A75 Internal State Registers

Name	Access
EL3_REGISTER_WIDTH	Read Only

1.5.2 Run To Debug Point

The “run to debug point” feature has been added to enhance Cycle Model debugging. The 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, 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 causes 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 Simulator. It is also available in the disassembler view.

1.5.3 Memory Information

Each memory space of the Cycle Model represents a different view of memory using a page table. The processor supports two memory spaces, which are selectable using the Space: pull-down menu in the Memory view (see the *SoC Designer User Guide* for more information):

- **Physical Memory (secure)** — Main memory space visible from the Memory view.
- **Secure Monitor** — CPU virtual memory visible from the Memory view.
- **Guest** — Uses the N_TTBR0, N_TTBR1, and N_TTBCR registers to translate from VA to PA. This space is active when (SCR.NS == 1) and (CPSR.M != HYP) and (CPSR.M != MON).

This component supports full system coherent memory views. Refer to the *SoC Designer User Guide* for details.

1.5.4 Disassembly View

SoC Designer Simulator supports a disassembly view of a program running on the Cycle Model. To display the disassembly view in the SoC Designer Simulator, right-click on the Cycle Model and select **View Disassembly…** from the context menu. Refer to the *SoC Designer User Guide* for more information.

1.6 Available Profiling Data

You can view Profiling data using the Profiling Manager, which is accessible via the Debug menu in the SoC Designer Simulator.

1.6.1 Hardware Profiling

Hardware events are uniquely identified by their Event Number as defined in the *Technical Reference Manual* for your IP. The event names that appear in the Profiling Manager view are a concatenation of the event number and a shortened form of the event name. If architecture mnemonics have been defined by ARM then that name has been used; otherwise, a short form of the name has been created.

All PMU events supported by the hardware are supported by the Cycle Model; PMU events are disabled by default. The Cycle Model organizes PMU events into the following Streams:

- Instructions
- Pipeline
- I-Cache
- D-Cache
- L2-Cache
- L3-Cache
- Memory
- Bus
- SCU
- Microarchitecture
- Cycle

Note: PMU events should be enabled only after SYSCOREQ has been acknowledged on the SYSCOACK channel. Enabling the events prior to this may skew your PMU data by counting triggers during the reset sequence.

To review the supported events, in SoC Designer Simulator:

1. Click **Profiling Manager** to access the Profiling Manager dialog.
2. Select the row for the Stream Name whose events you want to review.
3. Click **Display**. The Profiling dialog for that Stream displays, which shows each event included in that stream group.

To customize your view, enable only the events you are interested in before running your simulation. For more information about using the Profiling Manager, see the *SoC Designer User Guide* (ARM 100996).

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.