ARM® Cortex®-A57 MPCore™ Processor

Revision: r0p1

Integration Manual

Confidential

ARM Cortex-A57 MPCore Processor
Integration Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
04 June 2013	A	Confidential	First release for r0p0
01 October 2013	B	Confidential	First release for r0p1

Product Status

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

Web Address

`http://www.arm.com`

Contents

ARM Cortex-A57 MPCore Processor Integration Manual

Preface

This preface introduces the *ARM® Cortex®-A57 MPCore™ Processor Integration Manual*. It contains the following sections:

- *About this document* on page vi.
- *Feedback* on page x.

About this document

This document is for the Cortex-A57 MPCore multiprocessor. It describes the integration process for the multiprocessor.

Implementation obligations

This document is designed to help you implement an ARM product. The extent to which the deliverables can be modified or disclosed is governed by the contract between ARM and the Licensee. There might be validation requirements which, if applicable, are detailed in the contract between ARM and the Licensee and which, if present, must be complied with prior to the distribution of any devices incorporating the technology described in this document. Reproduction of this document is only permitted in accordance with the licenses granted to the Licensee.ARM assumes no liability for your overall system design and performance. Verification procedures defined by ARM are only intended to verify the correct implementation of the technology licensed by ARM, and are not intended to test the functionality or performance of the overall system. You or the Licensee are responsible for performing system level tests.You are responsible for applications that are used in conjunction with the ARM technology described in this document, and to minimize risks, adequate design and operating safeguards must be provided for by you. Publishing information by ARM in this document of information regarding third party products or services is not an express or implied approval or endorsement of the use thereof.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this document, where:

r*n*	Identifies the major revision of the product.
p*n*	Identifies the minor revision or modification status of the product.

Intended audience

This document is written for experienced design and verification engineers who want to integrate the Cortex-A57 MPCore multiprocessor into a system design and have experience of IP-based designs but might have no experience of ARM products.

Using this document

This document is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the integration process and the multiprocessor.

Chapter 2 *Functional Integration Guidelines*

Read this for information about how to integrate the multiprocessor into your SoC design.

Chapter 3 *DFT Integration Guidelines*

Read this for information about the *Design For Test* (DFT) guidelines.

Chapter 4 *Integration Kit*

Read this for a description of the Cortex-A57 MPCore multiprocessor *Integration Kit* (IK).

Chapter 5 *CoreSight Trace and Debug Integration*

Read this for a description about CoreSight trace and debug integration.

Appendix A *Revisions*

Read this for a description of the technical changes in this document.

Glossary

The *ARM® Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM® Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM® Glossary*, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Conventions

This document uses the conventions that are described in:

- *Typographical conventions*.
- *Timing diagrams*.
- *Signals* on page viii.

Typographical conventions

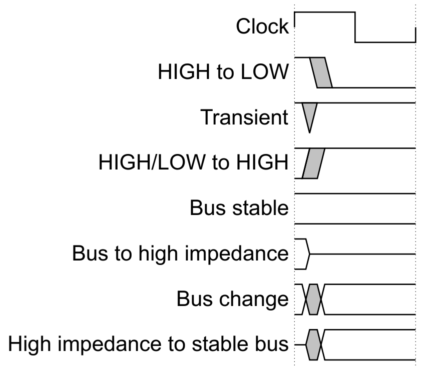
Typographical conventions

italic	Introduces special terminology, denotes cross-references, and citations
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
monospace italic	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, UNDEFINED, and UNKNOWN.

Timing diagrams

The figure named *Key to timing diagram conventions* on page viii explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.

Signals

The signal conventions are:

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

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

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

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, `http://infocenter.arm.com`, for access to ARM documentation.

ARM publications

This document contains information that is specific to this product. See the following documents for other relevant information:

- *ARM® AMBA® AXI™ and ACE™ Protocol Specification* (ARM IHI 0022).
- *ARM® AMBA® APB™ Protocol Specification* (ARM IHI 0024).
- *ARM® AMBA® ATB Protocol Specification* (ARM IHI 0032).
- *ARM® AMBA® 4 AXI4-Stream™ Protocol Specification* (ARM IHI 0051).
- *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).
- *ARM® CoreSight™ Components Technical Reference Manual* (ARM DDI 0314).
- *ARM® CoreSight™ DAP-Lite Technical Reference Manual* (ARM DDI 0316).
- *ARM® CoreSight™ SoC-400 Technical Reference Manual* (ARM DDI 0480).

- *ARM® Embedded Trace Macrocell Architecture Specification ETMv4* (ARM IHI 0064).
- *ARM® Generic Interrupt Controller Architecture Specification* (ARM IHI 0048).

The following confidential books are only available to licensees:

- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *ARM® CoreSight™ SoC-400 User Guide* (ARM DUI 0563).
- ARM® AMBA® 5 CHI Protocol Specification (ARM IHI 0050)
- *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* (ARM DDI 0488).
- *ARM® Cortex®-A57 MPCore™ Processor Cryptography Extension Technical Reference Manual* (ARM DDI 0514).
- *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* (ARM DII 0279).
- *ARM® Cortex®-A57 MPCore™ Processor Release Note*.

Other publications

This section lists relevant documents published by third parties:

- IEEE, *Standard Test Access Port and Boundary Scan Architecture*, Std. 1149.1-2001.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, DII0280B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

This chapter gives an overview of the Cortex-A57 MPCore multiprocessor integration process. It contains the following sections:

1.1 About integrating the delivery

Integration is the final phase in the process for the inclusion of the Cortex-A57 MPCore multiprocessor in your SoC design. You must complete the tasks specified in the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* before integrating the multiprocessor into your SoC design.

Various implications arise for your design during integration. You must consider:

- Connection of the multiprocessor to all necessary peripherals and buses.
- Connection of the required clocks and resets to the multiprocessor.
- Treatment of unused signals and tie-offs.
- *Design For Test* (DFT) strategies for the multiprocessor.
- Physical integration and layout.
- Verification of the multiprocessor within the SoC design.

The main connections in your SoC design are:

- AMBA 4 *AXI Coherency Extensions* (ACE) or CHI master interface.
- AMBA 4 AXI *Accelerator Coherency Port* (ACP) slave interface.
- Debug *Advanced Peripheral Bus* (APB) interface.
- *Generic Interrupt Controller* (GIC) CPU interface.
- *Memory Built-In Self Test* (MBIST) interface.
- CoreSight *Cross Trigger Interface* (CTI).
- AMBA *Advanced Trace Bus* (ATB) interface.

You must also integrate the:

- Clocks and resets.
- Power management signals.
- Debug signals.
- Performance monitor signals.
- Global timestamp input signal.
- Miscellaneous signals such as timers and interrupts.

1.2 Key integration tasks

This section gives a summary of the key integration tasks required to integrate the Cortex-A57 MPCore multiprocessor into your SoC design. Use the information in Table 1-1:

- In conjunction with the rest of this document and the documents in *Additional reading* on page viii.
- As a guide to verify that you cover the integration steps described in subsequent chapters of this document.

The value of *N* in Table 1-1 represents processor 0, 1, 2, or 3 in your design.

Table 1-1 Key integration tasks

Key task	Description
1. Connect clock signals correctly: • **CLK**. • **PCLKDBG**.	See *Clocks* on page 2-3
2. Connect or tie off reset controls correctly: • **nCPUPORESET[N:0]**. • **nCORERESET[N:0]**. • **nL2RESET**. • **L2RSTDISABLE**. • **nPRESETDBG**. • **WARMRSTREQ[N:0].**	See *Resets* on page 2-6
3. Connect or tie off the configuration signals correctly.	See *Configuration signals* on page 2-7
4. Connect or tie off the interface signals correctly: • GIC CPU interface signals. • Generic Timer signals. • Power control signals. • ACE or CHI memory interface. • ACP interface. • Debug interface. • ETM interface. • Cross trigger channel interface signals. • Performance monitoring signals. • DFT interface. • MBIST interface.	See *Interfaces* on page 2-9
5. Verify your design.	See Chapter 4 *Integration Kit*

1.3 Design tools

This document assumes that you have suitable *Electronic Design Automation* (EDA) tools and compute resources for implementation. See the *ARM® Cortex®-A57 MPCore™ Processor Release Note* for a list of any specific tool revisions required.

Chapter 2
Functional Integration Guidelines

This chapter describes the functional integration guidelines of the Cortex-A57 MPCore multiprocessor in your SoC design. It contains the following sections:

2.1 About functional integration

This section describes how to connect the Cortex-A57 MPCore multiprocessor signals in your SoC design. See the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* for more information about the multiprocessor interfaces.

Figure 2-1 shows the main interfaces of the Cortex-A57 MPCore multiprocessor that you must integrate in your SoC design.

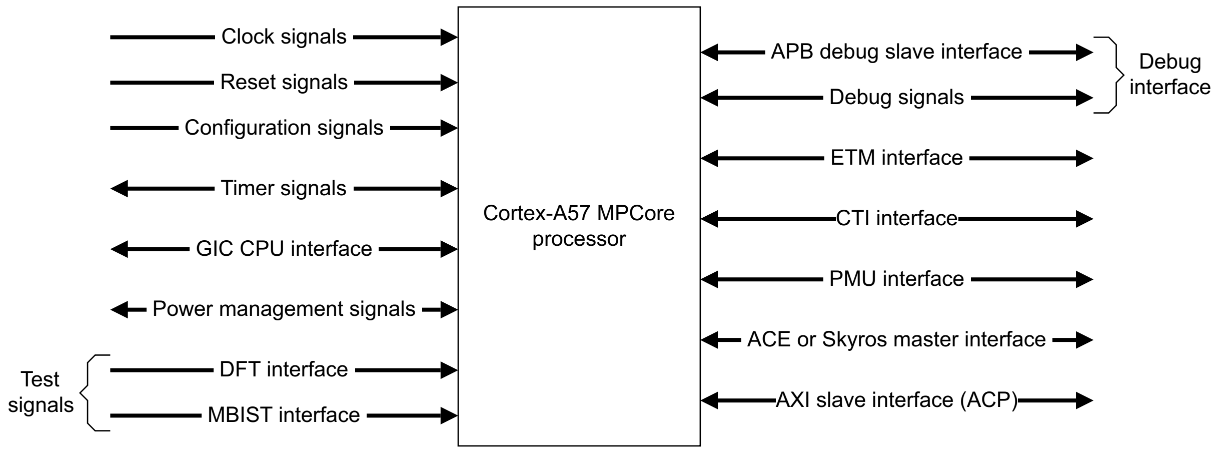


Figure 2-1 Cortex-A57 MPCore multiprocessor interfaces

2.2 Clocks

This section provides information about how to connect the Cortex-A57 MPCore multiprocessor clocks in your SoC design.

Table 2-1 shows the multiprocessor clocks.

Table 2-1 Multiprocessor clocks

Name	Direction	Description	Connection Information
CLKEN	Input	Main clock enable	Connect to your multiprocessor clock control logic
CLK	Input	Main clock	
PCLKDBG	Input	Debug APB clock	Connect to your debug clock control logic

CLK	The main clock of the Cortex-A57 MPCore multiprocessor. All processors, the shared *Level 2* (L2) memory system logic, the *Generic Interrupt Controller* (GIC), and the Generic Timer are clocked with derived versions of **CLK**.
PCLKDBG	The APB clock that controls the Debug APB, CTI and CTM logic in the **PCLKDBG** domain. **PCLKDBG** is asynchronous to **CLK**.

2.2.1 Internal clocks

The **CLKEN** signal is the clock enable for all internal clocks derived from **CLK**.

The relative timing of an internal clock is controlled with its associated clock enable signal. For more information about the internal clocks, see the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual*.

ACE master interface timing

The ACE interface is a synchronous AXI master interface that can operate at any integer multiple that is equal to or slower than the multiprocessor clock, **CLK**, using the **ACLKENM** signal.

CHI interface timing

The CHI interface is a synchronous interface that can operate at any integer multiple that is equal to or slower than the multiprocessor clock, **CLK**, using the **SCLKEN** signal.

ACP slave interface timing

The ACP is a synchronous AXI slave interface that can operate at any integer multiple that is equal to or slower than the multiprocessor clock, **CLK**, using the **ACLKENS** signal.

ATB interface timing

The ATB interface is a synchronous interface that can operate at any integer multiple that is slower than the multiprocessor clock, **CLK**, using the **ATCLKEN** signal.

Debug interface timing

The debug interface is an asynchronous APB slave interface that can operate at any integer multiple that is equal to or slower than the APB clock, **PCLKDBG**, using the **PCLKENDBG** signal.

2.2.2 Synchronization of signals

Input signals must be synchronous to the associated interface clock. See *Internal clocks* on page 2-3 for a description of the interface clocks.

However, the multiprocessor does permit some input signals to be asynchronous to **CLK** or **PCLKDBG** because it provides internal synchronizers. The multiprocessor synchronizes the following input signals to the **CLK** domain:

- **nFIQ[N:0]**.
- **nIRQ[N:0]**.
- **nREI[N:0]**.
- **nSEI[N:0]**.
- **nVFIQ[N:0]**.
- **nVIRQ[N:0]**.
- **nVSEI[N:0]**.
- **EVENTI**.
- **CLREXMONREQ**.
- **CPUQREQn[N:0]**.
- **L2QREQn**.
- **L2FLUSHREQ**.

The multiprocessor synchronizes the following input signals to the **PCLKDBG** domain:

- **CTIIRQACK[N:0]**.
- **EDBGRQ[N:0]**.

The multiprocessor synchronizes the following input signals to the **CLK** and the **PCLKDBG** domains:

- **DBGEN[N:0]**.
- **NIDEN[N:0]**.
- **SPIDEN[N:0]**.
- **SPNIDEN[N:0]**.

The following input signals go to both the **CLK** and **PCLKDBG** domains and the multiprocessor synchronizes them to the **PCLKDBG** domain:

- **CLUSTERIDAFF1[7:0]**.
- **CLUSTERIDAFF2[7:0]**.
- **CRYPTODISABLE[N:0]**.
- **GICCDISABLE**.

——— **Note** ———

CRYPTODISABLE[N:0] only exists if the multiprocessor implements the Cryptography Extension.

————————————

If **CISBYPASS** is tied LOW then the multiprocessor also provides internal synchronizers in the **PCLKDBG** domain for:

- **CTICHIN**.
- **CTICHOUTACK**.

2.3 Resets

Table 2-2 shows the Cortex-A57 MPCore multiprocessor resets. The reset signals enable you to reset different parts of the multiprocessor independently. The value of *N* is one less than the number of processors in your design.

Table 2-2 Multiprocessor resets

Signal	Direction	Description	Connection information
nCPUPORESET[N:0]	Input	Set LOW to initialize all of the logic in processor *N*.	Connect to reset control logic.
nCORERESET[N:0]	Input	Set LOW to initialize the logic in processor *N*, except the Debug, ETM, breakpoint, and watchpoint logic.	
nL2RESET	Input	Set LOW to initialize the shared L2 memory system, Generic Interrupt Controller, and Generic Timer logic.	
L2RSTDISABLE	Input	Disables L2 cache invalidation at Warm reset.	Tie HIGH to disable L2 cache invalidation at soft reset. Tie LOW to invalidate L2 cache at reset. Only change this signal when the multiprocessor is in the reset state.
nPRESETDBG	Input	Set LOW to initialize the shared Debug, ETM, CTI and CTM logic in the **PCLKDBG** domain.	Connect to reset control logic.
WARMRSTREQ[N:0]	Output	Goes HIGH when processor *N* requests a Warm reset. To request a Warm reset, software sets the RR bit to 1 in the Reset Management Register, RMR_EL3 or RMR.	

For more information about the valid reset combinations and the supported reset sequences, see the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual*.

2.4 Configuration signals

Table 2-3 shows the configuration signals and describes how to set these signals in your SoC design. The value of *N* is one less than the number of processors in your design.

Table 2-3 Configuration inputs

Signal	Direction	Description	Connection information
CFGEND[N:0]	Input	Controls the endianness for data exception handling at reset. Sets the value of the SCTLR.EE or SCTLR_EL3.EE bit.	Tie HIGH for big endian data during exception handling. Tie LOW for little endian data during exception handling. Only change this signal when processor *N* is in the reset state.
CFGTE[N:0]	Input	Controls processor state for exception handling at reset. Sets the value of the SCTLR.TE or SCTLR_EL3.TE bit.	If processor *N* boots into AArch32 state then: • Tie HIGH to select T32 exception handling. • Tie LOW to select A32 exception handling. Only change this signal when processor *N* is in the reset state.
VINITHI[N:0]	Input	Sets the base address of the vector table.	If processor *N* boots into AArch32 state then: • Tie HIGH to set the vector table at`0xFFFF0000`. • Tie LOW to set the vector table at `0x00000000`. Only change this signal when processor *N* is in the reset state.
CP15SDISABLE[N:0]	Input	Secure system register write access disable. Controls access to Secure system registers.[a]	Connect to system security control logic. Tie LOW if the system does not include this functionality.
CLUSTERIDAFF1[7:0][b]	Input	Sets the value of the Cluster ID Aff1 field in the *Multiprocessor Affinity Register* (MPIDR or MPIDR_EL1).	Tie HIGH, if the system contains only one multiprocessor device. In a system with more than one multiprocessor, set this bus to a different value for each multiprocessor device. Software can use this field to uniquely identify each MPCore cluster. Only change this signal when all processors are in the reset state.
CLUSTERIDAFF2[7:0][b]	Input	Sets the value of the Cluster ID Aff2 field in the *Multiprocessor Affinity Register* (MPIDR or MPIDR_EL1).	
AA64nAA32[N:0]	Input	Sets the execution state for processor *N* as it exits reset.	Tie HIGH to set AArch64 state. Tie LOW to set AArch32 state. Only change this signal when processor *N* is in the reset state.
RVBARADDRx[43:2]	Input	If **AA64nAA32[N]** is HIGH then **RVBARADDRx[43:2]** sets the reset address for processor *N*.	Set this to the reset address you require for processor *N*, when it boots to AArch64 state.
CRYPTODISABLE[N:0][bc]	Input	Controls whether the Cryptography engine is disabled for processor *N*.	Tie HIGH to disable the Cryptography engine. Tie LOW to enable the Cryptography engine. Only change this signal when processor *N* is in the reset state. This signal only exists if the multiprocessor implements the Cryptography Extension.

a. See the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* for a description of the Secure system control registers.

b. This signal goes to both the **CLK** and **PCLKDBG** domains. The multiprocessor synchronizes the signal to the **PCLKDBG** domain.

c. The optional Cryptography engine is not included in the base product of the multiprocessor. ARM requires licensees to have contractual rights to obtain the Cortex-A57 MPCore multiprocessor Cryptography engine.

2.5 Interfaces

This section describes the inputs and outputs of the multiprocessor and the requirements for connecting them into your Soc design. It contains the following sections:

- *GIC CPU interface signals*.
- *Generic Timer signals* on page 2-11.
- *Power control signals* on page 2-11.
- *Memory interface signals for ACE and CHI implementations* on page 2-12.
- *Memory interface, CHI* on page 2-13.
- *Memory interface, ACE* on page 2-18.
- *ACP interface* on page 2-21.
- *Debug interface* on page 2-22.
- *ETM interface* on page 2-25.
- *Cross trigger channel interface* on page 2-24.
- *PMU signals* on page 2-26.
- *DFT interface* on page 2-26.
- *MBIST controller interface* on page 2-27.

2.5.1 GIC CPU interface signals

Table 2-4 shows the *Generic Interrupt Controller* (GIC) CPU interface signals and describes how to set these signals in your SoC. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-4 GIC CPU interface signals

Signal	Type	Description	Connection information
nIRQ[N:0] [a]	Input	IRQ request input for processor *N*. The processor treats the **nIRQ** input as level-sensitive. This signal is only used when IRQ is in bypass mode, where it is used as legacy IRQ.	Connect to interrupt sources or external interrupt controller.
nFIQ[N:0] [a]	Input	FIQ request input for processor *N*. The processor treats the **nFIQ** input as level-sensitive. This signal is only used when FIQ is in bypass mode, where it is used as legacy FIQ.	
nVFIQ[N:0] [a]	Input	Virtual FIQ request input for processor *N*. The processor treats the **nVFIQ** signal as level-sensitive.	Connect to interrupt sources or external interrupt controller. Tie HIGH if unused.
nVIRQ[N:0] [a]	Input	Virtual IRQ request input for processor *N*. The processor treats the **nVIRQ** signal as level-sensitive.	
nSEI[N:0] [a]	Input	System Error Interrupt request for processor *N*. The processor treats the **nSEI** signal as edge-sensitive. The **nSEI** signal must be sent as a pulse to the processor.	Connect to the system error generator logic. The value of **GICCDISABLE** has no effect on these signals
nREI[N:0] [a]	Input	RAM Error Interrupt request for processor *N*. The processor treats the **nREI** signal as edge-sensitive. The **nREI** signal must be sent as a pulse to the processor.	
nVSEI[N:0] [a]	Input	Virtual System Error Interrupt request for processor *N*. The processor treats the **nVSEI** signal as edge-sensitive. The **nVSEI** signal must be sent as a pulse to the processor.	

Table 2-4 GIC CPU interface signals (continued)

Signal	Type	Description	Connection information
GICCDISABLE[ab]	Input	Disables the GIC CPU interface logic and routes the legacy **nIRQ**, **nFIQ**, **nVIRQ**, and **nVFIQ** signals directly to the processor. Enables the use of non-ARM interrupt controllers. The multiprocessor only samples this signal as it exits reset.	Tie HIGH if the system contains an external GIC. If the system contains an external Distributor, tie LOW to enable the GIC CPU interface.
nVCPUMNTIRQ[N:0]	Output	Virtual CPU interface maintenance interrupt request from processor *N*. The processor uses this signal to send a maintenance interrupt request to the external Distributor.	If **GICCDISABLE** is LOW then connect to the external Distributor. Otherwise leave unconnected.
PERIPHBASE[43:18]	Input	Base address for the GIC registers. The multiprocessor only samples this signal as it exits reset.	Tie this bus to the appropriate address in your memory map.
AXI4 Stream protocol signals:[c]			
ICDTVALID	Input	Indicates when the Distributor is sending a valid transfer.	If **GICCDISABLE** is LOW then connect these AXI4-Stream signals to the Distributor. If **GICCDISABLE** is HIGH then tie inputs LOW and leave outputs unconnected.
ICDTREADY	Output	Indicates that the multiprocessor can accept a transfer in the current cycle.	
ICDTDATA[15:0]	Input	The primary payload that passes data from Distributor to multiprocessor.	
ICDTLAST	Input	Indicates the boundary of a packet.	
ICDTDEST[1:0]	Input	Provides routing information for the data stream from the Distributor.	
ICCTVALID	Output	Indicates when the multiprocessor is sending a valid transfer.	
ICCTREADY	Input	Indicates that the Distributor can accept a transfer in the current cycle.	
ICCTDATA[15:0]	Output	The primary payload passes data from the multiprocessor to the Distributor.	
ICCTLAST	Output	Indicates the boundary of a packet.	
ICCTID[1:0]	Output	The data stream identifier that indicates different streams of data.	

a. This signal goes to the **CLK** domain. The multiprocessor synchronizes the signal to the **CLK** domain.

b. This signal goes to both the **CLK** and **PCLKDBG** domains. The multiprocessor synchronizes the signal to the **PCLKDBG** domain.

c. See the *ARM® AMBA® AXI4-Stream™ Protocol Specification* for more information.

2.5.2 Generic Timer signals

Table 2-5 shows the Generic Timer signals. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-5 Generic Timer signals

Signal	Type	Description	Connection information
CNTVALUEB[63:0]	Input	Global system counter value in binary format.	Connect to the counter value from the timestamp generator, if present
CNTCLKEN	Input	Counter clock enable.	Connect to counter clock control or tie HIGH if not used
nCNTHPIRQ[N:0]	Output	Hypervisor physical timer interrupt.	Connect to your interrupt controller
nCNTPNSIRQ[N:0]	Output	Non-secure physical timer interrupt.	
nCNTPSIRQ[N:0]	Output	Secure physical timer interrupt.	
nCNTVIRQ[N:0]	Output	Virtual timer interrupt.	

2.5.3 Power control signals

Table 2-6 shows the power control signals and describes how to connect these signals in your SoC design. The value of *N* is one less than the number of processors that your multiprocessor implements.

Note

Clamps must not be enabled until the processor has reached one of its standby modes.

Table 2-6 Power control signals

Signal	Direction	Description	Connection information
CLREXMONREQ[a]	Input	Indicates that an external global exclusive monitor has been cleared. This sends a WFE wake-up event to all processors in the MPCore device.	Connect to your system global exclusive monitor. If global exclusive monitor is not present, tie LOW.
CLREXMONACK	Output	Clearing of the external global exclusive monitor acknowledge. When this signal is set to HIGH, the multiprocessor is acknowledging the receipt of the **CLREXMONREQ** signal.	Connect to your system global exclusive monitor.

Table 2-6 Power control signals (continued)

Signal	Direction	Description	Connection information
CPUQACTIVE[N:0]	Output	Indicates when processor *N* is active.	Connect to your system power controller
CPUQREQn[N:0][a]	Input	The power controller sets this signal LOW, to request that processor *N* enters retention state.	
CPUQACCEPTn[N:0]	Output	This signal goes LOW, if processor *N* accepts the power controller retention request.	
CPUQDENY[N:0]	Output	Indicates that processor *N* denies the power controller retention request.	
L2QACTIVE	Output	Indicates when the L2 Data and Tag RAMs are active.	
L2QREQn[a]	Input	The power controller sets this signal LOW, to request that the L2 Data and Tag RAMs enter retention state.	
L2QACCEPTn	Output	This signal goes LOW, if the L2 Data and Tag RAMs accept the power controller retention request.	
L2QDENY	Output	Indicates that the L2 Data and Tag RAMs deny the power controller retention request.	
STANDBYWFE[N:0]	Output	Indicates if processor *N* is in WFE low-power state.	
STANDBYWFI[N:0]	Output	Indicates if processor *N* is in WFI low-power state.	
STANDBYWFIL2	Output	Indicates if the L2 memory system is in the WFI low-power state.	
EVENTI[a]	Input	Event input for multiprocessor wake up from WFE low-power state. The power controller must assert this signal for at least one **CLK** cycle.	Connect to your system power controller
EVENTO	Output	Event output. The multiprocessor sets this signal HIGH for three **CLK** cycles when a processor executes an `SEV` instruction.	
L2FLUSHREQ[a]	Input	L2 hardware flush request.	
L2FLUSHDONE	Output	L2 hardware flush done.	
SMPEN[N:0]	Output	CPUECTLR SMP output.	

a. ABC This signal goes to the **CLK** domain. The multiprocessor synchronizes the signal to the **CLK** domain.

2.5.4 Memory interface signals for ACE and CHI implementations

The memory interface implements either the ACE or CHI protocol. The following sections describe the signals that are present, irrespective of the protocol implementation:

- *Broadcast configuration* on page 2-13.
- *Asynchronous error signals* on page 2-13.

Broadcast configuration

Table 2-7 shows the broadcast configuration signals for the memory interface.

Table 2-7 Broadcast configuration signals

Signal	Direction	Description	Connection information
BROADCASTCACHEMAINT[a]	Input	Enable broadcast cache maintenance to downstream caches[b]	The multiprocessor only samples these signals as it exits reset: • Tie HIGH to enable broadcast. • Tie LOW to disable broadcast.
BROADCASTINNER[a]	Input	Enable broadcast Inner Shareable transactions[b]	
BROADCASTOUTER[a]	Input	Enable broadcast Outer Shareable transactions[b]	
SYSBARDISABLE[c]	Input	Broadcast barrier to system bus disable	The multiprocessor only samples this signal as it exits reset: • Tie HIGH for AXI3 compatibility in the ACE configuration. • Tie LOW for AXI4 compatibility.

a. Tie LOW for full AXI3 compatibility in the ACE configuration.

b. See the *ARM® AMBA® AXI™ and ACE™ Protocol Specification* for valid combinations of **BROADCASTCACHEMAINT, BROADCASTINNER,** and **BROADCASTOUTER.**

c. If tied HIGH, **BROADCASTCACHEMAINT**, **BROADCASTINNER**, and **BROADCASTOUTER** must be tied LOW.

Asynchronous error signals

Table 2-8 shows the asynchronous error signals for the memory interface.

Table 2-8 Asynchronous error signals

Signal	Direction	Description	Connection information
nEXTERRIRQ	Output	Error indicator for an ACP write transaction with a write response error from AXI or CHI.	Connect to external interrupt control logic or external interrupt controller
nINTERRIRQ	Output	Error indicator for an L2 RAM multi-bit ECC error.	

2.5.5 Memory interface, CHI

If the Cortex-A57 MPCore multiprocessor implements a CHI memory interface then the following sections describe how to connect the memory interface signals:

- *CHI clock and configuration signals* on page 2-14.
- *Transmit request virtual channel signals* on page 2-14.
- *Transmit response virtual channel signals* on page 2-14.
- *Transmit data virtual channel signals* on page 2-15.
- *Receive snoop virtual channel signals* on page 2-15.
- *Receive response virtual channel signals* on page 2-15.
- *Receive data virtual channel signals* on page 2-16.
- *System address map signals* on page 2-16.

CHI clock and configuration signals

Table 2-9 shows the clock and configuration signals for the CHI interface.

Table 2-9 CHI clock and configuration signals

Signal	Type	Description	Connection information
SCLKEN	Input	CHI interface clock enable.	Connect to an appropriate CHI interconnect interface
SINACT	Input	CHI snoop active in.	
NODEID[6:0]	Input	CHI node identifier. The multiprocessor only samples this signal during reset.	Tie to the CHI node ID of the interconnect interface to which this cluster attaches
RXSACTIVE	Input	Receive pending activity indicator.	Connect to an appropriate CHI interconnect interface
TXSACTIVE	Output	Transmit pending activity indicator.	
RXLINKACTIVEREQ	Input	Receive link active request.	
RXLINKACTIVEACK	Output	Receive link active acknowledge.	
TXLINKACTIVEREQ	Output	Transmit link active request.	
TXLINKACTIVEACK	Input	Transmit link active acknowledge.	

Transmit request virtual channel signals

Table 2-10 shows the transmit request virtual channel signals for the CHI interface.

Table 2-10 Transmit request virtual channel signals

Signal	Type	Description	Connection information
TXREQFLITPEND	Output	Transmit request flit pending	Connect to an appropriate CHI interconnect interface
TXREQFLITV	Output	Transmit request flit valid	
TXREQFLIT[99:0]	Output	Transmit request flit payload	
TXREQLCRDV	Input	Transmit request link-layer credit valid	
REQMEMATTR[7:0]	Output	Transmit request raw memory attributes	

Transmit response virtual channel signals

Table 2-11 shows the transmit response virtual channel signals for the CHI interface.

Table 2-11 Transmit response virtual channel signals

Signal	Type	Description	Connection information
TXRSPFLITPEND	Output	Transmit response flit pending	Connect to an appropriate CHI interconnect interface
TXRSPFLITV	Output	Transmit response flit valid	
TXRSPFLIT[44:0]	Output	Transmit response flit payload	
TXRSPLCRDV	Input	Transmit response link-layer credit valid	

Transmit data virtual channel signals

Table 2-12 shows the transmit data virtual channel signals for the CHI interface.

Table 2-12 Transmit data virtual channel signals

Signal	Type	Description	Connection information
TXDATFLITPEND	Output	Transmit data flit pending	Connect to an appropriate CHI interconnect interface
TXDATFLITV	Output	Transmit data flit valid	
TXDATFLIT[193:0]	Output	Transmit data flit payload	
TXDATLCRDV	Input	Transmit data link-layer credit valid	

Receive snoop virtual channel signals

Table 2-13 shows the receive snoop virtual channel signals for the CHI interface.

Table 2-13 Receive snoop virtual channel signals

Signal	Type	Description	Connection information
RXSNPFLITPEND	Input	Receive snoop flit pending	Connect to an appropriate CHI interconnect interface
RXSNPFLITV	Input	Receive snoop flit valid	
RXSNPFLIT[64:0]	Input	Receive snoop flit payload	
RXSNPLCRDV	Output	Receive snoop link-layer credit valid	

Receive response virtual channel signals

Table 2-14 shows the receive response virtual channel signals for the CHI interface.

Table 2-14 Receive response virtual channel signals

Signal	Type	Description	Connection information
RXRSPFLITPEND	Input	Receive response flit pending	Connect to an appropriate CHI interconnect interface
RXRSPFLITV	Input	Receive response flit valid	
RXRSPFLIT[44:0]	Input	Receive response flit payload	
RXRSPLCRDV	Output	Receive response link-layer credit valid	

Receive data virtual channel signals

Table 2-15 shows the receive data virtual channel signals for the CHI interface.

Table 2-15 Receive data virtual channel signals

Signal	Type	Description	Connection information
RXDATFLITPEND	Input	Receive data flit pending	Connect to an appropriate CHI interconnect interface
RXDATFLITV	Input	Receive data flit valid	
RXDATFLIT[193:0]	Input	Receive data flit payload	
RXDATLCRDV	Output	Receive data link-layer credit valid	

System address map signals

Table 2-16 shows the system address map signals for the CHI interface. The multiprocessor only samples the **SAM*** signals when it exits reset.

Table 2-16 System address map signals

Signal	Type	Description	Connection information
SAMMNBASE[43:24]	Input	MN base address	Tie this bus to the appropriate address in your memory map. Only change this address when the multiprocessor is in the reset state.

Table 2-16 System address map signals (continued)

Signal	Type	Description	Connection information
SAMADDRMAP0[1:0]	Input	0 to 512MB region mapping	Tie off these buses to values matching the corresponding to CHI interconnect address map.
SAMADDRMAP1[1:0]	Input	512MB to 1GB region mapping	
SAMADDRMAP2[1:0]	Input	1GB to 1.5GB region mapping	
SAMADDRMAP3[1:0]	Input	1.5GB to 2GB region mapping	
SAMADDRMAP4[1:0]	Input	2GB to 2.5GB region mapping	
SAMADDRMAP5[1:0]	Input	2.5GB to 3GB region mapping	
SAMADDRMAP6[1:0]	Input	3GB to 3.5GB region mapping	
SAMADDRMAP7[1:0]	Input	3.5GB to 4GB region mapping	
SAMADDRMAP8[1:0]	Input	4GB to 8GB region mapping	
SAMADDRMAP9[1:0]	Input	8GB to 16GB region mapping	
SAMADDRMAP10[1:0]	Input	16GB to 32GB region mapping	
SAMADDRMAP11[1:0]	Input	32GB to 64GB region mapping	
SAMADDRMAP12[1:0]	Input	64GB to 128GB region mapping	
SAMADDRMAP13[1:0]	Input	128GB to 256GB region mapping	
SAMADDRMAP14[1:0]	Input	256GB to 512GB region mapping	
SAMADDRMAP15[1:0]	Input	512GB to 1TB region mapping	
SAMADDRMAP16[1:0]	Input	1TB to 2TB region mapping	
SAMADDRMAP17[1:0]	Input	2TB to 4TB region mapping	
SAMADDRMAP18[1:0]	Input	4TB to 8TB region mapping	
SAMADDRMAP19[1:0]	Input	8TB to 16TB region mapping	
SAMMNNODEID[6:0]	Input	MN node ID	Tie off these buses to values that configure your CHI interconnect as you require.
SAMHNI0NODEID[6:0]	Input	HN-I 0 node ID	
SAMHNI1NODEID[6:0]	Input	HN-I 1 node ID	
SAMHNF0NODEID[6:0]	Input	HN-F 0 node ID	
SAMHNF1NODEID[6:0]	Input	HN-F 1 node ID	
SAMHNF2NODEID[6:0]	Input	HN-F 2 node ID	
SAMHNF3NODEID[6:0]	Input	HN-F 3 node ID	
SAMHNF4NODEID[6:0]	Input	HN-F 4 node ID	
SAMHNF5NODEID[6:0]	Input	HN-F 5 node ID	
SAMHNF6NODEID[6:0]	Input	HN-F 6 node ID	
SAMHNF7NODEID[6:0]	Input	HN-F 7 node ID	
SAMHNFMODE[2:0]	Input	HN-F interleaving mode	

2.5.6 Memory interface, ACE

This section describes how to connect the memory interface signals when the Cortex-A57 MPCore multiprocessor implements an AMBA 4 *AXI Coherency Extensions* (ACE) master interface.

Table 2-17 shows the clock and configuration signals for the ACE master interface.

Table 2-17 Clock and configuration signals for ACE master interface

Signal	Direction	Description	Connection information
ACLKENM	Input	ACE master bus clock enable.	Connect to the clock generation logic or tie HIGH if not used.
ACINACTM	Input	Prevents the L2 memory system from accepting any new requests from the ACE master interface.	Tie HIGH to enable the L2 memory system to: • Enter low-power state. • Enter L2 WFI low-power state. • Assert **STANDBYWFIL2**.

Table 2-18 shows the ACE master interface signals and describes how to connect these signals to an AXI3 or AXI4 bus infrastructure in your SoC design.

See the *ARM® AMBA® AXI™ and ACE™ Protocol Specification* and the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* for more information.

Table 2-18 ACE master interface signals

Signal	Direction	Description	Connection information
ARADDRM[43:0]	Output	Read address	Connect to the slaves through the AXI3 or AXI4 bus infrastructure
ARBURSTM[1:0]	Output	Read burst type	
ARCACHEM[3:0]	Output	Read cache type	
ARIDM[5:0]	Output	Read address ID	
ARLENM[7:0][a]	Output	Read burst length	
ARLOCKM	Output	Read lock type	
ARPROTM[2:0]	Output	Read protection type	
ARREADYM	Input	Read address ready	
ARSIZEM[2:0]	Output	Read burst size	
ARVALIDM	Output	Read address valid	
AWADDRM[43:0]	Output	Write address	
AWBURSTM[1:0]	Output	Write burst type	
AWCACHEM[3:0]	Output	Write cache type	
AWIDM[5:0]	Output	Write address ID	
AWLENM[7:0][b]	Output	Write burst length	
AWLOCKM	Output	Write lock type	
AWPROTM[2:0]	Output	Write protection type	
AWREADYM	Input	Write address ready	

Table 2-18 ACE master interface signals (continued)

Signal	Direction	Description	Connection information
AWSIZEM[2:0]	Output	Write burst size	
AWVALIDM	Output	Write address valid	
BIDM[5:0]	Input	Write response ID	
BREADYM	Output	Write response ready	
BRESPM[1:0]	Input	Write response	
BVALIDM	Input	Write response valid	
RDATAM[127:0]	Input	Read data	
RIDM[5:0]	Input	Read ID	
RLASTM	Input	Read last	
RREADYM	Output	Read ready	
RRESPM[3:0]	Input	Read response	
RVALIDM	Input	Read valid	
WDATAM[127:0]	Output	Write data	
WIDM[5:0]	Output	Write ID	
WLASTM	Output	Write last	
WREADYM	Input	Write ready	
WSTRBM[15:0]	Output	Write strobe	
WVALIDM	Output	Write valid	
The following signals are not present in the AMBA AXI protocol:			
RDMEMATTR[7:0]	Output	Read request raw memory attributes	You can connect these output sideband signals to your SoC interconnect or leave them unconnected
WRMEMATTR[7:0]	Output	Write request raw memory attributes	

a. **ARLENM[7:2]** is always `0b000000`.

b. **AWLENM[7:2]** is always `0b000000`.

Table 2-19 on page 2-20 shows the AXI coherency extension signals and describes how to connect these signals to the AXI4 bus infrastructure in your SoC design. If your design implements an AXI3 bus infrastructure, tie the inputs LOW and leave the outputs unconnected.

Table 2-19 AXI coherency extension signals

Signal	Direction	Description	Connection information
ARBARM[1:0]	Output	Read barrier type	Connect to the slaves through the AXI4 bus infrastructure
ARDOMAINM[1:0]	Output	Read domain type	
ARSNOOPM[3:0]	Output	Read snoop request type	
AWBARM[1:0]	Output	Write barrier type	
AWDOMAINM[1:0]	Output	Write domain type	
AWSNOOPM[2:0]	Output	Write snoop request type	
AWUNIQUEM	Output	Indicates if the WriteBack, WriteClean, or WriteEvict transaction is Shared or Unique	
ACADDRM[43:0]	Input	Snoop address	
ACPROTM[2:0]	Input	Snoop protection type	
ACREADYM	Output	Snoop address ready	
ACSNOOPM[3:0]	Input	Snoop transaction type	
ACVALIDM	Input	Snoop address valid	
CDDATAM[127:0]	Output	Snoop data	
CDLASTM	Output	Snoop last	
CDREADYM	Input	Snoop ready	
CDVALIDM	Output	Snoop valid	
CRREADYM	Input	Snoop response ready	
CRRESPM[4:0]	Output	Snoop response	
CRVALIDM	Output	Snoop response valid	
RACKM	Output	Snoop read acknowledge	
WACKM	Output	Snoop write acknowledge	

2.5.7 ACP interface

The *Accelerator Coherency Port* (ACP) is an AMBA 4 AXI slave interface.

Table 2-20 shows the clock and configuration signals for the ACP interface.

Table 2-20 Clock and configuration signals for ACP interface

Signal	Direction	Description	Connection information
ACLKENS	Input	AXI master bus clock enable	Connect to the clock generation logic or tie HIGH if not used.
AINACTS	Input	Prevents the L2 memory system from accepting any new requests from the ACP slave interface	You must drive this signal HIGH to enable the L2 memory system to: • Enter low-power state. • Enter L2 WFI mode. • Assert **STANDBYWFIL2**.

Table 2-21 shows the ACP interface signals and describes how to connect these signals to the AXI4 bus infrastructure in your SoC design.

Table 2-21 ACP interface signals

Signal	Direction	Description	Connection information
ARADDRS[43:0]	Input	Read address	Connect to the master AXI4 bus infrastructure
ARCACHES[3:0]	Input	Read cache type	
ARIDS[4:0]	Input	Read request ID	
ARLENS[7:0]	Input	Read burst length	
ARPROTS[2:0]	Input	Read protection type	
ARREADYS	Output	Read address ready	
ARUSERS[1:0]	Input	User signals for read shareability	
ARVALIDS	Input	Read address valid	
AWADDRS[43:0]	Input	Write address	
AWIDS[4:0]	Input	Write request ID	
AWCACHES[3:0]	Input	Write cache type	
AWLENS[7:0]	Input	Write burst length	
AWPROTS[2:0]	Input	Write protection type	
AWREADYS	Output	Write address ready	
AWUSERS[1:0]	Input	User signals for write shareability	
AWVALIDS	Input	Write address valid	
BIDS[4:0]	Output	Write response ID	
BREADYS	Input	Write response ready	
BRESPS[1:0]	Output	Write response	
BVALIDS	Output	Write response valid	
RDATAS[127:0]	Output	Read data	

Table 2-21 ACP interface signals (continued)

Signal	Direction	Description	Connection information
RIDS[4:0]	Output	Read ID	
RLASTS	Output	Read last	
RREADYS	Input	Read ready	
RRESPS[1:0]	Output	Read response	
RVALIDS	Output	Read response valid	
WDATAS[127:0]	Input	Write data	
WLASTS	Input	Write last	
WREADYS	Output	Write ready	
WSTRBS[15:0]	Input	Write strobes	
WVALIDS	Input	Write valid	

2.5.8 Debug interface

This section describes how to connect the debug interface and debug APB interface signals in your SoC design. The debug signals are described in:

- *APB debug interface*.
- *Miscellaneous debug interface signals* on page 2-23.
- *Authentication interface* on page 2-24.

APB debug interface

Table 2-22 shows the external debug interface signals. These signals are synchronous to **PCLKDBG**.

Table 2-22 APB debug signals

Signal	Direction	Description	Connection information
PCLKENDBG	Input	Clock enable for the debug APB slave interface	Connect to debug clock control
PADDRDBG[21:2]	Input	APB address bus bits[21:2]	Connect to the CoreSight DAP
PADDRDBG31	Input	APB address bus bit[31]	
PWDATADBG[31:0]	Input	APB write data bus	
PENABLEDBG	Input	Indicates a second and subsequent cycle of a transfer	
PSELDBG	Input	Selects the external debug interface	
PWRITEDBG	Input	APB read and write signal	
PREADYDBG	Output	Extend a transfer by inserting wait states	
PSLVERRDBG	Output	ABP slave transfer error	
PRDATADBG[31:0]	Output	APB read data bus	

Miscellaneous debug interface signals

Table 2-23 lists the miscellaneous debug signals. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-23 Miscellaneous debug interface signals

Signal	Direction	Description	Connection information
COMMRX[N:0]	Output	Receive communication channel.	If you require interrupt-driven debug communications operations, connect these signals to the interrupt controller. If you do not require interrupt-driven debug communications, you can leave these signals unconnected.
COMMTX[N:0]	Output	Transmit communication channel.	
DBGACK[N:0]	Output	Debug acknowledge.	Connect this signal to all the debug request signal sources in your system. Typically used directly or indirectly, through debug, trace or trigger control logic, by a debugger.
nCOMMIRQ[N:0]	Output	Communications channel receive or transmit interrupt request.	Connect to the interrupt controller.
EDBGRQ[N:0][a]	Input	External debug request.	The ORed value of all debug request signal sources in your system must be connected to this signal. Typically used directly or indirectly, through debug, trace or trigger control logic, by a debugger.
DBGROMADDR[43:12]	Input	Debug ROM physical address.	Set this signal to the value of bits[43:12] of the system base address of the debug ROM. If you want to change this value, the change must occur when the multiprocessor is in the reset state. In systems with multiple debug ROMs, you must set this value to the address of the top-level ROM address. Only change this signal during Debug reset.
DBGROMADDRV	Input	Debug ROM physical address valid.	Set this signal HIGH. If the debug ROM physical address cannot be determined, you must set: • **DBGROMADDR[43:12]** to all zeros. • **DBGROMADDRV** LOW. Only change this signal during Debug reset.
DBGNOPWRDWN[N:0]	Output	No power down request. Debugger has requested that processor *N* is not powered down.	Connect this signal to your system power controller. The system power controller must not remove power from processor *N* or memory when this signal is HIGH, even during a power down event.
DBGPWRDUP[N:0]	Input	Processor *N* power status.	Connect to your system power controller.
DBGPWRUPREQ[N:0]	Output	Processor *N* power up request.	
DBGRSTREQ[N:0]	Output	Processor *N* Warm reset request. Goes HIGH when an external debugger sets the EDPRCR.CWRR register bit to 1, for processor *N*.	

a. This signal goes to the **PCLKDBG** domain. The multiprocessor synchronizes the signal to the **PCLKDBG** domain.

Authentication interface

Table 2-24 shows the authentication interface and how to connect the signals in your SoC design. These signals are either tied to a fixed value or controlled by an external device. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-24 Authentication interface

Signal[a]	Direction	Description	Connection information
DBGEN[N:0]	Input	Invasive debug enable	Tie HIGH for debug functionality. **Caution** If you tie this signal LOW, debugging is not possible.
NIDEN[N:0]	Input	Non-invasive debug enable	Tie HIGH to enable non-invasive debug or LOW to disable non-invasive debug. **Caution** If you tie this signal LOW, non-invasive debug features such as the PMU and ETM are not available.
SPIDEN[N:0]	Input	Secure privileged invasive debug enable	Tie HIGH to enable secure privileged invasive debug or LOW to disable secure privileged invasive debug.
SPNIDEN[N:0]	Input	Secure privileged non-invasive debug enable	Tie HIGH to enable secure privileged non-invasive debug or LOW to disable secure privileged non-invasive debug.

a. The processor synchronizes these signals to an internal clock domain.
This signal goes to both the **CLK** and **PCLKDBG** domains. The multiprocessor synchronizes the signal to the **CLK** and the **PCLKDBG** domains.

2.5.9 Cross trigger channel interface

Each processor in the multiprocessor has its own CoreSight *Cross Trigger Interface* (CTI) that broadcasts the trigger requests it receives to other processor CTIs through a simplified *Cross Trigger Matrix* (CTM) as channel events. When a CTI receives a channel event, it maps the event to a trigger output. This enables the processors to cross trigger each other. See the *ARM® CoreSight™ Components Technical Reference Manual* and the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* for more information.

Table 2-25 shows the cross trigger channel interface signals. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-25 Cross trigger channel interface signals

Signal	Direction	Description	Connection information
CIHSBYPASS[3:0]	Input	CTI handshake bypass	Tie this signal: • HIGH to enable bypass. • LOW to disable bypass.
CISBYPASS	Input	CTI sync bypass	Tie this signal: • HIGH to enable bypass. • LOW to disable bypass.
CTICHIN[3:0]	Input	CTI input channels	Connect to channel output of another subsystem. Tie LOW if unused.

Table 2-25 Cross trigger channel interface signals (continued)

Signal	Direction	Description	Connection information
CTICHINACK[3:0]	Output	CTI input channel acknowledge	Connect to channel output acknowledge of another subsystem. Leave unconnected if unused.
CTICHOUT[3:0]	Output	CTI output channels	Connect to channel output of another subsystem. Leave unconnected if unused.
CTICHOUTACK[3:0]	Input	CTI output channel acknowledge	Connect to channel input acknowledge of another subsystem. Tie LOW if unused.
CTIIRQ[N:0]	Output	CTI interrupt (active-high). For system-level integration see Figure 2-2.	Connect to the interrupt controller, with inversion if necessary. Leave unconnected if unused.
CTIIRQACK[N:0] [a]	Input	CTI interrupt acknowledge	Tie HIGH if no handshaking is required on **CTIIRQ**. Tie LOW to enable software handshaking. With an interrupt controller that uses level-sensitive inputs, software handshaking must be used.

a. This signal goes to the **PCLKDBG** domain. The multiprocessor synchronizes the signal to the **PCLKDBG** domain.

Connect the **CTIIRQACK** feedback path as shown in Figure 2-2. The feedback path that passes through registers ensures that the pulse sent by the Cortex-A57 MPCore processor stays asserted until it is visible in the GIC domain. **CTIIRQACK** is a synchronized input as described in Input synchronization *Synchronization of signals* on page 2-4 and does not require synchronize registers.

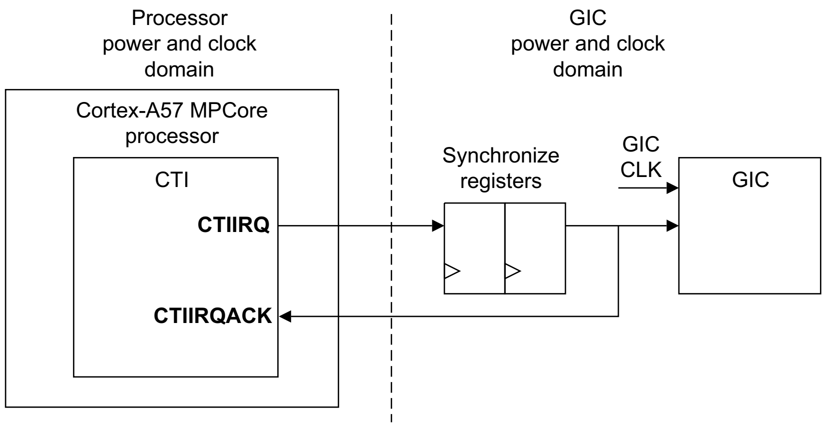

Processor power and clock domain
GIC power and clock domain
Cortex-A57 MPCore processor
CTI
CTIIRQ
CTIIRQACK
Synchronize registers
GIC CLK
GIC

Figure 2-2 System-level GIC interface

2.5.10 ETM interface

This section describes the ETM signals. Each processor in the multiprocessor has its own ETM. See the *ARM® Embedded Trace Macrocell Architecture Specification* for more information.

The ETM implements an AMBA 3 ATB interface that outputs trace information from the CoreSight components for debugging purposes. See the *ARM® AMBA® 3 ATB Protocol Specification* and the *ARM® CoreSight™ Components Technical Reference Manual* for more information.

Table 2-26 shows the ATB interface signals. The value of *x* is one less than the number of processors that your multiprocessor implements.

Table 2-26 ATB interface signals

Signal	Direction	Description	Connection information
AFREADYMx	Output	ATB FIFO flush acknowledge	Connect to your CoreSight trace system
AFVALIDMx	Input	ATB FIFO flush request	
ATBYTESMx[1:0]	Output	ATB device data size	
ATDATAMx[31:0]	Output	ATB data bus	
ATIDMx[6:0]	Output	ATB trace source identification	
ATREADYMx	Input	ATB device ready	
ATVALIDMx	Output	ATB valid data	
ATCLKEN	Input	ATB clock enable	
SYNCREQMx	Input	ATB synchronization request	

Miscellaneous ETM signal

Table 2-27 shows the miscellaneous ETM signal.

Table 2-27 Miscellaneous ETM signal

Signal	Direction	Description	Connection information
TSVALUEB[63:0]	Input	Global system timestamp value	Connect to your CoreSight system

2.5.11 PMU signals

Table 2-28 shows the *Performance Monitoring Unit* (PMU) signals and describes how to connect these signals in your SoC design. The value of *N* is one less than the number of processors and the value of *m* represents processor 0, 1, 2, or 3 in your multiprocessor.

Table 2-28 PMU signals

Signal	Direction	Description	Connection information
nPMUIRQ[N:0]	Output	PMU interrupt	Connect to interrupt sources or external interrupt controller
PMUEVENTm[24:0]	Output	PMU event bus	Connect to the trace logic of your SoC
PMUSNAPSHOTREQ[N:0]	Input	PMU snapshot trigger request	
PMUSNAPSHOTACK[N:0]	Output	PMU snapshot trigger acknowledge	

2.5.12 DFT interface

The Cortex-A57 MPCore multiprocessor implements a *Design For Test* (DFT) interface that enables an industry standard *Automatic Test Pattern Generation* (ATPG) tool to test logic outside of the embedded memories.

Table 2-29 shows the DFT interface signals. The value of *N* is one less than the number of processors that your multiprocessor implements.

Table 2-29 DFT interface signals

Signal	Direction	Description	Connection information
DFTCLKBYPASS	Input	L2 strobe clock register bypass	Connect to the test controller port. Tie LOW if unused.
DFTCRCLKDISABLE[N:0]	Input	Processor *N* clock grid disable	
DFTL2CLKDISABLE	Input	L2 clock grid disable	
DFTMCPHOLD	Input	Multi-cycle path disable on RAM interfaces	
DFTRAMHOLD	Input	RAM chip select disable	
DFTRSTDISABLE	Input	Internal reset disable	
DFTSE	Input	Scan shift enable	

2.5.13 MBIST controller interface

The *Memory Built-In Self Test* (MBIST) controller interface provides support for manufacturing testing of the memories embedded in the Cortex-A57 MPCore multiprocessor. MBIST is the industry standard method of testing memories.

Note

The Cortex-A57 MPCore multiprocessor does not contain embedded MBIST controllers. The controller must be implemented external to the multiprocessor and execute tests on the internal memory structure with the test interfaces supported at the multiprocessor I/O boundary.

Table 2-30 shows the MBIST interface signals.

Table 2-30 MBIST interface signals

Signal	Direction	Description	Connection information
nMBISTRESET	Input	MBIST reset	Tie HIGH if unused. See the *MBIST interface signals* on page 3-9.
MBISTREQ	Input	MBIST test request	Tie LOW if unused. See the *MBIST interface signals* on page 3-9.

See *MBIST interfaces* on page 3-8 for more information.

Chapter 3
DFT Integration Guidelines

This chapter provides guidelines for *Design For Test* (DFT). It contains the following sections:

3.1 About DFT integration

This chapter describes the issues that relate to DFT that you must consider when you integrate the multiprocessor into your SoC design. The multiprocessor supports scan and *Automatic Test Pattern Generation* (ATPG) techniques for production test vectors, and *Memory Built-In Self Test* (MBIST). The macrocell also uses scan compression to reduce the test pattern volume and test time and uses a scan wrapper to isolate the macrocell and provide testability of the shadow logic outside the macrocell.

This chapter describes:

- Test interface on the macrocell.
- Production test modes of the macrocell.
- Scan configuration of the macrocell.
- Scan wrapper structure and usage.
- MBIST and repair interfaces.
- Types of tests supported.

When you design the DFT strategy for your SoC, you must:

- Carefully consider access to the multiprocessor signals for control and observation during test.
- Ensure that, during production test, you can control the multiprocessor for all required test modes.

Failure to do this might make the multiprocessor impossible to test. There are two cases for controlling the multiprocessor signals:

- The dynamic signals that must be controlled on a cycle-by-cycle basis must be either:
  - Brought directly to the top-level of the chip for control by the tester.
  - Multiplexed with normal functional signals and made available during the manufacturing test modes using a test mode controller.

  This case also applies to all output signals that are compared during test modes.
- The control signals that are static during a test can be driven by an SoC test controller or directly connected to a top-level SoC pin.

For information about how the test signals are assigned to these two cases, see *ATPG test interface* on page 3-4.

The multiprocessor test patterns might require modification for additional test setup vectors to control SoC logic that connects to the multiprocessor DFT signals such as a *Test Access Port* (TAP) or PLL.

3.2 ATPG test interface

The implementation reference flow for Cortex-A57 MPCore multiprocessor includes scan compression and test wrappers. Figure 3-1 shows the implemented scan compression.

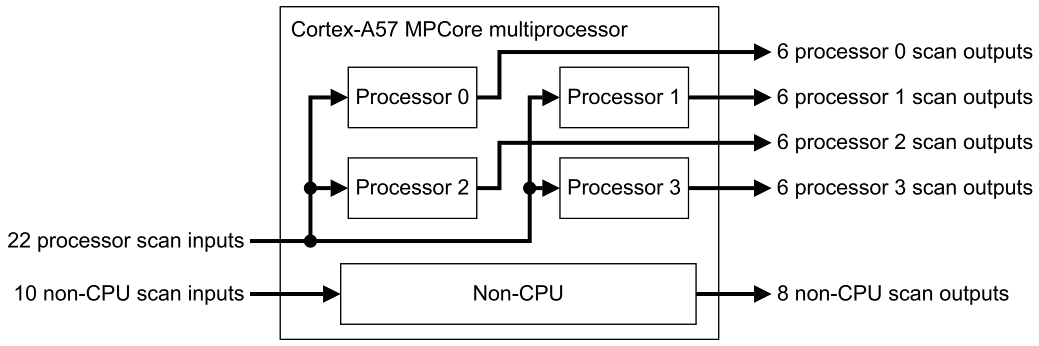

Cortex-A57 MPCore multiprocessor
Processor 0
Processor 1
Processor 2
Processor 3
Non-CPU
22 processor scan inputs
10 non-CPU scan inputs
6 processor 0 scan outputs
6 processor 1 scan outputs
6 processor 2 scan outputs
6 processor 3 scan outputs
8 non-CPU scan outputs

Figure 3-1 Scan compression

The reference flow provides 32 scan inputs and 32 scan outputs. All processors share a common 22-bit scan input bus with each processor having its own separate 6-bit scan output bus. The non-CPU block uses the remaining scan inputs. A test wrapper is also installed in the non-CPU block.

3.2.1 Scan considerations

The Cortex-A57 MPCore multiprocessor implementation reference flow includes the following pre-built macros:

`atl_cpu`	Processor macro.
`atl_noncpu`	Non-CPU macro.

Figure 3-2 shows how these macros connect to the top-level Cortex-A57 MPCore multiprocessor stitching level. All external I/O, including implementation scan channels, are serviced by the non-CPU block.

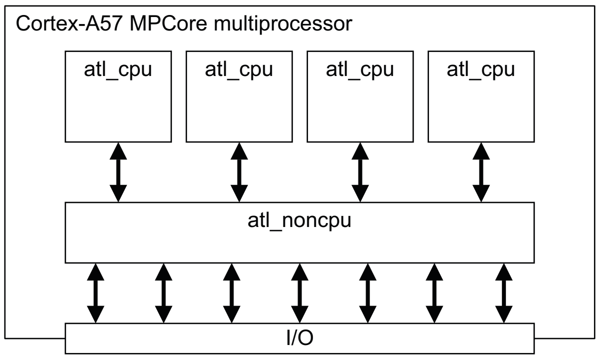


Figure 3-2 Cortex-A57 MPCore multiprocessor scan overview

When you architect your DFT solution, you must consider the estimated flop counts in the `atl_cpu` and `atl_noncpu` blocks and I/O usage in your implementation. If a wrapper is required and the ARM implementation reference flow is followed then the wrapper would only install in the `atl_noncpu` synthesis region.

The two primary clocks in the multiprocessor are **CLK** and **PCLKDBG**. The Level 2 RAMs in the `atl_noncpu` block receive generated clocks because they do not run at the full system frequency. The source of the generated clock is either:

- A local clock generator (not stretched) whose enable has a **DFTSE** override.
- A flop output (stretched) used to deliver a 50% duty cycle.

The choice of clock source is set during the RTL configuration process and depends on whether `` `ATL_STRETCH_L2RAMCLK `` is defined. See the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* for more information.

See *DFTCLKBYPASS signal* for ATPG considerations when `ATL_STRETCH_L2RAMCLK` is defined.

3.2.2 ATPG test interface

Table 3-1 shows the ATPG test signals.

Table 3-1 ATPG signals

Signal	Direction	Functional mode	Usage
DFTCLKBYPASS	Input	`0b0`	See *DFTCLKBYPASS signal*.
DFTCRCLKDISABLE[N:0]	Input	`0b0`	When HIGH, it disables the clock to processor *N*.
DFTRSTDISABLE	Input	`0b0`	See *DFTRSTDISABLE signal* on page 3-5.
DFTL2CLKDISABLE	Input	`0b0`	When HIGH, it disables the clock to the L2 RAMs.
DFTMCPHOLD	Input	`0b0`	See *DFTMCPHOLD signal* on page 3-5.
DFTRAMHOLD	Input	`0b0`	See *DFTRAMHOLD signal* on page 3-5.
DFTSE	Input	`0b0`	See *DFTSE signal* on page 3-6.
nMBISTRESET	Input	`0b1`	When LOW, it performs an MBIST mode reset. See *nMBISTRESET signal* on page 3-12.

DFTCLKBYPASS signal

As *Scan considerations* on page 3-3 describes, the L2 RAMs can be clocked from one of two sources.

If during the RTL implementation process, the define:

`ATL_STRETCH_L2RAMCLK` **is defined**

The RAM clock originates from a flop. In this case, if there are embedded scan flops in the RAM macros then the ATPG tool fails the scan shift DRC because of an uncontrolled clock. Similarly, the DRC identifies an uncontrolled RAM clock if testing through the RAM.

Setting **DFTCLKBYPASS** HIGH forces selection of the local clock-gated version of system clock, **CLK**, and resolves the DRC issues.

`ATL_STRETCH_L2RAMCLK` **is not defined**

The RAM clock originates from the local clock-gated version of system clock, **CLK**. In this case, there is no requirement to use **DFTCLKBYPASS**.

DFTMCPHOLD signal

DFTMCPHOLD disables certain multi-cycle paths. For example, during implementation the Level 2 cache Data RAM outputs are multi-cycle paths. In the reference implementation, these RAMs include embedded scan and are normally X-value free. However in delay-based ATPG, the RAMs might toggle their outputs so the ATPG simulator classifies the multi-cycle path as an X. This adversely affects test compression circuits causing a pattern count growth and potential drops in test coverage of delay fault models. This effect becomes more pronounced as the number of compression output channels decrease.

To prevent this X-pollution, set **DFTMCPHOLD** HIGH so that it:

- Blocks the clock to the multi-cycle RAMs during capture.
- Blocks the **ATCLKEN** domain logic.

To take advantage of this feature, a possible ATPG flow for transition delay is:

1. Set **DFTMCPHOLD** HIGH and generate F_{max} transition delay patterns.
2. Set **DFTMCPHOLD** LOW and generate F_{max} transition delay patterns, capturing single cycle inputs of the multi-cycled RAMs. The ATPG tool treats any multi-cycle RAM outputs and **ATCLKEN** domain logic states that toggle as Xs.
3. With **DFTMCPHOLD** LOW, generate F_{max}/MCP# delay patterns, capturing multi-cycle RAM fanouts and **ATCLKEN** domain logic.

DFTMCPHOLD does not block the **PCLKDBG** domain because the ATPG tool can control when to block the **PCLKDBG** signal.

DFTRAMHOLD signal

The **DFTRAMHOLD** signal disables the chip selects to the RAMs, so it effectively holds the data in the RAMs. Figure 3-3 shows the logic function.

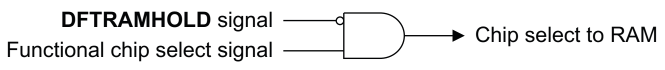


Figure 3-3 DFTRAMHOLD logic

This capability is useful in the following scenarios:

- To maintain the value in the RAMs during tests such as IDDQ.
- Testing shadow logic of the RAMs by testing through the RAMs.

  ATPG tools prefer the data in the RAMs to be static during shift. In this scenario, the tester sets **DFTRAMHOLD** and **DFTSE** HIGH during shift, and then LOW during capture.

Use of **DFTRAMHOLD** significantly improves test coverage and test pattern reduction especially when implementations do not include embedded scan inside their RAM macros.

DFTRSTDISABLE signal

The reset logic in the Cortex-A57 MPCore multiprocessor incorporates asynchronous assertion and synchronous deassertion. Figure 3-4 on page 3-6 shows the reset synchronization structure.

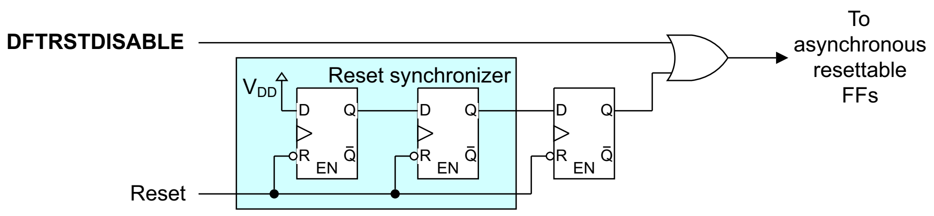


Figure 3-4 Reset logic

DFTRSTDISABLE is a top-level input that enables an ATPG tool to:

- Safely shift in scan chains.
- Test the functional deassertion of reset.

It is intended that an ATPG tool controls this input in a similar manner to the scan enable signal **DFTSE**. They are separate signals in the design to allow ATPG patterns to be generated with **DFTRSTDISABLE** driven HIGH at all times that prevents the internal resets from occurring in those patterns. The tester must also perform a capture when **DFTRSTDISABLE** is LOW, to get full test coverage of the asynchronous reset inputs to the flip-flops downstream from the synchronizers. If this signal is disabled during capture for only a single pattern set, it can make reset problems easier to diagnose on the tester. The ATPG tool must control the reset signals similar to the clock signals, so that it can provide coverage on the reset inputs to the synchronizers.

Note

- ARM recommends that you do not connect the **DFTSE** and **DFTRSTDISABLE** inputs together. If **DFTRSTDISABLE** is independent of **DFTSE** then it enables the tester to apply separate ATPG patterns with resets active and with resets inactive. If the signals connect together, the ATPG patterns cannot capture functional paths when resets are active.

- The ATPG tool might require special commands to test this reset structure. See the ATPG tool documentation for information about the commands and syntax to use.

The **DFTRSTDISABLE** signal must not change state during capture otherwise that can result in failing patterns because of race conditions in the reset logic.

If the ATPG tool performs delay tests when **DFTRSTDISABLE** is LOW then this might propagate at-speed reset assertions that are multi-cycled in the implementation flow. Only the deassertion of reset can be tested at-speed. If your ATPG tool does not read in SDF annotation during ATPG generation, you might have to avoid the potential for patterns that include internal at-speed reset assertions. If **DFTRSTDISABLE** is inactive then to prevent at-speed reset assertions, the reset synchronizer flops must be properly constrained to inactive reset states during at-speed pattern generation.

DFTSE signal

The scan enable signal, **DFTSE**, activates the clock enables of the architectural clock gates, and ensures that the primary clocks in the multiprocessor are active during shift mode.

ARM expects implementers to use this signal as the primary scan enable signal to:

- Control whether a register is in shift or capture mode.
- Enable the overrides to all RTL clock gates and any clock gates that synthesis adds.

Because the scan enable signal only controls the scan function that is in the implementation model, it must be disabled during formal equivalence checking between the gate-level model and the RTL. Formal equivalence checking cannot check the scan chain shift path or the override on the clock gates.

3.3 MBIST interfaces

The multiprocessor has three MBIST interfaces, to provide test access to the different logical memories. These interfaces are:

- L1 interface.
- L2 tag bank 0 interface.
- L2 tag bank 1 interface.

The size and location of these interfaces were chosen for a balance of lowest functional design impact and minimum test time. If less test time is required, you can install MBIST at the physical arrays. In this case, the ARM MBIST interfaces are not utilized.

All three MBIST interfaces enter MBIST test mode simultaneously and are controlled from the **MBISTREQ** and **nMBISTRESET** signals on the multiprocessor I/O boundary.

Figure 3-5 shows the MBIST interface hierarchy and embedded memory interface attachment points. All attachment points reside in the `atl_noncpu` block.

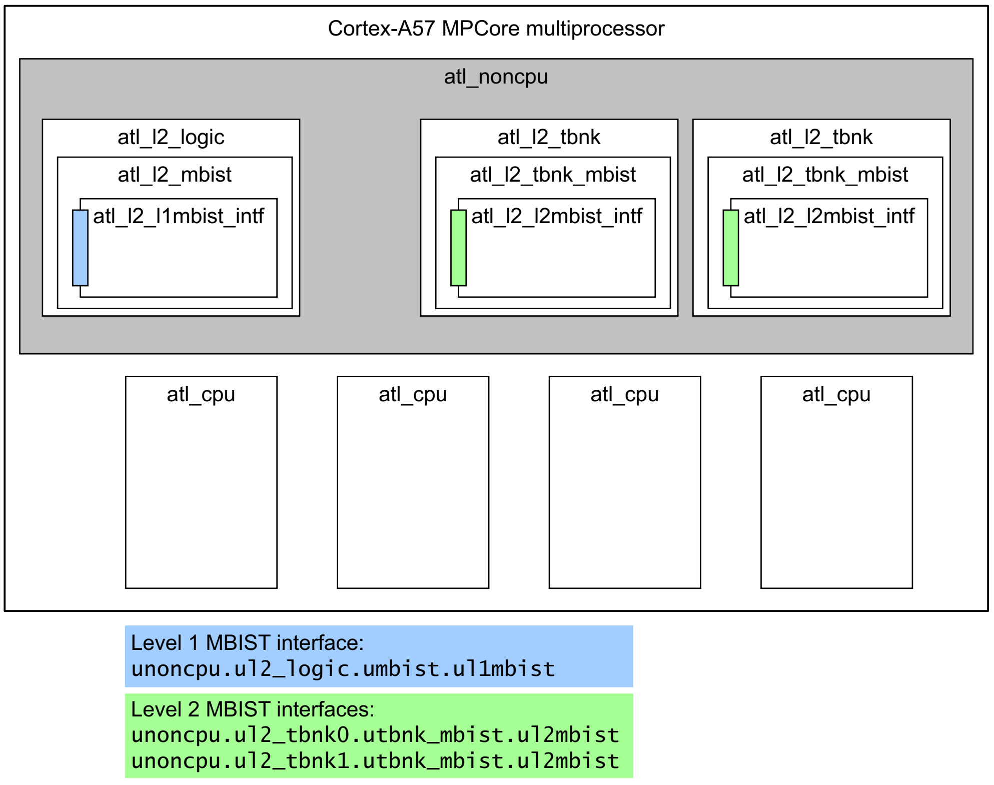


Figure 3-5 MBIST interface hierarchy

The L2 tag banks and the L1 caches can be tested in parallel by inserting a separate MBIST controller for each MBIST interface. This achieves minimal MBIST test time for the multiprocessor.

The following sections describe the MBIST interfaces:

- *MBIST interface signals* on page 3-9.
- *Arrays* on page 3-15.
- *MBIST test duration* on page 3-20.
- *MBIST interface timing* on page 3-21.
- *ALL mode* on page 3-27.

3.3.1 MBIST interface signals

Table 3-2 lists the global MBIST signals that are common to all three MBIST interfaces.

Table 3-2 Global signals for the MBIST interfaces

Signal	Direction	Description
MBISTREQ	Input	MBIST enable. See *MBISTREQ signal* on page 3-11
nMBISTRESET	Input	MBIST reset. See *nMBISTRESET signal* on page 3-12

Table 3-3 lists the L1 MBIST interface signals. Hierarchical references to the attachment points are provided as this is typically required for EDA MBIST insertion.

Table 3-3 L1 MBIST interface

Hierarchical port reference	Direction	Description
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTACK1**	Output	*MBISTACK signal* on page 3-12
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTADDR1[14:0]**	Input	*MBISTADDR signal* on page 3-12
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTARRAY1[5:0]**	Input	*MBISTARRAY signal* on page 3-13
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTBE1[7:0]**	Input	*MBISTBE signal* on page 3-12
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTCFG1**	Input	*MBISTCFG signal* on page 3-14
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTINDATA1[129:0]**	Input	*MBISTINDATA and MBISTOUTDATA signals* on page 3-12
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTOUTDATA1[129:0]**	Output	
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTREADEN1**	Input	*MBISTWRITEEN and MBISTREADEN signals* on page 3-12
`unoncpu.ul2_logic.umbist.ul1mbist.`**MBISTWRITEEN1**	Input	

Table 3-4 shows the L2 tag bank interface. The `atl_noncpu` block has two tag bank instances.

Table 3-4 L2 tag bank MBIST interfaces

Hierarchical port reference		Direction	Description
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTACK2**	Output	*MBISTACK signal* on page 3-12
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTACK2**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTADDR2[x:0]**	Input	*MBISTADDR signal* on page 3-12
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTADDR2[x:0]**		
The value *x* depends on the implemented L2 cache size: • *x*=15 for a 2MB cache. • *x*=14 for a 1MB cache. • *x*=13 for a 512KB cache.			
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTARRAY2[1:0]**	Input	*MBISTARRAY signal* on page 3-13
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTARRAY2[1:0]**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTBE2[17:0]**	Input	*MBISTBE signal* on page 3-12
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTBE2[17:0]**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTCFG2[4:0]**	Input	*MBISTCFG signal* on page 3-14
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTCFG2[4:0]**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTINDATA2[143:0]**	Input	*MBISTINDATA and MBISTOUTDATA signals* on page 3-12
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTINDATA2[143:0]**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTOUTDATA2[143:0]**	Output	
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTOUTDATA2[143:0]**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTREADEN2**	Input	*MBISTWRITEEN and MBISTREADEN signals* on page 3-12
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTREADEN2**		
Tag bank 0	`unoncpu.ul2_tbnk0.utbnk_mbist.ul2mbist.`**MBISTWRITEEN2**	Input	
Tag bank 1	`unoncpu.ul2_tbnk1.utbnk_mbist.ul2mbist.`**MBISTWRITEEN2**		

Table 3-5 describes the other multiprocessor signals that must be controlled for successful MBIST operation. All non-specified multiprocessor I/O can be assumed to be X. No redundancy I/O is present in the RTL for these interfaces. Any specialized memory control signals are added to the configured RTL before EDA MBIST insertion and before synthesis.

Table 3-5 Multiprocessor signals requiring control during MBIST test

Signal	Direction	Function	Comment
CLK	Input	Processor clock	MBIST controller must also run off this clock.
L2RSTDISABLE		Inhibits the L2 RAM reset	When requesting MBIST mode, tie this signal HIGH to prevent corruption of the L2 RAM data. This is necessary when performing a data retention test, when the processor is placed in a low-power state and the L2 logic is powered down but the RAMs are in a retention state. See *MBIST mode transitions* on page 3-21 for the MBIST reset procedure.
PCLKENDBG		Functional clock enable	Tie LOW for MBIST mode.
ATCLKEN			
ACLKENM			
ACLKENS			
CLKEN			Tie HIGH for MBIST mode.
nPRESETDBG		Functional reset	Tie HIGH for MBIST mode.
nCPUPORESET[3:0]			
nCORERESET[3:0]			
nL2RESET			
DFTSE		ATPG control signals	Tie LOW for MBIST mode.
DFTCLKBYPASS			
DFTCRCCLKDISABLE			
DFTL2CLKDISABLE			
DFTMCPHOLD			
DFTRAMHOLD			
DFTRSTDISABLE			

MBISTREQ signal

This input signal requests the multiprocessor to enter MBIST mode. This signal controls and enables all three internal MBIST interfaces.

MBISTREQ must remain HIGH throughout the entire MBIST test.

MBISTREQ LOW is the normal system functional mode.

nMBISTRESET signal

To move the multiprocessor into MBIST mode, **nMBISTRESET** must be LOW for a minimum of 16 **CLK** cycles. When the tester releases reset by setting **nMBISTRESET** HIGH, if the multiprocessor is in MBIST mode then it sets **MBISTACK** HIGH. See *MBIST mode transitions* on page 3-21 for the required start-up sequence. The **nMBISTRESET** signal controls all three internal MBIST interfaces.

All functional resets must be driven to their inactive state during MBIST mode.

MBISTACK signal

If **MBISTREQ** goes HIGH and the multiprocessor MBIST pipeline is in MBIST mode then **MBISTACK** goes HIGH.

After **MBISTACK** asserts, there must be a minimum of four **CLK** cycles before any read or write array operation is sent to the multiprocessor. This delay enables the **MBISTARRAY** and **MBISTCFG** settings to propagate into the pipeline.

MBISTADDR signal

This bus is a right-justified logical address that is sent to the RAM array under test. See:

- Table 3-10 on page 3-16 for the L1 MBIST interface **MBISTADDR** assignments.
- Table 3-12 on page 3-18 for the L2 MBIST interfaces **MBISTADDR** assignments.

MBISTINDATA and **MBISTOUTDATA** signals

The **MBISTINDATA** bus provides incoming write data to the array under test. The write data includes raw ECC bits.

The **MBISTOUTDATA** bus returns read data from the array under test. The read latency depends on the logical array and its position in the pipeline. Table 3-10 on page 3-16 shows the pipeline depth for the L1 MBIST interface and Table 3-13 on page 3-18 shows the pipeline depth for the L2 MBIST interfaces.

MBISTBE signal

This signal allows you to independently control bit and byte write enables for the array under test. In the case of large bit enable RAMs, the byte enable width is repeated to support the entire bit width.

Table 3-10 on page 3-16 shows the **MBISTBE** assignments for the L1 MBIST interface and Table 3-12 on page 3-18 shows the **MBISTBE** assignments for the L2 MBIST interfaces.

MBISTWRITEEN and **MBISTREADEN** signals

These signals provide additional controls for the write and read RAM operations. When:

- **MBISTWRITEEN** is HIGH, a write operation occurs and the associated **MBISTDATAOUT** for the transaction is unknown.
- **MBISTREADEN** is HIGH, a read operation occurs and the write data for the transaction is ignored.

No operation occurs if both signals are LOW, and it is illegal for both signals to be HIGH.

MBISTARRAY signal

This signal selects the logical array to test. Table 3-6 shows the **MBISTARRAY1[5:0]** encodings for the L1 RAM arrays and Table 3-7 shows the **MBISTARRAY2[1:0]** encodings for the L2 tag bank arrays.

Table 3-6 MBISTARRAY1[5:0] encoding for the L1 MBIST interface

MBISTARRAY1 bits	Value	Description
[5:4]	`0b00`	Processor 0
	`0b01`	Processor 1
	`0b10`	Processor 2
	`0b11`	Processor 3
[3:0]	`0b0000`	LS Tag array
	`0b0001`	LS Data array
	`0b0010`	IF Data array
	`0b0011`	IF Tag array
	`0b0100`	IF BTB array
	`0b0101`	IF GHB array
	`0b0110`	L2 TLB array
	`0b0111`	IF IP array
	`0b1000`	L2 PSQ array
	`0b1000` - `0b1111`	Reserved

Table 3-7 MBISTARRAY2[1:0] encoding for the L2 MBIST interfaces

MBISTARRAY2[1:0] value	Description
`0b00`	L2 Tag array
`0b01`	L2 Dirty array
`0b10`	L2 Snoop Tag array
`0b11`	L2 Data array

A logical array might consist of one or more RTL logical arrays, each of which might consist of one or more physical arrays. Usually a small array consists of a single physical array and larger logical arrays consist of multiple physical arrays.

MBISTARRAY must not change state when valid MBIST transactions are in the MBIST pipeline.

When **MBISTARRAY** changes value, there must be a minimum of four **CLK** cycles before any read or write array operation is sent to the multiprocessor.

MBISTCFG signal

This bus contains miscellaneous configuration signals that are specific to a particular MBIST interface.

For the L1 MBIST interface, **MBISTCFG1** is a single-bit width signal that controls the ALL mode. See *ALL mode access for L1 arrays* on page 3-27 for more information.

For the two L2 MBIST interfaces, **MBISTCFG2** is a multi-bit width signal that controls the ALL mode, the setup value, and the number of **CLK** cycles for a multi-cycle RAM. See Table 3-8.

Table 3-8 MBISTCFG2 for the L2 MBIST interfaces

Signal	Description
MBISTCFG2[4]	ALL signal. See *ALL mode access for L2 arrays* on page 3-28 for more information.
MBISTCFG2[3]	Setup value for the RAM. See *Setup timing* on page 3-25 for more information.
MBISTCFG2[2:0]	Number of cycles for multi-cycle RAMs. See Table 3-13 on page 3-18 for the allowed values and *Multi-cycle RAM MBIST operation* on page 3-24 for more information.

The **MBISTCFG2[3:0]** signals only apply to the L2 Tag and L2 Data arrays. The L2 Dirty and L2 Snoop Tag arrays ignore these signals because they are single-cycle RAMs.

3.3.2 Arrays

This section describes the logical arrays that each MBIST interface can access:

- *L1 arrays*.
- *L2 arrays* on page 3-17.

L1 arrays

Table 3-9 shows each L1 logical array and its hierarchical reference in the multiprocessor. This is useful for correctly mapping redundant elements in your MBIST solution.

Table 3-9 L1 arrays RTL hierarchical reference

Array name	Module name	Hierarchical reference
LS_TAG	`atl_ls_tag_ram`	`ucpuN.uls.utag_arr.utag_ram_bnk[0123]`
L2_PSQ	`atl_l2_psq_ram`	`unoncpu.ul2_logic.uprefetch.ucpuN_pf_logic.ucpu_psq_ram`
LS_DATA	`atl_ls_data_ram`	`ucpuN.uls.udata_arr.udata_ram_bnk[0123]`
IF_DATA	`atl_if_data_ram`	`ucpuN.uif.udata_arr.udata_ram_w[012]b[0123]_[lh]`
IF_TAG	`atl_if_tag_ram`	`ucpuN.uif.utag_arr.utag_ram_b[01]`
IF_BTB	`atl_if_btb_ram`	`ucpuN.uif.ubtb_arr.ubtb_b[0123]`
IF_GHB	`atl_if_ghb_ram`	`ucpuN.uif.ughb_arr.{uselstr_b[01], useldir_b[01], utstr_b[01], utdir_b[01], untstr_b[01], untdir_b[01]}`
L2_TLB	`atl_l2_tlb_ram`	`ucpuN.ul2_cpu_slv.usram_l2_tlb_[0123].usram`
IF_IP	`atl_if_ip_ram`	`ucpuN.uif.uip_arr.uip_ram_w[01]`

In Table 3-9:

`ucpuN`	*N*==0, 1, 2, or 3 and represents a processor within the multiprocessor. The **MBISTARRAY1[5:4]** bits select the processor number, see Table 3-6 on page 3-13.
`w[01]`, `w[012]`	`w` represents the way and the numbers within the brackets indicate the possible values. The Way column in Table 3-10 on page 3-16 shows which **MBISTADDR1** bits select the way number.
`b[01]`, `b[0123]`	`b` represents the bank and the numbers within the brackets indicate the possible values. The Bank column in Table 3-10 on page 3-16 shows which **MBISTADDR1** bits select the bank number.
`{ }`	Braces bound multiple RTL memory instances that share the MBIST interface data width and are simultaneously accessed.

Table 3-10 on page 3-16 shows the address and data mapping for the L1 logical arrays.

Table 3-10 Address and data mapping for the L1 MBIST interface

Array name	MBISTADDR1 bits			Data map	BE map	Pipeline depth[a]	Total cycles[b]
	Bank	Way	Index				
L2_PSQ	-	-	[4:0]	[51:0]	-	6	7
LS_TAG	[8:7]	[6]	[5:0]	[39:0] with ECC [32:0] without ECC	-	11	12
LS_DATA	[12:11]	[10]	[7:0][c]	[38:0] with ECC [31:0] without ECC	-		
IF_DATA[d]	[12:11,8][e]	[10:9]	[7:0]	[71:0]	-		
IF_TAG[d]	[9]	[8:7]	[6:0]	[35:0]	-		
IF_BTB	[10:9]	-	[8:0]	[85:0]	-		
IF_GHB[f]	[9]	-	[8:0]	[47:0]	{6{[7:0]}}		
IF_IP	-	[8]	[7:0]	[63:0]	-		
L2_TLB	[9:8]	-	[7:0]	[129:0]	{2{[1]}, 128{[0]}}		

a. Indicates the pipeline register depth.

b. The number of **CLK** cycles required before **MBISTOUTDATA1** is available.

c. [9:8] is the word select within the row.

d. IF_DATA and IF_TAG have three ways and the permitted Way values are `0b00`, `0b01`, and `0b10`. Both arrays are independently serviced in a single logical SRAM. MBIST tools might require the ways to be listed as most significant address bits to prevent non-continuous address spacing.

e. [8] selects between the high and low half of bank selected by bits[12:11].

f. IF_GHB consists of multiple logical arrays that are accessed and tested in parallel. All arrays must have the same RAM compiler physical construction to ensure physically oriented MBIST testing is accurate and of the highest quality.

L2 arrays

Table 3-11 shows each L2 logical array and its hierarchical reference in the multiprocessor.

Table 3-11 L2 tag bank arrays RTL hierarchical reference

Array name	Module name	Hierarchical reference	
L2_TAG	`atl_l2_tag_ram`	**Tag bank 0**	`unoncpu.ul2_tbnk0.usram_l2_tbnk_[01,23,45,67,89,ab,cd,ef]`
		Tag bank 1	`unoncpu.ul2_tbnk1.usram_l2_tbnk_[01,23,45,67,89,ab,cd,ef]`
L2_DTY	`atl_l2_dirty_ram`	**Tag bank 0**	`unoncpu.ul2_tbnk0.usram_l2_tbnk_dirty`
		Tag bank 1	`unoncpu.ul2_tbnk1.usram_l2_tbnk_dirty`
L2_SNP	`atl_l2_snp_tag_ram`	**Tag bank 0**	`unoncpu.ul2_logic.usram_l2_tbnk0_snp_`*`N`*
		Tag bank 1	`unoncpu.ul2_logic.usram_l2_tbnk1_snp_`*`N`*
		Where *N*==0, 1, 2, or 3 and represents a processor within the multiprocessor.	
L2_DATA	`atl_l2_data_ram`	**Tag bank 0**	`unoncpu.ul2_tbnk0.usram_l2_tbnk_dbnk[0,1,2,3]`
		Tag bank 1	`unoncpu.ul2_tbnk1.usram_l2_tbnk_dbnk[0,1,2,3]`

In Table 3-11:

`tbnk_[01,23,45,67,89,ab,cd,ef]`

Represents one of eight Tag RAM instances. In Table 3-12 on page 3-18, the INST identifier selects a Tag RAM as follows:

- INST=`0b000`, selects `tbnk_01`.
- INST=`0b001`, selects `tbnk_23`.
- INST=`0b010`, selects `tbnk_45`.
- INST=`0b011`, selects `tbnk_67`.
- INST=`0b100`, selects `tbnk_89`.
- INST=`0b101`, selects `tbnk_ab`.
- INST=`0b110`, selects `tbnk_cd`.
- INST=`0b111`, selects `tbnk_ef`.

`dbnk[0,1,2,3]` `dbnk` represents a data bank and the numbers within the brackets indicate the possible values. Table 3-12 on page 3-18 shows which **MBISTADDR2** bits select the data bank number.

Table 3-12 shows the address and data mapping for the L2 logical arrays.

Table 3-12 Address and data mapping for the L2 MBIST interfaces

Array name	MBISTADDR2 bits, when the L2 cache size is:			Data width	BE map
	2MB	1MB	512KB		
L2_TAG	[12:10]=INST [9:0]=index	[11:9]=INST [8:0]=index	[10:8]=INST [7:0]=index	[81:0]	{41{[1]}, 41{[0]}}
L2_DTY	[11:10]=DWsel [9:0]=index	[10:9]=DWsel [8:0]=index	[9:8]=DWsel [7:0]=index	[47:0]	{8{[7]}, 4{[6]}, 8{[5]}, 4{[4]}, 8{[3]}, 4{[2]}, 8{[1]}, 4{[0]}}
L2_SNP	[8:7]=Processor number [6:0]=index			[79:0]	{40{[1]}, 40{[0]}}
L2_DATA	[15:14]=DBNK [13:0]=index	[14:13]=DBNK [12:0]=index	[13:12]=DBNK [11:0]=index	[143:0]	{8{[17]}, 8{[16]}, 8{[15]}, 8{[14]}, 8{[13]}, 8{[12]}, 8{[11]}, 8{[10]}, 8{[9]}, 8{[8]}, 8{[7]}, 8{[6]}, 8{[5]}, 8{[4]}, 8{[3]}, 8{[2]}, 8{[1]}, 8{[0]}}

Table 3-13 shows how the number of implemented register slices and the state of the **MBISTCFG2** signal affects the latency for the L2 RAMs.

Table 3-13 MBISTCFG2 settings for the L2 tag bank arrays

Array name	Slice[a]	Pipeline depth	MBISTCFG2[3:0]	Read or write transaction rate, cycles/operation	Total cycles[b]

L2_TAG	0	8	`0b0001`
			`0b0010`
			`0b0011`
			`0b0100`
				2
				3
				4
				5
					10-13

			`0b1001`
			`0b1010`
			`0b1011`
				3
				4
				5
					11-13

	1	10	`0b0001`
			`0b0010`
				2
				3
					12-13

			`0b1010`	4	13
L2_DTY	0	8	-	1	9
	1	10	-		11
L2_SNP	0	6	-		7
	1	8	-		9

Table 3-13 MBISTCFG2 settings for the L2 tag bank arrays (continued)

Array name	Slice[a]	Pipeline depth	MBISTCFG2[3:0]	Read or write transaction rate, cycles/operation	Total cycles[b]
L2_DATA	0	9	`0b0001`	2	11-17
			`0b0010`	3	
			`0b0011`	4	
			`0b0100`	5	
			`0b0101`	6	
			`0b0110`	7	
			`0b0111`	8	
			`0b1001`	3	12-17
			`0b1010`	4	
			`0b1011`	5	
			`0b1100`	6	
			`0b1101`	7	
			`0b1110`	8	
	1	11	`0b0001`	2	13-17
			`0b0010`	3	
			`0b0011`	4	
			`0b0100`	5	
			`0b0101`	6	
			`0b1001`	3	14-17
			`0b1010`	4	
			`0b1011`	5	
			`0b1100`	6	
	2	13	`0b0001`	2	15-17
			`0b0010`	3	
			`0b0011`	4	
			`0b1001`	3	16-17
			`0b1010`	4	

a. The number of register slices that were added during implementation. The implementation script forces the Tag, Dirty, and Snoop Tag RAMs to have the same number of register slices. Each slice adds two pipeline stages. See *L2 RAM slice timing* on page 3-26 for more information.

b. The number of **CLK** cycles required before **MBISTOUTDATA2** is available.

3.3.3 MBIST test duration

The test duration is the number of clock cycles required to touch all memory entries once, that is, a 1N pattern in MBIST terminology. However, actual test patterns require more cycles, and are under user control. Various cache size configurations exist for the component array sizes.

The manufacturing process typically uses PLL-based MBIST testing because this performs multiple test cycles for each reference clock. This section ignores such optimizations because the test environment and target frequency are unknown.

The multiprocessor has two equivalent L2 tag bank interfaces and one L1 processor memory MBIST interface. The quoted test durations assume that each interface has its own MBIST controller and that the tests for each interface are performed simultaneously.

Some of the multiprocessor RAMs require multiple system clock cycles for each read or write operation. This analysis assumes that the multi-cycle RAMs require two system clocks per operation. User implementations might vary, affecting the estimated cycle counts.

Therefore, the test duration depends on the RAM type as follows:

Standard RAMs

1N test cycles = number of addresses.

Latency or multi-cycle RAMs

1N test cycles = (**MBISTCFG2[2:0]**+1) × number of addresses.

Table 3-14 lists the 1N cycle count for the L1 arrays.

Table 3-14 L1 array MBIST test duration

L1 array	Addresses	Test width	1N cycle count
LS_TAG	512	40	0.5K
L2_PSQ	32	52	32
LS_DATA	8K	39	8K
IF_DATA[a]	8K	72	8K
IF_TAG[a]	1K	36	1K
IF_BTB	2K	86	2K
IF_GHB[b]	1K	48	1K
L2_TLB	1K	130	1K
IF_IP	512	64	0.5K
		L1 MBIST total cycles =	**22K** × N[c]

a. This RAM has three ways independently serviced in a single logical SRAM. MBIST tools might require the ways to be listed as most significant address bits to prevent non-continuous address spacing.

b. This RAM consists of numerous small logical arrays, accessed and tested in parallel. All arrays must have the same RAM compiler physical construction to ensure physically oriented MBIST testing is accurate and of the highest quality.

c. *N* is the number of processors that the multiprocessor implements.

Table 3-15 lists the 1N cycle count for the L2 arrays in a single tag bank. As the multiprocessor contains two MBIST tag bank interfaces then the test duration doubles if the tag banks are tested sequentially.

Table 3-15 L2 array MBIST test duration for a single tag bank

L2 array	Address space[a]	Test width	1N cycle count
L2_SNP	128	72	$128 \times N$[b]
L2_TAG	$(L2size/512) \times 4K$	82	$(L2size/512) \times 4K \times ($**MBISTCFG2[2:0]**$+1)$
L2_DATA	$(L2size/512) \times 16K$	144	$(L2size/512) \times 16K \times ($**MBISTCFG2[2:0]**$+1)$
L2_DTY	$(L2size/512) \times 1K$	144	$(L2size/512) \times 1K \times ($**MBISTCFG2[2:0]**$+1)$
		Total 1N cycles =	$128 \times N + ((L2size/512) \times 21K \times ($**MBISTCFG2[2:0]**$+1))$

a. *L2size* is 512, 1024, or 2048. The value is the L2 cache size, in KB, that the multiprocessor implements.

b. *N* is the number of processors that the multiprocessor implements.

3.3.4 MBIST interface timing

The following sections describe the MBIST interface timing:

- *MBIST mode transitions*.
- *Single-cycle RAM MBIST operation* on page 3-23.
- *Multi-cycle RAM MBIST operation* on page 3-24.
- *Setup timing* on page 3-25.
- *L2 RAM slice timing* on page 3-26.
- *ALL mode* on page 3-27.

MBIST mode transitions

Prior to performing an MBIST operation, the following conditions must apply:

- The **CLK** signal is stable and operating at the required test frequency.
- The signals that Table 3-5 on page 3-11 shows, are in the appropriate state.

It is optional whether **L2RSTDISABLE** is set HIGH during MBIST testing. When HIGH, the L2 RAMs are not reset during the MBIST reset sequence. This feature is useful when performing a low-power data retention test.

The following steps are required before any MBIST operations can be processed by the multiprocessor:

1. Set **nMBISTRESET** LOW.
2. Set **MBISTREQ** HIGH. It must remain asserted for the duration of the MBIST test.
3. For each active interface:
    - Set the **MBISTWRITEEN** and **MBISTREADEN** signals LOW.
    - Set the **MBISTCFG** and **MBISTARRAY** signals to a valid non-X value. This is to prevent pessimistic Verilog simulation failures.
4. After a minimum of 16 **CLK** cycles after step 1, set **nMBISTRESET** HIGH.
5. Wait for **MBISTACK** to go HIGH. The multiprocessor sets **MBISTACK** HIGH when that interface enters MBIST mode, and this occurs within 8 **CLK** cycles after step 3.

6. Set the **MBISTARRAY** and **MBISTCFG** signals to the values required to test the RAM.

7. Wait for a minimum of 4 **CLK** cycles. This delay enables the **MBISTARRAY** and **MBISTCFG** settings to propagate into the pipeline.

8. The MBIST controller sends MBIST operations to the multiprocessor, to test the RAM selected in step 6.

   The **MBISTARRAY** and **MBISTCFG** signals must not change state while there are still active operations in the MBIST pipeline.

9. To test another RAM, repeat steps 6-8.

10. When all required testing is complete:
    - Set **MBISTREQ** LOW.
    - Set the functional resets to a values that are suitable for when the multiprocessor exits MBIST mode.

11. After 2 **CLK** cycles the multiprocessor sets **MBISTACK** LOW, to indicate exit from MBIST mode.

12. If required, perform the processor reset sequence specified in the *ARM® Cortex®-A57 MPCore™ Processor Technical Reference Manual* to begin executing functional code on the processor. During functional operation, **MBISTREQ** must be LOW and **nMBISTRESET** must be HIGH.

Figure 3-6 shows the entry and exit process for MBIST mode.

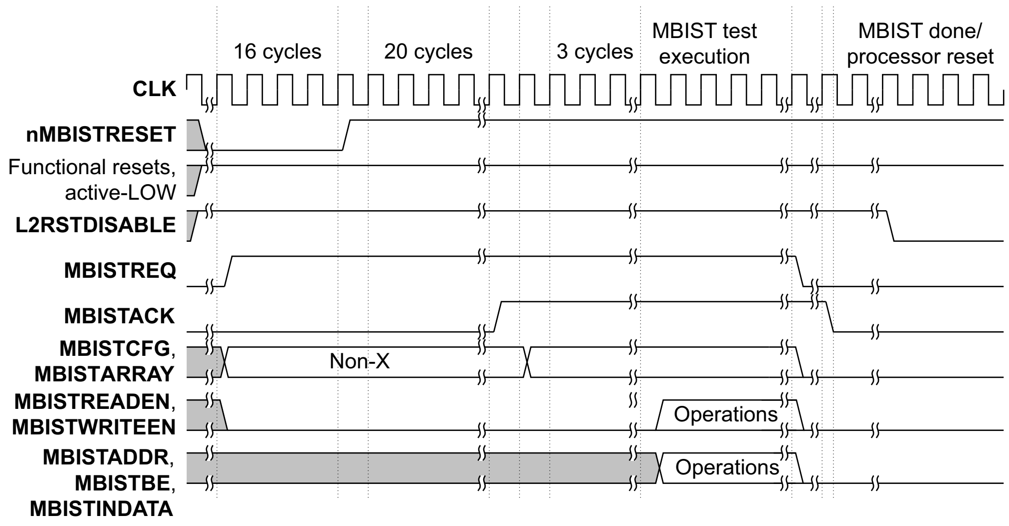


Figure 3-6 MBIST mode, entry and exit

Single-cycle RAM MBIST operation

Figure 3-7 shows the data timing for read and write MBIST operations on a single-cycle RAM in the LS Tag array. The timing is similar for the other arrays that use single-cycle RAM operations.

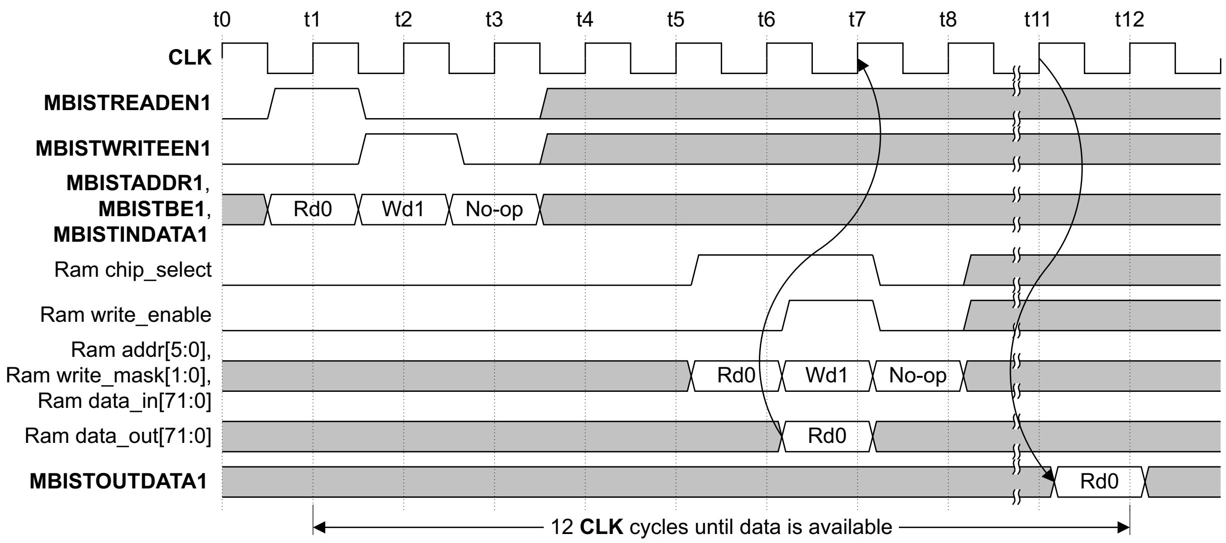


Figure 3-7 Single-cycle MBIST operation

In Figure 3-7:

- The signals with a Ram prefix are the signals at the boundary of the RAM integration level of the design, `*ram.v`.
- Rd<number> indicates a read operation.
- Wd<number> indicates a write operation.

MBIST read and write operations can be delivered on every **CLK** cycle. A *no-operation* (No-op), that is, when both **MBISTREADEN** and **MBISTWRITEEN** are LOW, is not required but is shown for completeness.

As Table 3-10 on page 3-16 specifies, the LS Tag array requires 12 **CLK** cycles for a read operation. Therefore, the MBIST controller must strobe **MBISTOUTDATA**, 12 **CLK** cycles after it sends the read operation to the multiprocessor.

Multi-cycle RAM MBIST operation

The L2 Tag and L2 Data arrays require several clock cycles to process a read or write operation. The MBIST tester must set the **MBISTCFG2[2:0]** signals to match the latency of the Tag and Data RAMs. See Table 3-13 on page 3-18 for information the **MBISTCFG2** settings. The tester must set the **MBISTCFG2** value, at least 4 **CLK** cycles before it sends the first MBIST operation to the L2 MBIST interface.

Figure 3-8 shows the timing for an L2 Tag array that uses 4 cycle RAM, so **MBISTCFG2[2:0]** = `0b011`. In this configuration, the MBIST controller can only send an MBIST operation, to the L2 MBIST interface, once every 4 **CLK** cycles. For each operation:

- The **MBISTREADEN2**, or **MBISTWRITEEN2**, signal must be HIGH for the first **CLK** cycle and then LOW for the remaining cycles.
- The **MBISTADDR2**, **MBISTBE2**, and **MBISTINDATA2** signals must remain constant, after their value is set.

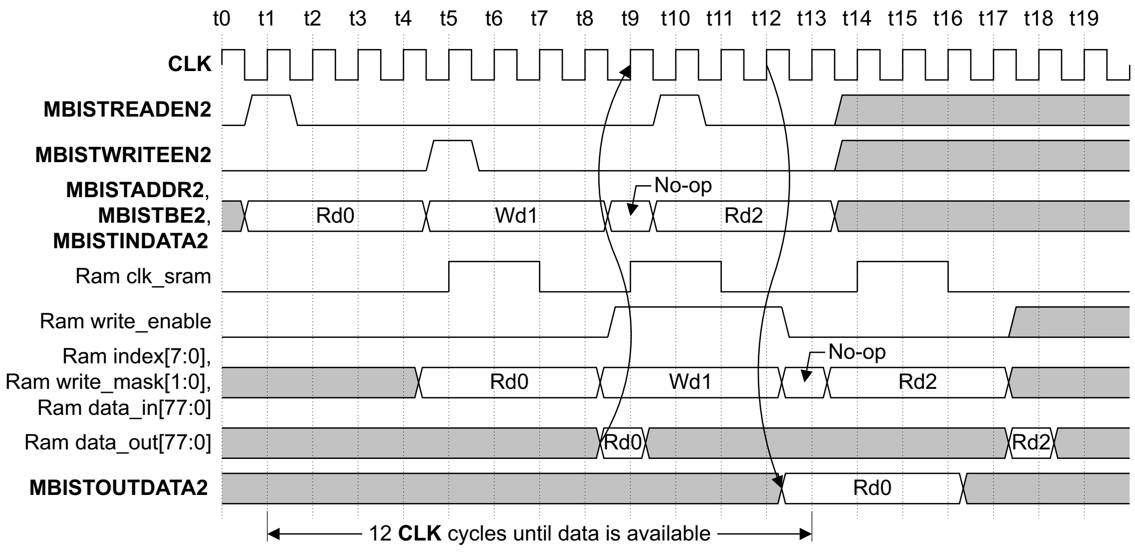

t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19
CLK
MBISTREADEN2
MBISTWRITEEN2
MBISTADDR2, MBISTBE2, MBISTINDATA2
Rd0
Wd1
No-op
Rd2
Ram clk_sram
Ram write_enable
Ram index[7:0], Ram write_mask[1:0], Ram data_in[77:0]
Rd0
Wd1
No-op
Rd2
Ram data_out[77:0]
Rd0
Rd2
MBISTOUTDATA2
Rd0
12 CLK cycles until data is available

Figure 3-8 Timing for 4-cycle RAM

Setup timing

The **MBISTCFG2[3]** signal selects the input setup time for the L2 Tag and L2 Data arrays. When **MBISTCFG2[3]** is:

LOW	Input setup time is 1 **CLK** cycle. See Figure 3-8 on page 3-24.
HIGH	Input setup time is 2 **CLK** cycles. See Figure 3-9.

The tester must set the **MBISTCFG2[3]** value, at least 4 **CLK** cycles before it sends the first MBIST operation to the L2 MBIST interface.

Figure 3-9 shows the timing for an L2 Tag array that:

- Uses 4-cycle RAM, so **MBISTCFG2[2:0]** = `0b011`.
- Has an input setup time of 2 **CLK** cycles because **MBISTCFG2[3]** is HIGH.

In this configuration, the MBIST controller can only send an MBIST operation, to the L2 MBIST interface, once every 5 **CLK** cycles.

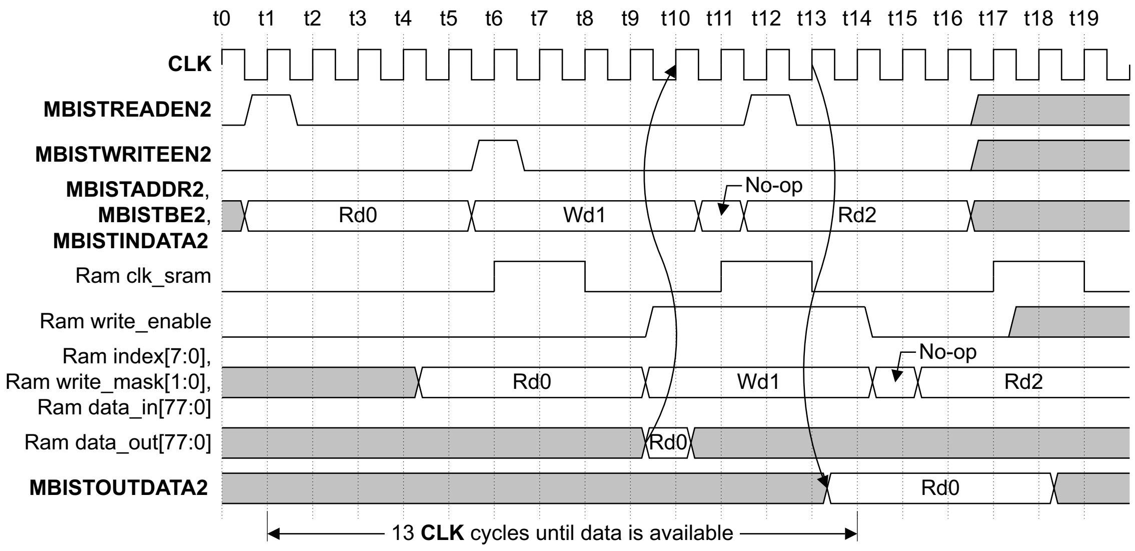


Figure 3-9 Timing for 4-cycle RAM with 1-cycle setup

L2 RAM slice timing

During implementation of the multiprocessor, there are options to control the number of register slices for the L2 Tag RAM and the L2 Data RAM. Each slice adds one pipeline stage on the inputs of the RAM and another pipeline stage to the outputs of the RAM. These pipeline stages are clocked by the **CLK** signal.

The stages are located within the RAM integration level of the design, `*ram.v`.

The Slice column in Table 3-13 on page 3-18, shows how the number of implemented register slices affects the clock latency of the multi-cycle RAMs.

Note

- The implemented Tag slice value controls the number of register slices for the Tag, Dirty, Snoop Tag, and PSQ arrays.
- The implemented Data slice value controls the number of register slices for the Data arrays.

Figure 3-10 shows the timing for an L2 Tag array that:

- Uses 2-cycle RAM, so **MBISTCFG2[2:0]** = `0b001`.
- Has an input setup time of 2 **CLK** cycles because **MBISTCFG2[3]** is HIGH.
- Is implemented with one register slice.

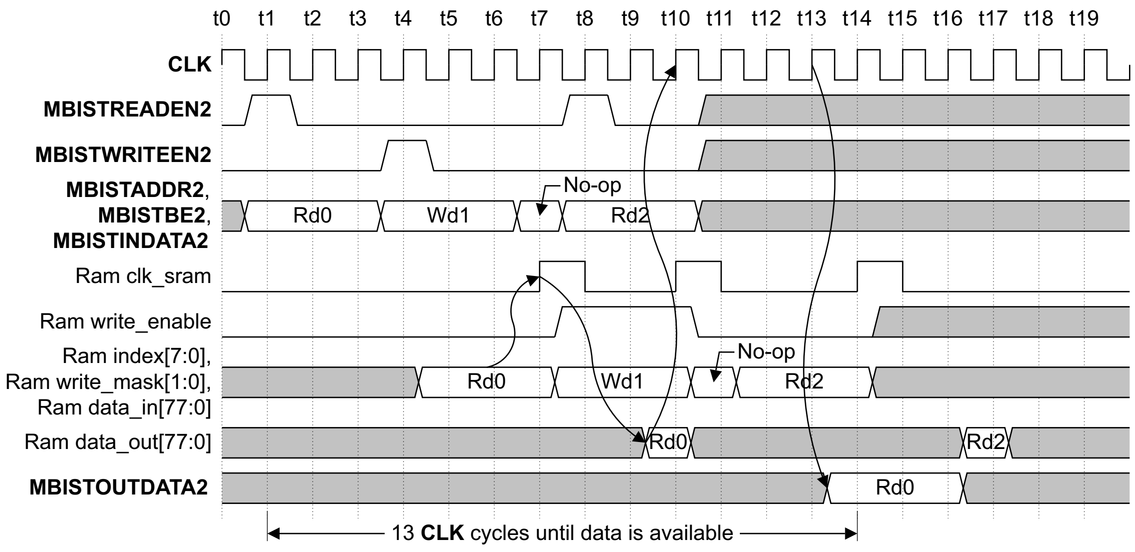


Figure 3-10 Timing for 2-cycle RAM with 1-cycle setup and one register slice

In Figure 3-10, the Ram* signals remain at the boundary of the RAM integration level, `*ram.v`. The inputs pass through a single level of registration before reaching the SRAM macro, and the outputs of the SRAM macro pass through a single level of registration before reaching the RAM data_out outputs. The clk_sram signal is the clock to the SRAM macro and it is delayed by one **CLK** cycle to compensate for the additional pipeline delay.

3.3.5 ALL mode

For each MBIST interface, an MBIST tester usually tests each RAM in sequence. However, this does not emulate the power footprint of the RAMs during functional mode, when accesses occur to multiple RAMs. Also, life testing or burn-in testing must simultaneously exercise most of the logic, without damaging the device. To satisfy these requirements, each MBIST interface can operate in the ALL mode, by setting the appropriate **MBISTCFG** bit as follows:

- To set the L1 MBIST interface into ALL mode, set **MBISTCFG** HIGH.
- To set an L2 MBIST interface into ALL mode, set **MBISTCFG[4]** HIGH.

The ALL mode performs simultaneously read and write accesses to multiple arrays. Table 3-16 shows how ALL mode affects read and write operations.

Table 3-16 How ALL mode affects read and write operations

MBISTWRITEEN	MBISTREADEN	ALL mode	Action
LOW	HIGH	Disabled	Reads from targeted array only.
HIGH	LOW		Writes to targeted array only.
LOW	HIGH	Enabled	Reads from all arrays but outputs data only from the targeted array.
HIGH	LOW		Writes to all arrays, unless the target array consists of multiple logical RAMs. For more information see: • *ALL mode access for L1 arrays*. • *ALL mode access for L2 arrays* on page 3-28.

ALL mode access for L1 arrays

When ALL mode is disabled, each array of the L1 interface is tested serially within each processor and the processors are also tested serially.

When ALL mode is enabled during a read operation, all arrays are read but only the targeted array read data is output on the **MBISTOUTDATA1** bus. The ALL mode setting is applied to all of the implemented processors.

When ALL mode is enabled during a write operation, all arrays are written, unless the targeted array consists of multiple logical RAMs.

If an array consists of multiple logical RAMs then only the addressed logical RAM is written, and the remaining RAMs in the array are read. This ensures that during ALL mode testing, the MBIST algorithm still performs correctly through the entire address space of the target array. For the logical RAMs from all of the non-targeted arrays, if these are simultaneously accessible during functional mode, then the algorithm writes to the arrays but not necessarily on the same **CLK** cycle because of differences in the various MBIST pipelines.

Figure 3-11 on page 3-28 shows an example of an ALL mode write to a L1 MBIST interface.

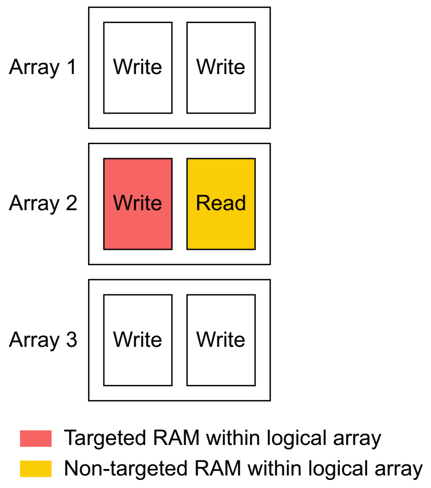

Array 1
Write
Write
Array 2
Write
Read
Array 3
Write
Write
Targeted RAM within logical array
Non-targeted RAM within logical array

Figure 3-11 ALL mode write to an L1 MBIST interface

In Figure 3-11, array 2 is the target array and it contains two logical RAMs. During an ALL mode write, the RAM that is addressed is written and the RAM that is not addressed is read. Array 1 and array 3 are also written. This preserves the data integrity but still emulates the power usage when all of the arrays are written during functional mode.

ALL mode access for L2 arrays

The L2 arrays are contained in two tag banks. Each tag bank contains four logical arrays and has its own MBIST interface.

When ALL mode is disabled, each array of the L2 interface is tested serially.

When ALL mode is enabled during a read operation, all arrays are read but only the targeted array read data is output on the **MBISTOUTDATA2** bus.

When ALL mode is enabled during a write operation, all arrays are written, unless the targeted array consists of multiple logical RAMs.

If an array consists of multiple logical RAMs then only the addressed logical RAM is written, and the remaining RAMs in the array are read. This ensures that during ALL mode testing, the MBIST algorithm still performs correctly through the entire address space of the target array. For the logical RAMs from all of the non-targeted arrays, if these are simultaneously accessible during functional mode, then the algorithm writes to the arrays but not necessarily on the same **CLK** cycle because of differences in the various MBIST pipelines.

Chapter 4
Integration Kit

This chapter is an overview of the Cortex-A57 MPCore multiprocessor DAP-Lite *Integration Kit* (IK). It describes how to use the kit to verify correct integration of the multiprocessor and DAP-Lite. It contains the following sections:

4.1 About the DAP-Lite integration kit

The *Debug Access Port* (DAP)-Lite is an implementation of an *ARM Debug Interface version 5* (ADIv5) comprising a number of components supplied in a single configuration. The DAP-Lite contains the following components:

- *Serial Wire JTAG Debug Port* (SWJ-DP).
- *Advanced Peripheral Bus Access Port* (APB-AP).
- *Advanced Peripheral Bus Multiplexer* (APB-Mux).
- CoreSight *Read Only Memory* (ROM) table.

The DAP-Lite IK is a configurable environment to demonstrate correct integration of the Cortex-A57 MPCore multiprocessor in a system using DAP-Lite as a simple debug solution. The DAP-Lite IK includes a portable integration test and verification components designed to check the integration of an Cortex-A57 MPCore multiprocessor with the DAP-Lite.

Note

The DAP-Lite IK does not demonstrate correct integration with a complex debug solution or any form of ETM trace. Design and integration of a more sophisticated debug solution for the Cortex-A57 MPCore multiprocessor requires use of the ARM CoreSight SoC-400 product. See Chapter 5 *CoreSight Trace and Debug Integration* for more information.

The following sections describes how to compile and run the DAP-Lite integration test on the multiprocessor in the Cortex-A57 *Execution Testbench* (`execution_tb`) environment.

Note

ARM recommends you port this test to your SoC testbench.

The Cortex-A57 DAP-Lite IK includes:

- Portable DAP-Lite integration test code, supplied as ARM assembler and DAP Macro Language source, to check the integration of the multiprocessor with the DAP-Lite.
- Components required for testing DAP-Lite integration, including:
  - *SerialWire/JTAG Interface Manager* (SWJIM).
  - *ARM Boundary Scan Testbox* (armBST).
- The `Makefile` that generates ELF, HEX, and BSI stimulus files for the multiprocessor and the SWJIM.
- The `Makefile.ik` that manages compiling the testbench and running simulations.
- Perl scripts that assist in the generation of HEX and BSI stimulus files for the SWJIM.
- Documents for the SWJIM, armBST, and DAP Macro Language that you can use as reference for the DAP-Lite integration test.

4.2 Test overview

The IK contains the following test to check integration of the multiprocessor and DAP-Lite:

`cs_daplite.s`	This test checks the integration of DAP-Lite with the Cortex-A57 MPCore multiprocessor. This test can be built to run using either JTAG or *SerialWire Debug* (SWD) as the external debug interface protocol.

4.3 Testbench directory structure

Figure 4-1 shows the DAP-Lite IK and the Cortex-A57 MPCore multiprocessor execution testbench directory structure.

```
logical/
  └─testbench/
       ├─integration_kit_daplite/
       │    ├─doc/
       │    ├─logs/
       │    ├─tests/
       │    ├─tools/
       │    └─verilog/
       └─execution_tb/
            ├─sim/
            ├─tests/
            └─verilog/
```

Figure 4-1 Testbench directory structure

4.4 Execution testbench structure

The Cortex-A57 MPCore multiprocessor execution testbench is used to run the DAP-Lite IK test simulations. Figure 4-2 shows the structure of the execution testbench.

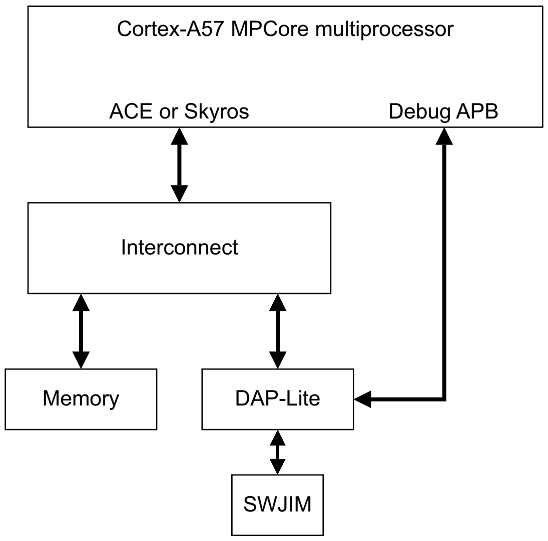


Figure 4-2 Testbench structure

Note

See *Configuring DAP-Lite RTL* on page 4-9 for how to install the DAP-Lite RTL into the execution testbench environment.

4.5 Testbench memory map

The execution testbench has the following memory maps:

- *System memory map*.
- *Debug APB memory map* on page 4-7.

4.5.1 System memory map

Figure 4-3 shows the system memory map for an ACE interface and the system memory map for a CHI interface.

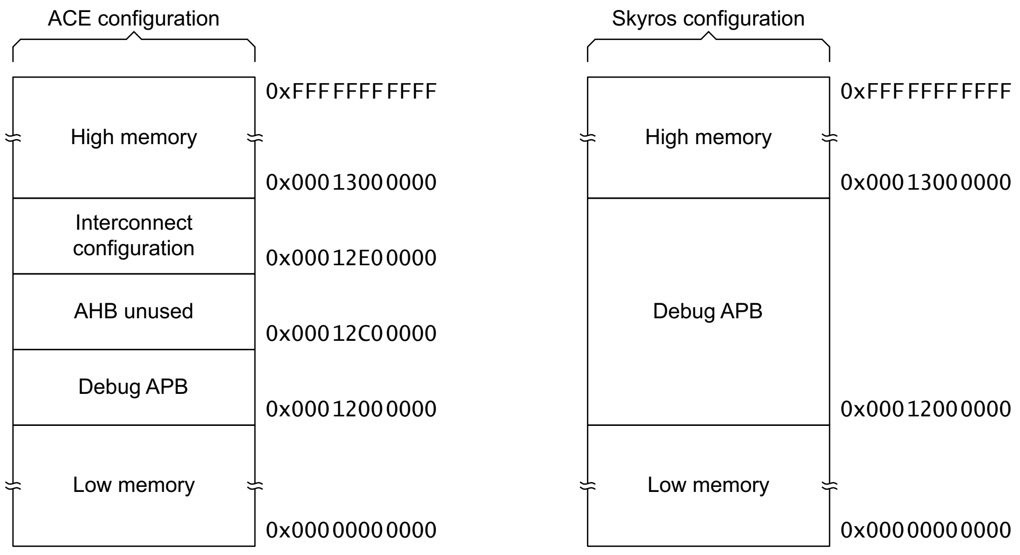


Figure 4-3 System memory map

Note

For a multiprocessor with an ACE interface, the `execution_tb` sets the state of the **SYSBARDISABLE**, **BROADCASTINNER**, **BROADCASTOUTER**, and **BROADCASTCACHEMAINT** signals so that the ACE interface operates in AXI3 mode.

4.5.2 Debug APB memory map

Figure 4-4 shows the debug APB memory map.

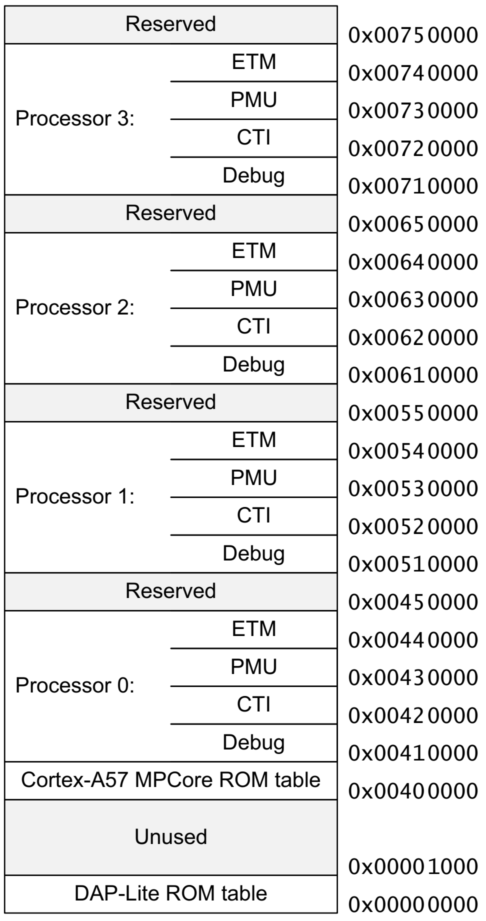


Figure 4-4 Debug APB memory map

Note

The debug address differs between Figure 4-3 on page 4-6 and Figure 4-4 because one is the system memory map view and the other is the view from the debug interface.

4.6 Design flow

This section describes:

- *Integration kit setup*.
- *Common environment setup*.
- *Configuring DAP-Lite RTL* on page 4-9.
- *Building the test* on page 4-9.
- *Compiling and running scripts* on page 4-10.
- *Example test output* on page 4-13.

The DAP-Lite IK enables you to:

1. Run the supplied DAP-Lite integration test in the execution testbench. This demonstrates DAP-Lite to Cortex-A57 MPCore multiprocessor connectivity, familiarizes you with the test build flow, and provides a reference platform for debug when porting the integration test to your SoC testbench.
2. Port the supplied DAP-Lite integration test to your SoC testbench environment to check for correct integration of the DAP-Lite in the actual target system. This typically involves code modification for items such as system specific memory maps, and involves reusing the IK verification components, such as the SWJIM, as required to support the integration test methodology.

4.6.1 Integration kit setup

The DAP-Lite IK involves running the integration test in the existing execution testbench environment rather than providing a separate testbench environment. You can run the testbench on the RTL with either the default supplied RAM models or the physical RAMs integrated following the RAM integration procedures as described in the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide*.

You must ensure that you can run ARM supplied standard test vectors in the testbench environment, as described in the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* before continuing with this integration. When this has been verified, you can proceed with the DAP-Lite IK test.

The DAP-Lite IK supports Synopsys VCS, Cadence Incisive, and Mentor QuestaSim on 32-bit and 64-bit platforms. The instructions are simulator or machine specific and the appropriate simulator and machine specific commands are given explicitly.

4.6.2 Common environment setup

Begin the common environment setup by setting the `$PROJ_LOGICAL` environment variable to your configured RTL.

```
> setenv PROJ_LOGICAL {full_path_to_configured_rtl}
```

If you require the model to support Tarmac trace then you must add the Protocol Buffers to your simulation environment. See the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* for information about Tarmac trace and how to add the Protocol Buffers to your environment.

If your multiprocessor implements the CHI interface then the *Universal Verification Methodology* (UVM) Class Library Code must be present. See the *ARM® Cortex®-A57 MPCore™ Processor Configuration and Sign-off Guide* for information about how to add UVM to your simulation environment.

Depending on your platform, you set up the required variables for ModelManager and armBST as follows:

32-bit platform

```
> setenv MG_LIB ${PROJ_LOGICAL}/testbench/integration_kit_daplite/verilog/armBST/ModelManager/Linux/MM
> setenv DIR_armBST ${PROJ_LOGICAL}/testbench/integration_kit_daplite/verilog/armBST/RH_Linux_x86
```

64-bit platform

```
> setenv MG_LIB ${PROJ_LOGICAL}/testbench/integration_kit_daplite/verilog/armBST/ModelManager/Linux_64/MM
> setenv DIR_armBST ${PROJ_LOGICAL}/testbench/integration_kit_daplite/verilog/armBST/RH_Linux_x86_64
```

4.6.3 Configuring DAP-Lite RTL

To configure the DAP-Lite RTL you must ensure that the DAP-Lite RTL is present. The *ARM® Cortex®-A57 MPCore™ Processor Release Note* describes how to copy the DAP-Lite RTL into the `execution_tb/verilog` area.

Configure the DAP-Lite RTL by editing the DAP-Lite ROM table, `DapRomDefs.v`, to match the example system. Edit `$PROJ_LOGICAL/testbench/execution_tb/verilog/daprom/verilog/DapRomDefs.v`.

Change the definition of `ROMENTRY00` as follows:

```
`define ROMENTRY00 {20'h00400, 10'h000, `ROM_FORMAT, `ROM_ENTRY} // Cortex-A57 ROM table
```

Comment out definition of `ENABLE_32` as follows:

```
//`define ENABLE_32    // Uncomment for 25 to 32 entries
```

Note

- This configuration matches the example system in the `execution_tb` environment. You might require a different ROM table configuration depending on the Debug APB memory map of your SoC.
- This configuration uses the default Peripheral ID values for the DAP-Lite ROM table. You must modify the `ROM_PERIPHIDn_VAL` values for the ROM table to reflect your implementation, for example, Manufacturer ID, customer assigned part, number, and revision.

See the *ARM® CoreSight™ DAP-Lite Technical Reference Manual* and the *ARM® CoreSight™ Components Implementation Guide*, that are supplied in the Cortex-A57 MPCore multiprocessor documentation bundle, for more information about the configuration of the ROM table and Peripheral IDs.

4.6.4 Building the test

To build the DAP-Lite test, you must set up the tools to build the code as follows:

```
> setenv PATH $PROJ_LOGICAL/testbench/integration_kit_daplite/tools/:$PATH
> setenv CS_TOOLS_HOME $PROJ_LOGICAL/testbench/integration_kit_daplite/tools/
> cd $PROJ_LOGICAL/testbench/integration_kit_daplite/tests/
```

Edit the `Makefile` to match your configuration. The instructions are in the `Makefile`.

The `Makefile` enables you to build the single DAP-Lite test program so that program supports the following debug modes:

JTAG	Use this when the external debug interface is JTAG. The `Makefile` creates files that use the `cs_daplite_jtag` prefix.
SerialWire Debug	Use this when the external debug interface is SerialWire Debug. The `Makefile` creates files that use the `cs_daplite_sw` prefix.

Note

The execution testbench supports running the test in either mode.

Ensure that the ARM RVCT tools are in the execution path.

Run `make`. This writes `cs_daplite_`*`mode`*`.elf`, `cs_daplite_`*`mode`*`.bsi`, and various `cs_daplite_`*`mode`*`.hex` files to the `bin` directory, where *`mode`* is either `jtag` or `sw` depending on the `Makefile` options.

```
> make
```

Copy the `Makefile.ik` into the `execution_tb/sim` directory, and change to that directory as follows:

```
cp $PROJ_LOGICAL/testbench/integration_kit_daplite/Makefile.ik $PROJ_LOGICAL/testbench/execution_tb/sim/
cd $PROJ_LOGICAL/testbench/execution_tb/sim/
```

4.6.5 Compiling and running scripts

To compile and run a test, use the command format for your Verilog simulator. For information about the command format required by a specific Verilog simulator see the examples in:

- *How to compile a build*.
- *How to run the test when the multiprocessor implements an ACE interface* on page 4-12.
- *How to run the test when the multiprocessor implements a CHI interface* on page 4-12.

How to compile a build

Table 4-1 lists the compile command you use, depending on the simulator and whether the multiprocessor implements an ACE or CHI memory interface.

Table 4-1 Command to compile a build

Simulator		ACE memory interface	CHI memory interface
IUS	32-bit	`make -f Makefile.ik SIM=ius OS64=0 BUS=ACE build`	`make -f Makefile.ik SIM=ius OS64=0 BUS=Skyros build`[a]
	64-bit	`make -f Makefile.ik SIM=ius OS64=1 BUS=ACE build`	`make -f Makefile.ik SIM=ius OS64=1 BUS=Skyros build`[a]
QuestaSim	32-bit	`make -f Makefile.ik SIM=mti OS64=0 BUS=ACE build`	`make -f Makefile.ik SIM=mti OS64=0 BUS=Skyros build`[a]
	64-bit	`make -f Makefile.ik SIM=mti OS64=1 BUS=ACE build`	`make -f Makefile.ik SIM=mti OS64=1 BUS=Skyros build`[a]
VCS	32-bit	`make -f Makefile.ik SIM=vcs OS64=0 BUS=ACE build`	`make -f Makefile.ik SIM=vcs OS64=0 BUS=Skyros build`[a]
	64-bit	`make -f Makefile.ik SIM=vcs OS64=1 BUS=ACE build`	`make -f Makefile.ik SIM=vcs OS64=1 BUS=Skyros build`[a]

a. Use this command if the UVM library install is in the default location otherwise see *Build commands to use when the UVM install differs from the default location* on page 4-11.

If your simulation environment includes support for the Protocol Buffers then you can generate a model that supports Tarmac trace by including the `ENABLE_TARMAC=1` option. For example:

- `make -f Makefile.ik SIM=ius OS64=0 BUS=ACE ENABLE_TARMAC=1 build`
- `make -f Makefile.ik SIM=vcs OS64=1 BUS=Skyros ENABLE_TARMAC=1 build`

Build commands to use when the UVM install differs from the default location

For a CHI implementation, the `Makefile` requires the UVM library and expects it to exist at `testbench/shared/globalshared/uvm-`*`version`*.

Note

`version` is the UVM version number that the *ARM® Cortex®-A57 MPCore™ Processor Release Note* recommends to use.

If your UVM install exists in a different folder then to compile a build for each simulator use the command:

IUS
```
make -f Makefile.ik SIM=ius OS64=0 BUS=Skyros build \
UVM_PATH={absolute path to, and including, the uvm-version directory}
```

QuestaSim
```
make -f Makefile.ik SIM=mti OS64=0 BUS=Skyros build \
UVM_PATH={absolute path to, and including, the uvm-version directory}
```

VCS
```
make -f Makefile.ik SIM=vcs OS64=0 BUS=Skyros build \
UVM_PATH={absolute path to, and including, the uvm-version directory}
```

Note

- To compile a 64-bit build then change `OS64=0` to `OS64=1`.
- To display the default values of `UVM_PATH` you can use the `make` or `make help` commands.
- If your simulation environment includes support for the Protocol Buffers and you require Tarmac trace support then append `ENABLE_TARMAC=1` to the `make` command.

For example, if your UVM install exists at `/usr/lib/uvm-`*`version`* then the build command for each build on the IUS simulator is:

32-bit build
```
make -f Makefile.ik SIM=ius OS64=0 BUS=Skyros build \
UVM_PATH=/usr/lib/uvm-version
```

64-bit build
```
make -f Makefile.ik SIM=ius OS64=1 BUS=Skyros build \
UVM_PATH=/usr/lib/uvm-version
```

How to run the test when the multiprocessor implements an ACE interface

If the multiprocessor implements an ACE interface then Table 4-2 lists the command you use to run the test, for each simulator and build.

Table 4-2 Command to run the test on a multiprocessor that implements an ACE interface

Simulator		Command to run the test
IUS	32-bit	`make -f Makefile.ik SIM=ius OS64=0 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=ius OS64=1 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`
QuestaSim	32-bit	`make -f Makefile.ik SIM=mti OS64=0 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=mti OS64=1 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`
VCS	32-bit	`make -f Makefile.ik SIM=vcs OS64=0 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=vcs OS64=1 BUS=ACE TEST=cs_daplite_`*`mode`*[a] `run`

a. Set *`mode`* to either `jtag` or `sw`. Use `TEST=cs_daplite_jtag` if the test was built for JTAG mode or use `TEST=cs_daplite_sw` if the test was built for SerialWire Debug mode. See *Building the test* on page 4-9.

If your simulation environment includes support for the Protocol Buffers then you can run a test that supports Tarmac trace by including the `ENABLE_TARMAC=1` and `TESTARGS="+tarmacALL"` options. For example:

- `make -f Makefile.ik SIM=ius OS64=0 BUS=ACE TEST=cs_daplite_`*`mode`* `\`
  `ENABLE_TARMAC=1 TESTARGS="+tarmacALL" run`
- `make -f Makefile.ik SIM=vcs OS64=1 BUS=ACE TEST=cs_daplite_`*`mode`* `\`
  `ENABLE_TARMAC=1 TESTARGS="+tarmacALL" run`

How to run the test when the multiprocessor implements a CHI interface

If the multiprocessor implements a CHI interface then Table 4-3 lists the command you use to run the test, for each simulator and build.

Table 4-3 Commands to run the test on a multiprocessor that implements a CHI interface

Simulator		Command to run the test
IUS	32-bit	`make -f Makefile.ik SIM=ius OS64=0 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=ius OS64=1 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`
QuestaSim	32-bit	`make -f Makefile.ik SIM=mti OS64=0 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=mti OS64=1 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`
VCS	32-bit	`make -f Makefile.ik SIM=vcs OS64=0 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`
	64-bit	`make -f Makefile.ik SIM=vcs OS64=1 BUS=Skyros TEST=cs_daplite_`*`mode`*[a] `run`

a. Set *`mode`* to either `jtag` or `sw`. Use `TEST=cs_daplite_jtag` if the test was built for JTAG mode or use `TEST=cs_daplite_sw` if the test was built for SerialWire Debug mode. See *Building the test* on page 4-9.

If your simulation environment includes support for the Protocol Buffers then you can run a test that supports Tarmac trace by including the `ENABLE_TARMAC=1` and `TESTARGS="+tarmacALL"` options. For example:

- `make -f Makefile.ik SIM=ius OS64=0 BUS=Skyros TEST=cs_daplite_`*`mode`* `\`
  `ENABLE_TARMAC=1 TESTARGS="+tarmacALL" run`

- `make -f Makefile.ik SIM=vcs OS64=1 BUS=Skyros TEST=cs_daplite_`*`mode`* `\`
  `ENABLE_TARMAC=1 TESTARGS="+tarmacALL" run`

4.6.6 Example test output

Example 4-1 shows an example of the output for a multiprocessor with two processors, when the DAP-Lite IK test is configured with a JTAG debug interface.

Additional output example logs are supplied in `${PROJ_LOGICAL}/testbench/integration_kit_daplite/logs/example_logs`.

Example 4-1 Example output for an Cortex-A57 MPCore multiprocessor with two processors

```
        TUBE: Make powerup requests to Debug and System domains
CPU0: cs_daplite.s
        TUBE: Check presence of APB-AP
CPU0: Unlock processor CTIs
        TUBE: Check DEVICEEN to APB-AP
CPU0: Set up processor CTIs for cross-halt and synchronized restart
        TUBE: Clear OS lock
CPU0: Fire debug request trigger to halt all CPUs
        TUBE: Wait for CPUs to enter Debug state
        TUBE: CPU0 is halted
        TUBE: CPU1 is halted
        TUBE: All CPUs in Debug state
        TUBE: Clear all CTI debug requests
        TUBE: Perform checks from SWJIM
        TUBE: SECTION 1: DAPLITE ROM table check
        TUBE: Check DAPLITE ROM location
        TUBE: Check DAPLITE ROM table Component ID and Peripheral ID
        TUBE: Check DAPLITE ROM table content
        TUBE:  check next location is Cortex-A57 ROM table entry
        TUBE:  check next location is ROM termination entry
        TUBE: SECTION 2: DAPLITE APB-AP check
        TUBE:  Cortex-A57 ROM table
        TUBE:  DBG0
        TUBE:  CTI0
        TUBE:  PMU0
        TUBE:  ETM0
        TUBE:  DBG1
        TUBE:  CTI1
        TUBE:  PMU1
        TUBE:  ETM1
        TUBE: Checks from SWJIM finished
        TUBE: Prepare to restart CPUs
        TUBE: Fire debug restart trigger to restart all CPUs
CPU0: CPU has been restarted
CPU1: CPU has been restarted
CPU0: SECTION 3: DAPLITE System APB check
CPU0: SECTION 4: Processor DBGROMADDR(V) check
CPU0: All checks finished
CPU0: ** TEST PASSED OK **
```

Chapter 5
CoreSight Trace and Debug Integration

This chapter describes CoreSight debug and trace integration. It contains the following section:

- *About CoreSight trace and debug integration* on page 5-2.

5.1 About CoreSight trace and debug integration

You can integrate the Cortex-A57 MPCore processor with CoreSight debug and trace components. The ARM CoreSight SoC-400 r3p0, or later, provides CoreSight debug and trace components for use with the Cortex-A57 processor. Alternatively, you can use your own debug and trace components.

The SoC-400 provides a set of configurable components and example CoreSight systems. You can use these as provided or modify them so that they integrate with your system. See the *ARM® CoreSight™ SoC-400 Technical Reference Manual* for more information.

The SoC-400 is licensed separately. If you do not have this license, you can still use the CoreSight DAP-Lite to build a debug only system, as Chapter 4 *Integration Kit* describes.

Appendix A
Revisions

This appendix describes the technical changes between released issues of this document.

Table A-1 issue A

Change	Location	Affects
First release	-	-

Table A-2 Changes between issue A and Issue B

Change	Location	Affects
Renamed time events to timer interrupts	Throughout the document	All
Updated timing of **MBISTACK** assertion and deassertion	Figure 3-6 on page 3-22	All
Updated the description of **CLREXMONREQ** and **CLREXMONACK**	Table 2-6 on page 2-11	All
Updated the description of **CTIIRQ** and **CTIIRQACK**	Table 2-25 on page 2-24	All
Updated the connection information	Table 2-29 on page 2-27	All
Updated the LS_PSQ to L2_PSQ	Table 3-10 on page 3-16	All