Fixed Virtual Platforms

Version 1.4

VE Cortex-A15 Cortex-A7 CCI-400 User Guide

ARM

ARM DUI0848F

Fixed Virtual Platforms

VE Cortex-A15 Cortex-A7 CCI-400 User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	30 November 2014	Non-Confidential	New document for Fast Models v9.1, from DUI0585C for v9.0.
B	28 February 2015	Non-Confidential	Update for v9.2.
C	31 May 2015	Non-Confidential	Update for v9.3.
D	31 August 2015	Non-Confidential	Update for v9.4.
E	30 November 2015	Non-Confidential	Update for v9.5.
F	29 February 2016	Non-Confidential	Update for v9.6.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

Fixed Virtual Platforms VE Cortex-A15 Cortex-A7 CCI-400 User Guide

Preface

This preface introduces the *Fixed Virtual Platforms VE Cortex-A15 Cortex-A7 CCI-400 User Guide*.

It contains the following:

About this book

This book describes how to configure and use the *Fixed Virtual Platform* (FVP) VE Cortex-A15 Cortex-A7 CCI-400 Cache Coherent Interconnect. The model enables software applications to run on a virtual implementation based on the *Versatile™ Express* (VE) memory map. The model contains a CoreTile that models an ARM Cortex-A15 SMP and an ARM Cortex-A7 SMP connected using the CCI-400.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction

Read this for an introduction to the *Versatile Express* (VE) Cortex-A15 Cortex-A7 CCI-400 *Fixed Virtual Platform* (FVP).

Chapter 2 Getting Started with the Cortex-A15 Cortex-A7 CCI-400 FVP

This chapter describes how to use the model. It also contains information regarding license key requirements and installation.

Chapter 3 Programmers Reference

Read this for a description of the functionality that the model supports, including the memory map and configuration parameters.

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 the *ARM Glossary* for more information.

Typographic 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>`
: Encloses 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, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

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 *Fixed Virtual Platforms VE Cortex-A15 Cortex-A7 CCI-400 User Guide*.
- The number ARM DUI0848F.
- If applicable, the page number(s) to which your comments refer.
- 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.

Other information

- *ARM Information Center*.
- *ARM Technical Support Knowledge Articles*.
- *Support and Maintenance*.
- *ARM Glossary*.

Chapter 1
Introduction

Read this for an introduction to the *Versatile Express* (VE) Cortex-A15 Cortex-A7 CCI-400 *Fixed Virtual Platform* (FVP).

Describes system models, the VE FVPs, and the VE Cortex-A15 Cortex-A7 CCI-400 FVP.

It contains the following sections:

1.1 About system models

The *Fixed Virtual Platforms* (FVPs) enable you to develop software without the requirement for actual hardware.

The software models provide a *Programmer's View* (PV) model of processors and peripheral devices. The functional behavior of a model is based on functionality in equivalent real hardware. Absolute timing accuracy is sacrificed to achieve fast simulated execution speed. To confirm software functionality, use the PV models. However, do not rely on them for:

- The accuracy of cycle counts.
- Low-level component interactions.
- Other hardware-specific behavior.

1.2 About the VE FVP

Versatile Express (VE) is a hardware development platform that ARM produces.

You can split a VE system into the following sub-components:

Motherboard
The Motherboard Express µATX includes a range of peripherals that provide a general purpose I/O platform. The motherboard contains two daughterboard sockets for CoreTile Express or LogicTile Express boards. For more information, see the *Motherboard Express µATX V2M-P1 Technical Reference Manual*.

Daughterboard
Field Programmable Gate Array (FPGA) and processor daughterboards provide:
- Custom peripherals.
- Early access to processor designs.
- Production test chips.

System memory is also implemented on the daughterboard.

Processor
An implementation of an ARM processor.

The VE *Fixed Virtual Platform* (FVP) is a system model implemented in software. The model contains:
- Virtual implementations of a motherboard.
- A single daughterboard.
- A specific ARM processor.
- Associated interconnections.

The model is based on the VE platform memory map, but it is not intended to be an accurate representation of a specific VE hardware revision. The VE FVP supports selected peripherals as this document describes. The supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware. The model is developed using the ARM® Fast Models library product.

1.3 About the Cortex-A15 Cortex-A7 CCI-400 FVP

The *Dual Cluster System* (DCS) contains the Cortex-A15 and Cortex-A7 clusters.

This includes the interconnect required to implement the heterogeneous multi-processing platform, and to support the software-controlled powering up and powering down of the Cortex-A15 and Cortex-A7 clusters, and task migration between the two. The interconnect is implemented using:

CCI-400 Cache Coherent Interconnect
: Provides bus interconnect and coherency functions in a single module. It supports connectivity for up to two ACE masters. For example, a Cortex-A15 processor and a Cortex-A7 processor. For more information, see the *CCI-400 Cache Coherent Interconnect Architecture Specification*.

Dual Cluster System Configuration Block (DCSCB)
: Provides basic functionality for controlling clocks, resets, and configuration pins in the CoreTile. For more information, see *3.8.4 Dual cluster system configuration block parameters* on page 3-34.

A single external Virtual GIC
: This is shared between the Cortex-A15 and Cortex-A7 clusters. For more information, see the *Virtual GIC Specification*.

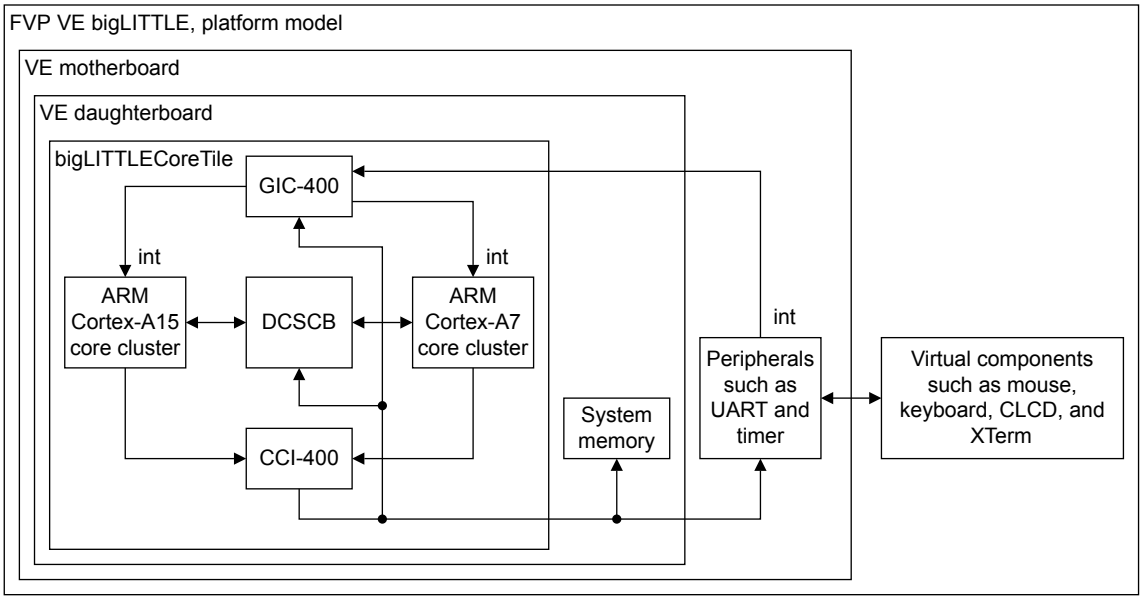

Figure 1-1 Model architecture

Related references

3.8.4 Dual cluster system configuration block parameters on page 3-34.

Chapter 2
Getting Started with the Cortex-A15 Cortex-A7 CCI-400 FVP

This chapter describes how to use the model. It also contains information regarding license key requirements and installation.

It contains the following sections:

2.1 Supported operating systems for the Cortex-A15 Cortex-A7 CCI-400 FVP

The Cortex-A15 Cortex-A7 CCI-400 FVP needs certain software.

Red Hat Enterprise Linux 6.4	on 64-bit architectures.
Ubuntu 12.04 LTS	on 64-bit architectures.
Microsoft Windows 7 with Service Pack 1	on 64-bit architectures, with runtime support for Visual Studio.

2.2 Licenses for Cortex-A15 Cortex-A7 CCI-400

The platform model requires FlexLm license keys to run.

The license feature names are as follows:

- `SG_ARM_Cortex-A15_CT`.
- `SG_ARM_Cortex-A7_CT`.
- `SG_v7SystemIP_CT`.
- `FM_Simulator`.

In addition, the Model Debugger requires a license key with the feature name `MaxView_Debugger`.

2.3 Installing the Cortex-A15 Cortex-A7 CCI-400

This section describes how to install the Cortex-A15 Cortex-A7 CCI-400.

ARM provides the model as an installer package.

Prerequisites

On Microsoft Windows 7, you must install runtime support for Visual Studio applications beforehand. These libraries are available free of charge from Microsoft:

http://www.microsoft.com/en-gb/download/details.aspx?id=40784

Alternatively, visit *http://www.microsoft.com* and search for “Visual 20*XX* runtime”.

Procedure

1. Start the installer program for the required host platform, `setup.exe` or `setup.bin`, and follow the instructions on the screen.

2.4 Running models from the command line

The pre-built models are supplied as *Integrated SIMulator* (ISIM) binaries.

You can execute these binaries:

- Directly from the command line.
- Indirectly using Model Debugger.

Options

To run the model from the command line, type the name of the ISIM binary, for example:

```
FVP_VE_Cortex-A15x1-A7x1
```

```
FVP_VE_Cortex-A15x1-A7x1.exe
```

Note

The `FVP_VE_Cortex-A15x1-A7x1` file corresponds to the ISIM binary for a Cortex-A15×1 Cortex-A7×1 CCI-400 dual cluster platform model.

The `FVP_VE_Cortex-A15x4-A7x4` file corresponds to the ISIM binary for a Cortex-A15×4 Cortex-A7×4 CCI-400 dual cluster platform model.

The following table shows the arguments and options that you can specify on the command line.

Table 2-1 Command line options

Short	Long option	Description
`-a`	`--application` *`target=filename`*	Loads the application file *`filename`* into processor *`target`*. For example: `coretile.cluster0.cpu0=brot_ve.axf`
-	`--data` *`target=file@[space:]address`*	Loads raw data from *`file`* at the specified *`address`* in the specified *`target`*. Optionally, you can also specify a memory *`space`*.
-	`--dump` *`target=file@[space:]address`*	Dumps raw data into *`file`* from the specified *`address`* in the specified *`target`*. Optionally, you can also specify a memory *`space`*.
`-b`	`--break` *`target=address`*	Sets a break-point at *`address`* for *`target`*.
`-s`	`--start` *`target=address`*	Sets initial PC to application start address.
`-C`	`--parameter` *`parameter`*	Sets a single parameter of the model. Parameters are specified as a path that names the instance and the parameter name using dot separators. For example: `foo.bar.inst.parameter=1000` If it is necessary to set multiple parameters at the same time, use the `--config-file` option instead.
-	`--timelimit` *`n`*	Number of seconds to run, excluding startup and shutdown. The default is unlimited.
-	`--cpulimit` *`n`*	Number of host processor seconds to run, kernel and user, excluding startup and shutdown. The default is unlimited.
-	`--cyclelimit` *`n`*	Number of cycles to run. This is ignored if `-S` is specified. The default is unlimited.
-	`--list-instances`	Lists target instances.
`-l`	`--list-params`	Lists target instances and their parameters.

Table 2-1 Command line options (continued)

Short	Long option	Description
-	`--list-memory`	Lists memory information.
`-f`	`--config-file` *filename*	Loads model configuration parameters from file *filename*.
`-o`	`--output` *filename*	Redirects parameter, memory, and instance lists to the output file *filename*.
`-t`	`--cadi-trace`	Enables diagnostic output of CADI calls and call-backs.
`-L`	`--cadi-log`	Log all CADI calls into an XML logfile.
`-S`	`--cadi-server`	Starts the CADI server and enables debuggers to connect to targets in the simulation.
-	`--stat`	Prints run statistics on exit.
-	`--trace-plugin filename`	Loads trace plugin *filename*.
`-h`	`--help`	Shows command line help message and exits.
`-P`	`--prefix`	Prefixes semi-hosting output with target instance name.
`-R`	`--run`	Runs simulation immediately after load even with CADI server.
`-V`	`--verbose`	Specifies more verbosity about the current status.
`-v`	`--version`	Prints version and copyright information.
`-q`	`--quiet`	Suppresses all informational output.
-	`--print-port-number`	Prints the port number on which the CADI server is listening.
`-k`	`--keep-console`	Keeps the console window open after completion. Microsoft Windows only.

2.5 Running models using Model Debugger

Describes how to use the Model Debugger application to debug an application running on the model.

You can start the models directly using Model Debugger from the command line, for example:

```
modeldebugger --debug-isim bin/FVP_VE_Cortex-A15x1-A7x1 \
--application coretile.cluster0.cpu0=examples/brot_ve.axf
```

Alternatively, you can start the model using the CADI server by using the `--cadi-server` command line argument that *2.4 Running models from the command line* on page 2-17 describes. In this case, you can start Model Debugger separately and connect it to the model using CADI by selecting:

File > **Connect to Model** > ***<select model from list>*** > **Connect**.

Related information

Model Debugger for Fast Models User Guide.

2.6 Configuring the model

Describes how to configure *Versatile Express* (VE) *Fixed Virtual Platforms* (FVPs).

Note

3.5 Model parameters on page 3-27 describes the valid user settings for the VE FVP parameters and their effects.

When you start the model from the command line, you can configure it using either:

- The `--parameter` command line argument.
- A configuration file and the `--config-file` command line argument.

You can configure a model started from the command line by including a reference to an optional plain text configuration file.

Each line of the configuration file must contain:

- The name of the component instance.
- The parameter to modify.
- Its value.

You must use the following format:

`instance.`*`parameter=value`*

The `instance` can be a hierarchical path, with each level separated by a dot "." character. You can include comment lines in your configuration file. These lines begin with a `#` character. You can set Boolean values using either `true` or `false`, or `1` or `0`.

You can generate a valid configuration file with all parameters set to default values with the `--list-params` option by directing the output into the new configuration file.

Example 2-1 Sample configuration file including syntax examples

```
# Disable semihosting using true/false syntax
coretile.cluster0.cpu0.semihosting-enable=false
#
# Enable VFP at reset using 1/0 syntax
coretile.cluster0.cpu0.vfp-enable_at_reset=1
#
# Set the baud rate for UART 0
motherboard.pl011_uart0.baud_rate=0x4800
```

Chapter 3
Programmers Reference

Read this for a description of the functionality that the model supports, including the memory map and configuration parameters.

This chapter also describes:

- Registers.
- Debug behaviors.
- Expected performance.
- Known limitations.
- Functionality that is not supported.

It contains the following sections:

3.1 Fixed Virtual Platforms for VE platform functionality

The Fixed Virtual Platforms for the VE platform provide functionality for the following components:

VE motherboard model with:

- VE System Register block.
- Two Dual Timer modules, SP804.
- Watchdog module, SP805.
- System Controller, SP810.
- Four UARTs, PrimeCell PL011.
- Color LCD Controller, PrimeCell PL111 CLCD.
- Real-Time Clock, PrimeCell PL031 RTC.
- Two PS/2 keyboard and mouse interfaces, PrimeCell PL050 KMI.
- Multimedia Card Interface, PrimeCell PL180 MCI.
- Advanced Audio CODEC Interface, PrimeCell PL041 AACI.
- 10/100 Non-PCI Ethernet Controller, SMSC 91C111.
- Two 64MB areas of user NOR Flash.
- 8MB of local video SRAM.

The VE motherboard model also includes the following virtual components:

- PS/2 mouse and keyboard models connected to the PL050 KMIs.
- Visualization for CLCD display with keyboard and mouse support.
- Generic *Multi-Media Card* (MMC) connected to the PL180 MCI.
- Four telnet terminals, one attached to each UART.
- Flash loaders for both banks of flash.
- Audio out connected to the PL041 AACI.
- Ethernet crossover cable and host-bridge connected to the SMSC 91C111.
- *Virtual File System 2* (VFS2) for host file system access.

VE daughterboard model with the following:

- *VE Daughterboard Configuration and Control block* (VEDCC).
- VE Interrupt mapper.
- 64Kb System RAM.
- 4GB DRAM.

ARM Cortex-A15 Cortex-A7 CoreTile models with the following:

- ARM Cortex-A15 core cluster.
- ARM Cortex-A7 core cluster.
- Shared Virtual Generic Interrupt Controller, v7 GIC-400.
- Dual Cluster System Configuration Block.
- CCI-400 Cache Coherent Interconnect.

3.2 Fixed Virtual Platform VE Cortex-A15 Cortex-A7 CCI-400 memory map and interrupts

The following table shows the global memory map for the platform model. This map is based on the Versatile Express RS1 memory map with the RS2 extensions.

Table 3-1 Global memory map for the platform model

Address range	Size	Modeled	Description
0x00_00000000-0x00_03FFFFFF	64MB	Yes	NOR FLASH0, CS0.
0x00_04000000-0x00_08FFFFFF	64MB	Yes	Secure RAM.
0x00_08000000-0x00_0BFFFFFF	64MB	Yes	NOR FLASH0 alias, CS0.
0x00_0C000000-0x00_0FFFFFFF	64MB	Yes	NOR FLASH1, CS4.
0x00_10000000-0x00_0001FFFF	-	-	Unused, CS5.
0x00_10021000-0x00_13FFFFFF	-	-	Unused, CS5.
0x00_14000000-0x00_17FFFFFF	-	No	PSRAM, CS1.
0x00_18000000-0x00_1BFFFFFF	64MB	Yes	Peripherals, CS2. See *3.3 CS2 peripheral memory map* on page 3-25.
0x00_1C000000-0x00_1FFFFFFF	64MB	Yes	Peripherals, CS3. See *3.4 CS3 peripheral memory map* on page 3-26.
0x00_20000000-0x00_2BFFFFFF	-	No	CoreSight and peripherals.
0x00_2C000000-0x00_2C000FFF	-	-	Unused, CPU PERIPHBASE.
0x00_2C001000-0x00_2C007FFF	128MB	Yes	Shared v7 GIC-400.
0x00_2C008000-0x00_2C08FFFF	-	-	Unused.
0x00_2C090000-0x00_2C09FFFF	64MB	Yes	CCI-400 PMU.
0x00_2C0A0000-0x00_2CFFFFFF	-	-	Unused.
0x00_2D000000-0x00_2DFFFFFF	-	No	Graphics space.
0x00_2E000000-0x00_2E00FFFF	64MB	Yes	System SRAM.
0x00_2E010000-0x07_FFFFFFFF	-	No	Ext AXI.
0x00_60000000-0x00_60000FFF	4KB	Yes	Dual Cluster System Configuration Block.
0x00_80000000-0x00_FFFFFFFF	2GB	Yes	4GB DRAM, in 32-bit address space. The model contains 4GB of DRAM. The DRAM memory address space is aliased across the three different regions, and where the mapped address space is greater than 4GB.
0x01_00000000-0x07_FFFFFFFF	-	-	Unused.
0x08_00000000-0x08_FFFFFFFF	4GB	Yes	4GB DRAM, in 36-bit address space. The model contains 4GB of DRAM. The DRAM memory address space is aliased across the three different regions, and where the mapped address space is greater than 4GB.
0x09_00000000-0x7F_FFFFFFFF	-	-	Unused.
0x80_00000000-0xFF_FFFFFFFF	512GB	Yes	4GB DRAM, in 40-bit address space. The model contains 4GB of DRAM. The DRAM memory address space is aliased across the three different regions, and where the mapped address space is greater than 4GB.

3.3 CS2 peripheral memory map

The following table shows the memory map for peripherals in the CS2 region.

Table 3-2 CS2 peripheral memory map

Address range	Size	Int	Modeled	Description
0x00_18000000-0x00_19FFFFFF	32MB	-	Yes	8MB VRAM
0x00_1A000000-0x00_1AFFFFFF	16MB	47	Yes	Ethernet, SMSC 91C111
0x00_1B000000-0x00_1BFFFFFF	16MB	-	No	USB

3.4 CS3 peripheral memory map

The following table shows the memory map for peripherals in the CS3 region.

Table 3-3 CS3 peripheral memory map

Address range	Size	Int	Modeled	Description
0x00_1C000000-0x 0x00_1C00FFFF	64KB	-	No	Local DAP ROM
0x00_1C010000-0x 0x00_1C01FFFF	64KB	-	Yes	VE System Registers
0x00_1C020000-0x 0x00_1C02FFFF	64KB	-	Yes	System Controller, SP810
0x00_1C030000-0x00_1C03FFFF	64KB	-	No	TwoWire serial interface, PCIe
0x00_1C040000-0x00_1C04FFFF	64KB	43	Partial	AACI, PL041
0x00_1C050000-0x00_1C05FFFF	64KB	41, 42	Partial	MCI, PL180
0x00_1C060000-0x00_1C06FFFF	64KB	44	Yes	KMI, keyboard, PL050
0x00_1C070000-0x00_1C07FFFF	64KB	45	Yes	KMI, mouse, PL050
0x00_1C080000-0x00_1C08FFFF	64KB	-	-	Reserved
0x00_1C090000-0x00_1C09FFFF	64KB	37	Yes	UART0, PL011
0x00_1C0A0000-0x00_1C0AFFFF	64KB	38	Yes	UART1, PL011
0x00_1C0B0000-0x00_1C0BFFFF	64KB	39	Yes	UART2, PL011
0x00_1C0C0000-0x00_1C0CFFFF	64KB	40	Yes	UART3, PL011
0x00_1C0D0000-0x00_1C0EFFFF	64KB	73	Yes	VFS2
0x00_1C0D0000-0x00_1C0EFFFF	64KB	-	-	Reserved
0x00_1C0F0000-0x00_1C0FFFFF	64KB	32	Yes	Watchdog, SP805
0x00_1C100000-0x00_1C10FFFF	64KB	-	-	Reserved
0x00_1C110000-0x00_1C11FFFF	64KB	34	Yes	Timer-0, SP804
0x00_1C120000-0x00_1C12FFFF	64KB	35	Yes	Timer-1, SP804
0x00_1C130000-0x00_1C15FFFF	192KB	-	-	Reserved
0x00_1C160000-0x00_1C16FFFF	64KB	-	No	TwoWire serial interface, DVI
0x00_1C170000-0x00_1C17FFFF	64KB	36	Yes	Real-time Clock, PL031
0x00_1C180000-0x00_1C19FFFF	128KB	-	-	Reserved
0x00_1C1A0000-0x00_1C1AFFFF	64KB	-	No	CF card
0x00_1C1B0000-0x00_1C1EFFFF	256KB	-	-	Reserved
0x00_1C1F0000-0x00_1C1FFFFF	64KB	46	Yes	Color LCD Controller, PL111
0x00_1C200000-0x00_1FFFFFFF	62MB	-	-	Reserved

3.5 Model parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.`*`parameter`*`=`*`value`*

Table 3-4 FVP_VE_Cortex-A15x1-A7x1 parameters

Parameter	Type	Allowed values	Default value	Description
`proc_id0`	`int`	-	`0xC000000`	Processor ID at CoreTile Express Site 1.
`proc_id1`	`int`	-	`0xFF000000`	Processor ID at CoreTile Express Site 2.

Table 3-5 FVP_VE_Cortex-A15x4-A7x4 parameters

Parameter	Type	Allowed values	Default value	Description
`proc_id0`	`int`	-	`0x14000000`	Processor ID at CoreTile Express Site 1.
`proc_id1`	`int`	-	`0xFF000000`	Processor ID at CoreTile Express Site 2.

3.6 Motherboard peripheral parameters

This section describes the motherboard peripheral parameters.

This section contains the following subsections:

3.6.1 Color LCD controller parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.pl111_clcd.`*`parameter`*`=`*`value`*

Table 3-6 Color LCD controller parameters

Parameter	Type	Allowed values	Default value	Description
`pixel_double_limit`	`int`	`-`	`12C`	The threshold, in horizontal pixels, below which pixels sent to the frame-buffer are doubled in size in both dimensions.

3.6.2 Ethernet parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.smsc_91c111.`*`parameter`*`=`*`value`*

Table 3-7 Ethernet parameters

Parameter	Type	Allowed values	Default value	Description
`enabled`	`bool`	`true`, `false`	`false`	Host interface connection enabled.
`mac_address`	`string`	See *3.6.3 MAC address parameter* on page 3-28	`00:02:f7:ef:60:30`	Host and model MAC address.
`promiscuous`	`bool`	`true`, `false`	`true`	Places the host into promiscuous mode, for example, when sharing the Ethernet controller with the host OS.

3.6.3 MAC address parameter

There are options available for the `mac_address` parameter.

1. If a MAC address is not specified, when the simulator is run, it takes the default MAC address and changes its bottom two bytes from 00:02 to the bottom two bytes of the MAC address of one of the adaptors on the host PC. This provides some degree of MAC address uniqueness when running models on multiple hosts on a local network.
2. If you specify the MAC address as auto, this generates a completely random local MAC address each time the simulator is run. The address has bit 1 set and bit 0 clear in the first byte to indicate a locally-administered unicast MAC address.

Note

DHCP servers allocate IP addresses, but because they sometimes do this based on the MAC address provided to them, then using random MAC addresses might interact unfavorably with some DHCP servers.

3.6.4 System controller parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.sp810_sysctrl.`*`parameter`*`=`*`value`*

Table 3-8 System controller configuration parameters

Parameter	Type	Allowed values	Default value	Description
sysid	int	-	0x00000000	Value for the system identification register.
use_s8	bool	true, false	false	Select whether switch S8 is enabled.

3.6.5 VE System Register block parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.ve_sysregs.`*`parameter`*`=`*`value`*

Table 3-9 VE system register block parameters

Parameter	Type	Allowed values	Default value	Description
user_switches_value	int	-	0x00	User switch.
tilePresent	bool	true, false	true	CoreTile fitted status.

3.6.6 UART parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.pl011_uart`*`x`*`.`*`parameter`*`=`*`value`*

x is the UART identifier 0, 1, 2, or 3.

Table 3-10 UART parameters

Parameter	Type	Allowed values	Default value	Description
baud_rate	int	-	0x9600	Baud rate.
clock_rate	int	-	0xE10000	Clock rate for PL011.
in_file	string	-	""	Input file.
out_file	string	-	""	Output file, use "-" to send all output to stdout.
in_file_escape_sequence	string	-	##	Input file escape sequence string.
shutdown_on_eot	bool	true, false	false	Shutdown simulation when an EOT, ASCII 4, character is transmitted.

Table 3-10 UART parameters (continued)

Parameter	Type	Allowed values	Default value	Description
unbuffered_output	bool	true, false	false	Unbuffered output.
untimed_fifos	bool	true, false	false	Ignore the clock rate and transmit or receive serial data immediately.
uart_enable	bool	true, false	false	Enable the UART when the system starts.

3.6.7 Watchdog parameter

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.sp805_wdog.`*`parameter`*`=`*`value`*

Table 3-11 Watchdog parameter

Parameter	Type	Allowed values	Default value	Description
simhalt	bool	true, false	false	Halt on reset.

3.7 Motherboard virtual component parameters

This section describes the motherboard virtual component parameters.

This section contains the following subsections:

3.7.1 FLASH loader parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.flashloader`*`x`*`.`*`parameter`*`=`*`value`*

x is the FLASH identifier 0 or 1.

Table 3-12 FLASH loader parameters

Parameter	Type	Allowed values	Default value	Description
`fname`	`string`	Valid file name	-	Path to the host file that initializes FLASH contents when the model starts. You can gzip compress the file.
`fnameWrite`	`string`	Valid file name	-	Path to the host file used to save FLASH contents when the model exits.

3.7.2 Host bridge parameter

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.hostbridge.`*`parameter`*`=`*`value`*

Table 3-13 Host bridge parameter

Parameter	Type	Allowed values	Default value	Description
`interfaceName`	`string`	Valid string	`ARM0`	Host Interface identifier.

3.7.3 Multimedia card parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.mmc.`*`parameter`*`=`*`value`*

Table 3-14 Multimedia card parameters

Parameter	Type	Allowed values	Default value	Description
`p_mmc_file`	`string`	Valid filename	`mmc.dat`	File used for the MMC component backing store.
`p_prodName`	`string`	Six character string	`ARMmmc`	Card ID product name.
`p_prodRev`	`int`	-	`0x1`	Card ID product revision.
`p_manid`	`int`	-	`0x2`	Card ID manufacturer ID.

Table 3-14 Multimedia card parameters (continued)

Parameter	Type	Allowed values	Default value	Description
p_OEMid	int	-	0x0000	Card ID OEM ID.
p_sernum	int	-	0xCA4D0001	Card serial number.

3.7.4 Terminal parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.terminal_`*`x`*`.`*`parameter`*`=`*`value`*

x is the terminal identifier 0, 1, 2, or 3.

Table 3-15 Terminal parameters

Parameter	Type	Allowed values	Default value	Description
mode	string	telnet, raw	telnet	Terminal initialization mode.
start_telnet	bool	true, false	true	Enables the terminal when the system starts.
start_port	int	Valid port number	5000	Port used for the terminal when the system starts. If the specified port is not free, the port value is incremented by 1 until a free port is found.

3.7.5 VFS2 parameter

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.vfs2.`*`parameter`*`=`*`value`*

Table 3-16 VFS2 parameter

Parameter	Type	Allowed values	Default value	Description
mount	string	Valid path	-	Path to host folder to make accessible inside the model.

3.7.6 Visualization parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`motherboard.vis.`*`parameter`*`=`*`value`*

Table 3-17 Visualization parameters

Parameter	Type	Allowed values	Default value	Description
trap_key	int	0x00-0xFF	0x4A, left Alt key	Trap key that works with the left Ctrl to toggle the mouse display.
rate_limit-enable	bool	true, false	true	Rate limit simulation.
disable_visualization	bool	true, false	false	Disables visualization.

3.8 CoreTile parameters

This section describes the CoreTile parameters.

This section contains the following subsections:

3.8.1 Cluster parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`coretile.cluster`*x*`.`*parameter*`=`*value*

x 0 for the Cortex-A15 cluster, 1 for the Cortex-A7 cluster.

Table 3-18 Cluster parameters

Parameter	Type	Allowed values	Default value	Description
`CFGDISABLE`	`bool`	`true`, `false`	`false`	Disables some access to DIC registers.
`CLUSTER_ID`	`int`	0-15	0	Cluster ID value.
`device-accurate-tlb`	`bool`	`true`, `false`	`false`	Sets whether device-accurate number of TLBs are modeled.
`dic-spi_count`	`int`	0-224, in increments of 32	64	Number of shared peripheral interrupts implemented.
`IMINLN`	`bool`	`true`, `false`	`true`	Instruction cache minimum line size for cluster 0, not cluster 1. `false` 32 bytes. `true` 64 bytes.
`l1_dcache-state_modelled`	`bool`	`true`, `false`	`false`	Includes Level 1 data cache state model.
`l1_icache-state_modelled`	`bool`	`true`, `false`	`false`	Includes Level 1 instruction cache state model.
`l2_cache-size`	`int`	512KB, 1MB, 2MB, 4MB	`0x80000`	Sets Level 2 cache size in bytes integer.
`l2_cache-state_modelled`	`bool`	`true`, `false`	`false`	Includes Level 2 cache state model.

3.8.2 Core parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`coretile.cluster`*x*`.cpu`*y*`.`*parameter*`=`*value*

x 0 for the Cortex-A15 cluster, 1 for the Cortex-A7 cluster.

y the core index, in the range 0-3.

Table 3-19 Core parameters

Parameter	Type	Allowed values	Default value	Description
CFGEND	bool	true, false	false	Initializes to BE8 endianness.
CFGNMFI	bool	true, false	false	Enables non-maskable fast interrupts on startup.
CP15SDISABLE	bool	true, false	false	Initializes to disable access to some CP15 registers.
TEINIT	bool	true, false	false	Thumb exception enable. The default has exceptions including reset handled in ARM state.
VINITHI	bool	true, false	false	Initializes with high vectors enabled.
SMPnAMP	bool	true, false	false	Sets whether the processor is part of a coherent domain.
POWERCTLI	int	0x0-0x3	0x0	Default power control state for core.
semihosting-enable	bool	true, false	true	Enables semi-hosting SVC traps.
semihosting-ARM_SVC	int	0x000000-0xFFFFFF	0x123456	ARM SVC number for semi-hosting.
semihosting-Thumb-SVC	int	0x00-0xFF	0xAB	Thumb SVC number for semi-hosting integer.
semihosting-heap_base	int	0x00000000- 0xFFFFFFFF	0x0	Virtual address of heap base.
semihosting-heap_limit	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of heap limit integer.
semihosting-stack_base	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of stack base.
semihosting-stack_limit	int	0x00000000-0xFFFFFFFF	0x0	Virtual address of stack limit.
vfp-enable_at_reset	bool	true, false	false	Enables coprocessor access and VFP at reset.
vfp-preset	bool	true, false	true	Sets whether the model has VFP support.
ase-present	bool	true, false	true	Sets whether the model has NEON support.

3.8.3 GIC-400 parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`coretile.v7_vgic.`*`parameter=value`*

Table 3-20 GIC-400 parameters

Parameter	Type	Allowed values	Default value	Description
enabled	bool	true, false	true	Enables the component. If disabled, then no register writes have any effect.
enable_log_warnings	bool	true, false	false	Enables warning messages.
enable_log_errors	bool	true, false	false	Enables error messages.
enable_log_fatal	bool	true, false	false	Enables fatal messages.

3.8.4 Dual cluster system configuration block parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`coretile.dualclustersytemconfigurationblock.`*`parameter`*`=`*`value`*

Table 3-21 Dual cluster system configuration block parameters

Parameter	Type	Allowed values	Default value	Description
CFG_ACTIVECLUSTER	int	0x1-0x3, bit mask	0x1	Select the cluster that comes out of reset at power on: **Bit[0]** For the Cortex-A15. **bit[1]** For the Cortex-A7.
Cluster0IdOnPOReset	int	0x0-0xF	0x0	Cortex-A15 Cluster ID on power-on reset.
Cluster1IdOnPOReset	int	0x0-0xF	0x1	Cortex-A7 Cluster ID on power-on reset.
DCS_LED	bool	0x00-0xFF	0x00	Initial value of the DCS_LED register that represents the state of the daughterboard LEDs.
ResetValueOfDaughterUserSwitches	int	0x00-0xFF	0x00	The state of the daughterboard user switches at reset.
INTGEN_INTS	int	0-3	3	Number of custom IRQs that the interrupt generator controls is (INTGEN_INTS × 32) + 32.
DCS_ID	int	0x0-0xFFFFFFFF	0x41120000	The value returned by the DCS_ID register.
FlipVGICWiringForCluster0AndCluster1	bool	true, false	false	`true` If this is `true`, then core 0 of cluster 1 becomes core interface 0 on the GIC-400. `false` If this is `false`, then core 0 of cluster 0 becomes core interface 0 on the GIC-400.

3.8.5 CCI-400 parameters

You set these instantiation-time parameters when starting the model.

The syntax to use in a configuration file or on the command line is:

`coretile.cci400.`*`parameter`*`=`*`value`*

Table 3-22 CCI-400 configuration parameters

Parameter	Type	Allowed values	Default value	Description
broadcastcachemain	int	0x0-0x7, bit mask	0x0	For each downstream port, a bit determines whether broadcast cache maintenance operations are forwarded down that port.
barrierterminate	int	0x0-0x7, bit mask	0x7	For each downstream port, determines whether barriers are terminated at that port.

Table 3-22 CCI-400 configuration parameters (continued)

Parameter	Type	Allowed values	Default value	Description
`bufferableoverride`	`int`	`0x0-0x7`, bit mask		For each downstream port, determines whether all transactions are forced to non-bufferable.
`force_on_from_start`	`bool`	`true`, `false`	`false`	The CCI-400 normally starts up with snooping disabled. This parameter enables snooping when the model starts without it being necessary to program it. This parameter applies to simulation reset, not at signal reset.
`log_enabled`	`int`	`0` Logging off. `1` Log only access violations. `2` Also log writes. `3` Also log reads.	`1`	Enables log messages from the CCI-400 register file.

3.9 Memory map differences between the VE hardware and the system model

The model is based on the memory map of the hardware VE platform. However, it is not intended to be an accurate representation of a specific VE hardware revision.

The memory map in the supplied model is sufficiently complete and accurate to boot the same operating system images as for the VE hardware. In the memory map, memory regions that are not explicitly occupied by a peripheral, or by memory, are unmapped. This includes:

- Regions otherwise occupied by a peripheral that is not implemented.
- Areas that are specified as being reserved.

If a host processor accesses these regions, the model issues a warning.

3.10 Memory aliasing differences between the VE hardware and the system model

The model implements address space aliasing of the DRAM. This means that the same physical memory locations are visible at different addresses.

The lower 2GB of the DRAM is accessible at `0x00_80000000`.

The full 4GB of DRAM is accessible at `0x08_00000000`, and again at `0x80_00000000`.

The aliasing of DRAM then repeats from `0x81_00000000` up to `0xFF_FFFFFFFF`.

3.11 Features not present in the model

Some features in the hardware version of the Versatile Express motherboard are not implemented in the system models.

The following features are not implemented in the system models:

- Two-wire serial bus interfaces.
- USB interfaces.
- PCI Express interfaces.
- Compact Flash.
- DVI interfaces.
- Debug and test interfaces.
- *Dynamic Memory Controller* (DMC).
- *Static Memory Controller* (SMC).
- CoreSight.

3.12 Features partially implemented in the model

Partial implementation means that some of the components are present, but their functionality has not been fully modeled.

If you use these features, they might not work as you expect. Check the model release notes for the latest information.

Sound

The VE FVPs implement the PL041 AACI PrimeCell and the audio CODEC in the same way as in the VE hardware, but with a limited number of sample rates.

3.13 Restrictions on the processor models

Separate documentation contains detailed information regarding the features that are not fully implemented in the processor models that are included with the VE *Fixed Virtual Platforms* (FVPs). See the Fast Models Reference Manual.

The following general restrictions apply to the FVP implementations of ARM processors:

- The simulator does not model cycle timing. In aggregate, all instructions execute in one core master clock cycle, with the exception of Wait For Interrupt.
- Write buffers are not modeled.
- Most aspects of *Translation Lookaside Buffer* (TLB) behavior are implemented in the models. Architecture v7 models use the TLB memory attribute settings when you enable stateful cache.
- No MicroTLB is implemented.
- A single memory access port is implemented. The port combines accesses for:
  - Instruction.
  - Data.
  - DMA.
  - Peripherals.

  Configuration of the peripheral port memory map register is ignored.
- All memory accesses are atomic and are performed in programmers view order. All transactions on the PVBus are a maximum of 32 bits wide. Unaligned accesses are always performed as byte transfers.
- Some instruction sequences are executed atomically, ahead of the component master clock, so that system time advances during their execution. This can sometimes have an effect in sequential access of device registers where devices expect time to progress between each access.
- Interrupts are not taken at every instruction boundary.
- The `semihosting-debug` configuration parameter is ignored.
- Integration and test registers are not implemented.
- Not all CP14 debug registers are implemented.
- You must use an external debugger to debug an FVP.
- The model supports the following breakpoint types:
  - Single address unconditional instruction breakpoints.
  - Single address unconditional data breakpoints.
  - Unconditional instruction address range breakpoints.
- Pseudo-registers in the debugger support processor exception breakpoints. Setting an exception register to a non-zero value stops execution on entry to the associated exception vector.
- Performance counters are not implemented.

3.14 Timing considerations

The Fixed Virtual Platforms provide environments that enable you to run software applications in a functionally-accurate simulation.

However, because of the relative balance of fast simulation speed compared to timing accuracy, situations exist where the models might behave unexpectedly. When code interacts with real world devices such as timers and keyboards, data arrives in the modeled device in real world, or wall clock, time, but simulation time can be running much faster than the wall clock.

This means that a single key-press might be interpreted as several repeated key presses, or a single mouse click is incorrectly interpreted as a double click. The VE FVPs provide the Rate Limit feature to match simulation time to wall-clock time. Enabling Rate Limit, either by using the **Rate Limit** button in the CLCD display, or the `rate_limit-enable` model instantiation parameter, forces the model to run at wall clock time. This avoids issues with two clocks running at significantly different rates. For interactive applications, ARM recommends enabling Rate Limit.

3.15 Bus consistency messages

Checkers monitor for suspicious interactions between conflicting memory attributes and cache and *Translation Lookaside Buffer* (TLB) maintenance operations.

These checkers print messages on stderr. You must launch the model from the command line to observe these messages.

The `FASTSIM_PVCACHE_VERBOSE` environment variable controls the checkers and you can set its value as follows:

0 Does not display cconsistency messages. This is the default behavior if `FASTSIM_PVCACHE_VERBOSE` is not set.

1 Only displays severe errors.

2 or higher Displays both severe errors and warnings.

The checkers within the model monitor for the following:

- Multiple entries in the cache with the same security world.

  A Secure and a Non-secure line in the cache and at least one of them is dirty.
- Mismatched attributes.

  Mixed attributes for a particular physical address, that is UNPREDICTABLE in the architecture.
- Cache Coherent Interconnect snoop or DVM enables.

  Misuses of the Cache Coherent Interconnect during snoop or DVM enables or disables.
- Snoop or DVM messages received while in reset.

  A snoop or DVM request is sent to a core in reset. This could deadlock the hardware.
- Invalidation of Dirty lines.

  A cache is invalidated with dirty lines.

3.16 Multiple entries in the cache with the same security world

The models contain AMBA® 4 consistency checkers.

Cause

A secure and a non-secure line both exist, and are dirty, in the same cache at the same time. The system issues warnings at 1MiB boundaries and only issues them once for each 1MiB section of memory. This means that it is possible to observe this error from multiple caches.

Reason

This is not necessarily an error if the backing memory for the cache line in the main memory system is different. However, if they are the same, then you must perform cache maintenance operations with care to ensure that they observe the updated value.

Example 3-1 Example of multiple entries

```
<name>-entry3 is for UD-ns-0000e693be7b2400-ish-iHittable-oHittable, and
<name>-entry4 is for ud- s-0000e693be7b2400-ish-iHittable-oHittable,
that is, same address but different security worlds and one or more are dirty.
This isn't necessarily an error if different memory is backing the lines,
but it is a common error.
This is only reported once for each 1024KiB region for each
cache, so you might observe this error from multiple caches.
```

3.17 Mismatched attributes

The cache models perform the following checks for mismatched attributes:

Cause

1. A 4KiB page is accessed for the first time and the system checks whether any lines from that region exist in the cache. If they mismatch, the system issues a warning.
2. A cacheable access hits in the cache, but the cache line was fetched with different attributes.
3. A snoop request is received that hits a cache line that has it marked as non-shared.

Reason

It is UNPREDICTABLE in the architecture if shareability or cacheability are mixed. You must follow an appropriate sequence of cache maintenance and *Translation Lookaside Buffer* (TLB) operations to change the attributes.

Example 1

```
FVP_VE_Cortex_A15x4_Cortex_A7x4.coretile.cluster0.cluster0.l2_cache: Mismatch in attributes
for page including address ns-000000008ff03000
    29 cache lines are allocated with attributes:-
        inner-WB-cacheable outer-WB-cacheable shareability: osh
    but transaction attributes are:-
        inner-WB-cacheable outer-WB-cacheable shareability: ish
         (PNI-u0x0-m0x2-ish-rawaC-rawaC) inner-WB-cacheable Inner cacheable write back (WB)
          outer-WB-cacheable Outer cacheable write back (WB)
```

`PNI-u0x0-m0x2-ish-rawaC-rawaC` is interpreted as follows:

`P/p`	Privileged/Non-privileged.
`N/S`	Non-secure/Secure.
`I/D`	Instruction-side access/Data-side access.
`u0x0/m0x2`	Corresponds to the internal concepts of user flags and AXI IDs of the model.
`ish/osh/nsh/sys`	Inner shareable/Outer shareable/Non-shareable/System.
`ra`	Read-allocate.
`wa`	Write-allocate.
`C`	Cacheable.
`NC`	Non-Cacheable to normal memory.
`SO`	Strongly-Ordered memory.
`DV`	DeVice memory.

Example 2

```
cluster0.l1icache_2-entry177: Mismatch in transaction from upstream port0 and an existing
entry in the cache:
    Mismatch between cache entry ns-000000008ff05600-osh-iHittable-oHittable
                  and transaction ns-000000008ff05600-ish-iHittable-oHittable
```

`ns/s`	Non-secure/Secure.
`000000008ff05600`	The physical address.
`ish/osh/nsh/sys`	Inner shareable/Outer shareable/Non-shareable/System.
`iHittable`	The inner cacheability marks it as hittable in this cache.
`oHittable`	The outer cacheability marks it as hittable in this cache.

Example 3

```
<name>-entry177: Received a snoop request for [<address-range>]
and we hit an existing entry in the cache but it was incompatible with
being the response to a snoop request:
    <mis-match message>
```

```
We are suppressing the hit in the cache, but a specific implementation
may or may not do the same thing.
```

3.18 Cache Coherent Interconnect snoop and DVM enables

You can program the CCI-400 to control where snoop and DVM messages are transmitted.

Cause

The changes to the snoop control and DVM routing require a finite amount of time to take effect. To determine when they have taken effect, you must poll a register in the CCI-400.

Reason

Any transactions created in the interim period might or might not observe the change and therefore might lead to race conditions and UNPREDICTABLE behavior. The checker sets a flag when the snoop or DVM filters are changed. This flag is only de-asserted after a successful read of a 0, representing snoops in effect, from the CCI-400 Status Register. While the flag is set, any shared transactions or DVM messages produce warning messages.

Examples

- ```
  FVP_VE_Cortex_A15x4_Cortex_A7x4.coretile.cci400.cciinterconnect: received a transaction
  in page ns-000000008ff0d000 It is shared, however, a pending snoop request is in progress
  and so there is a race condition as to whether this request will obey the snoop request
  or not. The transaction attributes are:-
  PND-u0x0-m0x2-ish-rawaC-rawaC upstream port3 ns-000000008ff0d000
  You should poll the CCI-400 Status Register until it says that the snoop changes have
  taken effect. During this time you should ensure that no shared data is prefetchable
  from any of the cores attached to this cluster and that it shouldn't contain shared data
  in any of the caches as they might be evicted.
  ```
- ```
  FVP_VE_Cortex_A15x4_Cortex_A7x4.coretile.cci400.cciinterconnect: received a DVM message:
   <message> However, a snoop change is pending and so there is a race condition as to
  whether this message would obey the new snoop value or not.
  ```

3.19 Snoop or DVM messages received while in reset

Sometimes, snoop or DVM messages are received while a cluster is in reset.

Cause

The CCI-400 has been instructed to enable snoop or DVM messages to a cluster that is held in reset.

Reason

Sending these messages to the cluster in reset might result in deadlock.

Examples

- ```
  <name>: Received a Distributed Virtual Memory (DVM) message whilst cache is held in reset.
  This could deadlock the hardware. Message is:-
      <DVM-message>
  ```
- ```
   *** ERROR: Cache <name> received a snoop request whilst it was in reset!
  ```
- ```
  <name>: upstream port <N> (numbered from 0) changed its reset signal from <old-level> to
  <new-level>
  However, a pending change in snoop request is pending. This might deadlock the hardware.
  ```
  ```

3.20 Invalidation of dirty lines

Sometimes, one or more dirty lines become invalidated from the cache.

Cause

This can only occur if an invalidate instruction is issued, or a core is reset with dirty lines in the cache.

Reason

A particular operation caused one or more dirty lines to become invalidated from the cache. This is likely to lead to data corruption. Although it is legal for this to occur, because it is sufficiently rare to want to do such a thing, the model issues a warning when this occurs.

Example

```
<name>: Reset received whilst dirty lines exist in the cache.
  (Both secure and non-secure worlds)
    UD-ns-0000e6803cf64ae0-ish-iHittable-oHittable
    uD-ns-0000e6823ce64bd0-ish-iHittable-oHittable-prefetched
    uD-ns-0000e6803cf64ae8-ish-iHittable-oHittable
    UD- s-0000e6803ce64ac8-nsh-iHittable-oHittable-prefetched
Total : 4
```

`u/U`	Not-unique/Unique, in terms of ACE terminology.
`d/D`	Not-dirty/Dirty, in terms of ACE terminology.
`prefetched/rmon/wmon`	Internal flags related to the implementation in the model of exclusive monitors.

3.21 Dual Cluster System Configuration Block

The *Dual Cluster System Configuration Block* (DCSCB) provides basic functionality for controlling clocks, resets, and configuration pins in the dual cluster system.

It is not intended to provide the complete trick-box functionality that is found in a typical top-level simulation test-bench. Instead, its main use is to form the programming interface for the clock, power, and reset controllers so that software can implement the ARM big.LITTLE™ switching. The following table shows the DCSCB registers, that this section describes. The DCSCB occupies a 4kB region of memory in the range `0x10020000-0x10020FFF`, and any accesses to undefined areas of this address space result in an error response. The registers are divided into the following categories:

1. The system control registers are identical across all platform implementations of the dual cluster system. They control system-level functions such as the resets of individual cores and clusters.
2. The platform control registers provide a common model for controlling and reading platform-level configuration options for the dual cluster system implementation. Some platform implementations might support only a subset of these registers. The definitions and address offsets of these registers are based on the Versatile Express daughter-card configuration controller command codes.

Note

- All registers are word-sized and only support word-sized transactions.
- Reads from write-only registers or fields return zero.
- Some registers do not implement all 32 bits. See the Width column of the following table.
- Unimplemented bits are RAZ/WI.

The following table shows the DCSCB System Control Registers.

Table 3-23 DCSCB System Control Registers

Name	Offset	Type	Width	Reset	Description
RST_HOLD0	`0x000`	RW	9	`0x00000000` or `0x000001001`	Holds the selected resets in the Cortex-A15 cluster.
RST_HOLD1	`0x004`	RW	9	`0x00000000` or `0x000001001`	Holds the selected resets in the Cortex-A7 cluster.
SYS_SWRESET	`0x008`	WO	24	`0x00000000` or `0x000001001`	Asserts a software reset of the system.
RST_STAT0	`0x00C`	RO	9	`0x00000000` or `0x000001001`	Determines the Cortex-A15 processor resets that are asserted.
RST_STAT1	`0x010`	RO	9	`0x00000000` or `0x000001001`	Determines the Cortex-A7 processor resets that are asserted.
CLUSTER0_CFG_R	`0x020`	RO	20	-	Current configuration of the static configuration input pins of the Cortex-A15 processor.
CLUSTER0_CFG_W	`0x024`	RW	20	`0x00000000`	Configuration of the static configuration input pins of the Cortex-A15 processor at the next reset.
CLUSTER1_CFG_R	`0x028`	RO	20	-	Current configuration of the static configuration input pins of the Cortex-A7 processor.
CLUSTER1_CFG_W	`0x02C`	RW	20	`0x00010000`	Configuration of the static configuration input pins of the Cortex-A7 processor at the next reset.
DCS_CFG_R	`0x030`	RO	2	-	Reads the power-on configuration of system-level configuration pins.

The following table shows the DCSCB Platform Control Registers.

Table 3-24 DCSCB Platform Control Registers

Name	Offset	Type	Width	Reset	Description
DCS_LEDS	0x104	RW	8	0x00000000	Controls platform LEDS or other platform-specific indication methods.
DCS_SW	0x108	RO	8	0x00000000	Reads the value of platform switches or other platform-specific configuration methods.

The following table shows the DCSCB Interrupt Generator Registers.[a]

Table 3-25 DCSCB Interrupt Generator Registers

Name	Offset	Type	Width	Reset	Description
INT_CTRL	0x120	RW	2	0x00000000	Controls generation of interrupts from the interrupt generation trickbox.
INT_FREQ	0x124	RW	10	0x00000000	Controls the frequency of timer-generated interrupts.
INT_TYPE0	0x130	RW	32	0x00000000	Configures the interrupt generator to use level or edge-triggered interrupts for each interrupt line.
INT_TYPE1	0x134	RW	32	0x00000000	Configures the interrupt generator to use level or edge-triggered interrupts for each interrupt line. If the interrupt generator implements fewer than the maximum 128 interrupts, higher order registers corresponding to unimplemented interrupts are RAZ/WI.
INT_TYPE2	0x138	RW	32	0x00000000	
INT_TYPE3	0x13C	RW	32	0x00000000	
INT_GENERATE	0x140	WO	1	-	Generates the next interrupt.
INT_NUMBER	0x144	RO	8	0x00000000	Number of the next interrupt.
INT_ACK	0x148	WO	1	-	Acknowledges all the generated interrupts.
INT_SEQ0 – INT_SEQ127	0x200-0x3FC	RW	7	0x00000000	Sequence number for all generated interrupts.

The following table shows the DCSCB ID Registers.

Table 3-26 DCSCB ID Registers

Name	Offset	Type	Width	Reset	Description
DCS_AID	0xFF8	RO	32	-	Dual Cluster System auxiliary platform ID register.
DCS_ID	0xFFC	RO	32	-	Dual Cluster System platform ID register.

The following table shows the DCSCB Debug Control Registers.

Table 3-27 DCSCB Debug Control Registers

Name	Offset	Type	Width	Reset	Description
DBG_RST_CTRL	0x520	RW	32	0x00000000	Debugger reset control register that defines the resets to assert. This register is intended for debug access only. Although software running on the system can currently access it, this might change in the future.
DBG_RST_SCHED	0x018	RW	9	0x00000100	Debugger reset schedule register to control when to assert resets. This register is intended for debug access only. Although software running on the system can currently access it, this might change in the future.

This section contains the following subsections:

- *3.21.1 Reset hold registers, RST_HOLD0 and RST_HOLD1* on page 3-52.

[a] Interrupt generator registers are RAZ/WI if the interrupt generator is not present.

3.21.1 Reset hold registers, RST_HOLD0 and RST_HOLD1

The reset hold registers enable a processor to place any core or cluster into a reset state.

The reset is scheduled for when the core to be reset enters the STANDBYWFI state, and remains in reset until another core brings it out of reset.

The following reset hold registers exist:

RST_HOLD0 Toggles resets for the Cortex-A15 cluster.

RST_HOLD1 Toggles resets for the Cortex-A7 cluster.

The following table shows the bits in the RST_HOLDx registers that force a particular reset.

For all supported resets, the reset is held for as long as the corresponding bit in the RST_HOLDx register is HIGH.

Table 3-28 RST_HOLDx register bit assignments

Bits	Name	Type	Description
[8]	CLUSTER_RESET	RW	Write 1 to this bit to reset the entire cluster. The reset is scheduled to occur when every core in the cluster has entered the STANDBYWFI state. The cluster is then held in reset until 0 is written to the field. A cluster reset places each core into a power-on reset and resets the interrupt controller and L2 logic.
[7:4]	CPU_PORESET	RW	Write 1 to bit n to assert a core power-on reset for core n. The reset is scheduled to occur when core n enters the STANDBYWFI state. The reset line is then held indefinitely until 0 is written to the field. A core power-on reset resets the core logic, NEON/VFP and debug logic.
[3:0]	CPU_RESET	RW	Write 1 to bit n to assert a core reset for core n. The reset is scheduled to occur when core n enters the STANDBYWFI state. The reset line is then held indefinitely until 0 is written to the field. A core reset resets the core logic and NEON/VFP.

Power-on behavior

At power-on, the CLUSTER_RESET fields of the RST_HOLDx registers take their values from the CFG_ACTIVECLUSTER static configuration input.

This means that you can choose the cluster, or clusters, that are active at power-on without re-generating the system. *3.22 Reset architecture* on page 3-67 describes the CFG_ACTIVECLUSTER static configuration input.

Writing to RST_HOLDx

Writes to RST_HOLDx might not result in the reset being asserted or de-asserted immediately.

Certain conditions must be met before the requested action can be performed, and the reset controller completes the action as soon as possible. The following figure shows the simplest sequence of events when a bit of RST_HOLDx is written from 0 to 1.

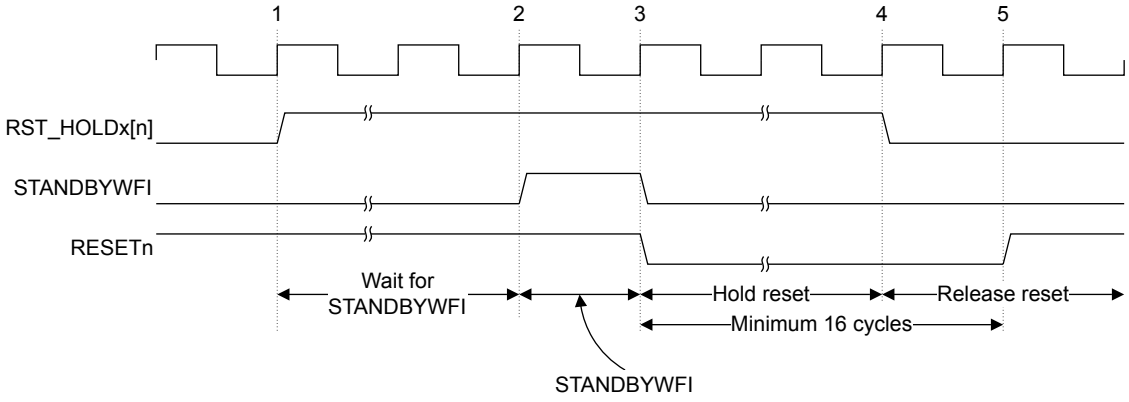

Figure 3-1 RST_HOLDx sequence

In the figure above, the following actions occur:

1. Bit n of **RST_HOLDx** is written to request the assertion of the reset. The reset controller waits for the **STANDBYWFI** signal from core n to go HIGH.
2. **STANDBYWFI** from core n goes HIGH. The reset controller can now assert the reset.
3. The reset is held for as long as **RST_HOLDx[n]** is HIGH.
4. When **RST_HOLDx[n]** is de-asserted, the reset is released.
5. Core n is out of reset.

The figure above shows the simple case where the requested reset does not depend on other resets.

Note

- A cluster reset is not asserted until **STANDBYWFI** has been asserted for every core in the cluster.
- Writing a bit of **RST_HOLDx** from 1 to 0 only releases the reset when all higher-level resets have been released. For example, higher-level resets always have priority over lower-level resets.
- Writing a bit of **RST_HOLDx** from 0 to 1 implicitly resets lower-level resets. For example, writing 1 to only **PORESET** for core 1 resets both **PORESET** and **CPURESET** for core 1.

It is possible to withdraw a requested reset before it has been asserted if the bit of **RST_HOLDx** is written from 1 to 0 during the 'wait for STANDBYWFI' window. This is the time between points 1 and 2 in the figure above. If the **RST_HOLDx** request is cleared during this window, then the reset is not asserted, provided that there are no higher-level resets that take priority. When a reset has been asserted, the reset controller ensures that the reset is applied for a minimum of 16 clock cycles.

If the **RST_HOLDx** request is cleared when the reset has been asserted for fewer than 16 clock cycles, that is, the number of clock cycles between points 3 and 4 in the figure above, is fewer than 16, then the reset is released only when 16 clock cycles have passed.

During this time, writing 1 back to **RST_HOLDx** cancels the de-assertion of the reset.

Writing 1 to a bit of **RST_HOLDx** that is already 1, or that is reset implicitly because of a higher-level reset, has no effect on the active resets.

Writing 0 to a bit of **RST_HOLDx** that is already 0 has no effect.

Reading from RST_HOLDx

Reading the RST_HOLDx register returns the value that was last written to the register.

To determine the resets that are currently active, see *3.21.3 Reset Status Registers, RST_STAT0 and RST_STAT1* on page 3-57.

Software sequence to assert reset using RST_HOLDx

For core A to reset Core B, software must perform the following actions:

Note

Core A and core B can be the same core.

1. Core A writes to the appropriate RST_HOLDx register to request the reset of core B or its cluster.
2. Core B programs the GIC of its cluster to prevent IRQs and FIQs being asserted to Core B.
3. Core B executes a WFI. After Core B executes WFI, its STANDBYWFI output goes HIGH. At this point, the reset controller asserts reset until the register that caused it is cleared, and always for a minimum of 16 clock cycles.

Implicit resets based on requested reset levels

Requesting a cluster reset implies that all core in the cluster are to have **PORESETn** applied.

Applying **PORESETn** to a core implies that the core reset is applied. See *3.22 Reset architecture* on page 3-67. Writing 1 to a field of RST_HOLDx therefore implicitly forms a reset-assertion request for lower-level fields, as the following table shows. In the table, the symbols w, x, y, and z each represent a logic 1 or 0. A dash represents a don't care value, and a pipe, |, represents a logical OR function. The table also references the RST_STATx registers, that discover the current set of applied resets. See *3.21.3 Reset Status Registers, RST_STAT0 and RST_STAT1* on page 3-57.

Table 3-29 Effect of RST_HOLDx write values

Value written to RST_HOLDx			Value read from RST_STATx when serviced			Description
[8]	[7:4]	[3:0]	[8]	[7:4]	[3:0]	-
0	0000	0000	0	0000	0000	All resets are de-asserted.
0	0000	wxyz	0	0000	wxyz	Core resets are asserted for the requested cores, and all other reset lines are de-asserted.
0	wxyz	abcd	0	wxyz	abcd | wxyz	Power-on resets for the requested cores are asserted, together with core reset for the same cores, regardless of the value written in bits [3:0]. Core resets for other explicitly-requested cores are also asserted. All other resets are de-asserted.
1	----	----	1	1111	1111	A cluster reset is asserted, and this means that the core reset and PORESET lines are also asserted.

3.21.2 Software Reset Register, SYS_SWRESET

Use the Software Reset Register to reset the system from a given level.

See *3.22 Reset architecture* on page 3-67 for information about reset levels in the system. The resets are applied, then automatically released. The following table shows the bit assignments for the software reset register.

Table 3-30 SYS_SWRESET Register bit assignments

Bits	Name	Type	Description
[23:20]	CORES1	WO	Defines the cores in the Cortex-A7 cluster to reset when LEVEL is set to 1 and CLUSTER_LEVEL0 is set to `0b00` or `0b01`. One bit exists for each core. A core is reset if the corresponding bit is 1. **Note** If fewer than four cores are implemented in a cluster, then the higher order bits are ignored.
[19:16]	CORES0	WO	Defines the cores in the Cortex-A15 cluster to reset when LEVEL is set to 1 and CLUSTER_LEVEL0 is set to `0b00` or `0b01`. One bit exists for each core. A core is reset if the corresponding bit is 1. **Note** If fewer than four cores are implemented in a cluster, then the higher order bits are ignored.
[15:12]	-	-	-
[11:10]	CLUSTER_LEVEL1	WO	For a Cortex-A7 cluster reset, defines the reset level: `0b00` Individual core reset. `0b01` Individual core power-on-reset. `0b10` Full cluster reset. `0b11` Reserved.
[9:8]	CLUSTER_LEVEL0	WO	For a Cortex-A15 cluster reset, defines the reset level: `0b00` Individual core reset. `0b01` Individual core power-on-reset. `0b10` Full cluster reset. `0b11` Reserved.
[7:6]	-	-	-
[5:4]	CLUSTERS	WO	Enables reset for each cluster when LEVEL is set to a cluster reset: **Bit[4]** Set to 1 to reset the Cortex-A15 cluster. **Bit[5]** Set to 1 to reset the Cortex-A7 cluster.
[3]	-	-	-
[2]	SWRESET	WO	Write 1 to apply the software reset.
[1]	-	-	This bit is reserved for future reset levels.
[0]	LEVEL	WO	Choose what to reset: **0** System reset. Resets the system and both clusters. **1** Core or cluster reset. Resets the clusters and cores specified in the CLUSTERS field.

Writing to SYS_SWRESET forms a request to the reset controller that is serviced as soon as possible.

Writing to SYS_SWRESET while an operation is already in progress modifies that operation as far as possible. Any resets that have not been asserted are no longer asserted, but resets that have been asserted already remain active for at least 16 clock cycles.

Writing to SYS_SWRESET

When a reset is requested using SYS_SWRESET:

- A core reset is only asserted when that core has entered the STANDBYWFI state.
- A cluster reset is only asserted when all cores in the cluster have entered a STANDBYWFI state.
- If a reset request applies to more than one core, the resets of the core might be applied at different times depending on when they each enter the STANDBYWFI state.
- The resets are all released simultaneously.
- Resets are asserted for a minimum of 16 clock cycles.
- A complete system reset does not wait for any STANDBYWFI signals to be asserted.

Software sequence to assert reset using SYS_SWRESET

For core A to reset Core B, software must perform the following actions:

——— **Note** ———

Core A and core B can be the same core.

———————————

1. Core A writes to the appropriate SYS_SWRESET fields to request the reset of core B or its cluster.
2. Core B programs the GIC of its cluster to prevent IRQs and FIQs being asserted to Core B.
3. Core B executes a WFI. After Core B executes WFI, its STANDBYWFI output goes HIGH. At this point, the reset controller asserts reset for a minimum of 16 cycles and then de-asserts it.

Interaction with RST_HOLDx

The SYS_SWRESET and RST_HOLDx registers operate independently.

Resetting a core or cluster using SYS_SWRESET does not cause any resets that are held using RST_HOLDx to be released. However, using SYS_SWRESET to assert a complete system reset resets all DCSCB registers to their default values, including RST_HOLDx. This means that clusters that are held in reset might be released, depending on the value of the **CFG_ACTIVECLUSTER** input.

The following table shows the components that are reset depending on the values written to the SYS_SWRESET register. In the table, the symbols w, x, y, and z each represent a logic 1 or 0. A dash represents a don't care value.

Table 3-31 Components reset depending on values written to SYS_SWRESET register

LEVEL	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CLUSTERS	-	00	01	01	01	10	10	10	11	11	11	11	11	11	11	11	11
CLUSTER_LEVEL0	-	-	00	01	10	-	-	-	00	01	10	00	01	10	00	01	10
CLUSTER_LEVEL1	-	-	-	-	-	00	01	01	00	00	00	01	01	01	10	10	10
CORES0	-	-	abcd	abcd	abcd	-	-	-	abcd	abcd	-	abcd	abcd	-	abcd	abcd	-
CORES1	-	-	-	-	-	wxyz	wxyz	wxyz	wxyz	wxyz	wxyz	wxyz	wxyz	wxyz	-	-	-
Cortex-A15 core 0 reset	-	-	d	d	d	-	-	-	d	d	-	d	d	-	d	d	-
Cortex-A15 core 1 reset	-	-	c	c	c	-	-	-	c	c	-	c	c	-	c	c	-
Cortex-A15 core 2 reset	-	-	b	b	b	-	-	-	b	b	-	b	b	-	b	b	-
Cortex-A15 core 3 reset	-	-	a	a	a	-	-	-	a	a	-	a	a	-	a	a	-
Cortex-A15 core 0 PORESET	-	-	-	d	d	-	-	-	-	d	-	-	d	-	-	d	-
Cortex-A15 core 1 PORESET	-	-	-	c	c	-	-	-	-	c	-	-	c	-	-	c	-

Table 3-31 Components reset depending on values written to SYS_SWRESET register (continued)

LEVEL	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Cortex-A15 core 2 PORESET	-	-	-	b	b	-	-	-	-	b	-	-	b	-	-	b	-
Cortex-A15 core 3 PORESET	-	-	-	a	a	-	-	-	-	a	-	-	a	-	-	a	-
Cortex-A15 cluster reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Cortex-A7 core 0 reset	-	-	-	-	-	z	z	z	z	z	z	z	z	z	-	-	-
Cortex-A7 core 1 reset	-	-	-	-	-	y	y	y	y	y	y	y	y	y	-	-	-
Cortex-A7 core 2 reset	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	-	-
Cortex-A7 core 3 reset	-	-	-	-	-	w	w	w	w	w	w	w	w	w	-	-	-
Cortex-A7 core 0 PORESET	-	-	-	-	-	-	z	z	-	-	-	z	z	z	-	-	-
Cortex-A7 core 1 PORESET	-	-	-	-	-	-	y	y	-	-	-	y	y	y	-	-	-
Cortex-A7 core 2 PORESET	-	-	-	-	-	-	x	x	-	-	-	x	x	x	-	-	-
Cortex-A7 core 3 PORESET	-	-	-	-	-	-	w	w	-	-	-	w	w	w	-	-	-
Cortex-A7 cluster reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
System	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

3.21.3 Reset Status Registers, RST_STAT0 and RST_STAT1

The read-only Reset Status Registers are RST_STAT0 and RST_STAT1.

RST_STAT0 Returns the reset status of the Cortex-A15 cluster.

RST_STAT1 Returns the reset status of the Cortex-A7 cluster.

Each read-only RST_STATx register can determine the resets that the reset controller is currently asserting.

Table 3-32 RST_STATx Register bit assignments

Bits	Name	Type	Description
[8]	CLUSTER_RESET	RO	Reads as 1 if the cluster is held in a cluster reset. Otherwise, it reads as 0.
[7:4]	CPU_PORESET	RO	Each bit n reads as 1 if core n is held in core power-on reset. Otherwise, it reads as 0.
[3:0]	CPU_RESET	RO	Each bit n reads as 1 if core n is held in core reset. Otherwise, it reads as 0.

Note

It is not possible to read whether a system reset or a power-on reset is active because both core clusters are fully reset in both of these cases.

3.21.4 Cortex-A15 static configuration read and write Registers

The Cortex-A15 static configuration read and write registers provide a method to observe and change the values driven onto the static configuration input pins of the Cortex-A15 processor.

The static configuration inputs must only be changed when the processor is under reset. Therefore, the following registers control and observe the values:

- The read-only static configuration read register shows the values currently driven onto the static input pins.
- The read and write static configuration write register contains the value that is to be driven onto the static configuration inputs when the processor is next reset.

When the processor is reset, the contents of the CLUSTER0_CFG_W register are copied to the CLUSTER0_CFG_R register, and this drives the static configuration input pins. The following table shows the bit assignments for the registers. Both registers contain the same bit fields. Only the access type is different.

Table 3-33 CLUSTER0_CFG_R and CLUSTER0_CFG_W Register bit assignments

Bits	Name	CLUSTER0_CONFIG_R Access	CLUSTER0_CONFIG_W Access	Description
[19:16]	CLUSTER0_CLUSTERID	RO	RW	The value driven onto the **CLUSTERID** static input pins.
[15:13]	-	RAZ	RAZ/WI	Reserved.
[12]	CLUSTER0_IMINLN	RO	RW	The value of the **IMINLN** static input pin.
[11:8]	CLUSTER0_CFGTE	RO	RW	The value driven onto the **CFGTE** static input pins. One bit exists for each core in the cluster.
[7:4]	CLUSTER0_VINITHI	RO	RW	The value driven onto the **VINITHI** static input pins. One bit exists for each core in the cluster.
[3:0]	CLUSTER0_CFGEND	RO	RW	The value driven onto the **CFGEND** static input pins. One bit exists for each core in the cluster.

The default values of the Cortex-A15 static configuration registers:

- Set the exception endianness to little-endian for every core.
- Disable high exception vectors for every core, that is, exception vectors start at address `0x00000000`.
- Disable Thumb exceptions for every core, that is, exceptions are entered in ARM state.
- Do not prevent write access to secure CP15 registers.
- Set the cluster ID to `0x0`.

3.21.5 Cortex-A7 static configuration read and write Registers

The Cortex-A7 static configuration read and write registers determine and control the values driven onto the static configuration input pins of Cortex-A7.

The behavior of these registers is the same as the Cortex-A15 static configuration read and write registers. See *3.21.3 Reset Status Registers, RST_STAT0 and RST_STAT1* on page 3-57 for information.

The following table shows the bit assignments for both registers.

Table 3-34 CLUSTER1_CFG_R and CLUSTER1_CFG_W Register bit assignments

Bits	Name	CLUSTER1_CONFIG_R Access	CLUSTER1_CONFIG_W Access	Description
[19:16]	CLUSTER1_CLUSTERID	RO	RW	The value driven onto the **CLUSTERID** static input pins.
[15:12]	-	RAZ	RAZ/WI	Reserved.
[11:8]	CLUSTER1_TEINIT	RO	RW	The value driven onto the **TEINIT** static input pins. One bit exists for each core in the cluster.
[7:4]	CLUSTER1_VINITHI	RO	RW	The value driven onto the **VINITHI** static input pins. One bit exists for each core in the cluster.
[3:0]	CLUSTER1_CFGEND	RO	RW	The value driven onto the **CFGEND** static input pins. One bit exists for each core in the cluster.

The default values of the Cortex-A7 static configuration registers:

- Set the exception endianness to little-endian for every core.
- Disable high exception vectors for every core, that is, exception vectors start at address `0x00000000`.
- Disable Thumb exceptions for every core, that is, exceptions are entered in ARM state.
- Do not prevent write access to secure CP15 registers.
- Set the cluster ID to `0x1`.

3.21.6 System Static Configuration Read Register, DCS_CFG_R

The DCS_CFG_R register discovers the values that were applied to system static configuration input pins when the system was powered on, that is, system power-on reset.

The following table shows the bit assignments.

Table 3-35 DCS_CFG_R Register bit assignments

Bits	Name	Type	Description
[31:28]	NUM_CPU3	RO	Reserved for the NUM_CPU configuration for cluster 3. This system does not contain a cluster 3 and this field is RAZ.
[27:24]	NUM_CPU2	RO	Reserved for the NUM_CPU configuration for cluster 2. This system does not contain a cluster 2 and this field is RAZ.
[23:20]	NUM_CPU1	RO	Returns the NUM_CPU configuration for Cluster 1, Cortex-A7 cluster: `0x0` Cluster not present. `0x1` One core cluster. `0x2` Two core clusters. **...** And so on.

Table 3-35 DCS_CFG_R Register bit assignments (continued)

Bits	Name	Type	Description
[19:16]	NUM_CPU0	RO	Returns the NUM_CPU configuration for Cluster 0, Cortex-A15 cluster: `0x0` Cluster not present. `0x1` One core cluster. `0x2` Two core clusters. **...** And so on.
[15:2]	-	RAZ	Reserved.
[1:0]	CFG_ACTIVECLUSTER	RO	Returns the value that was driven on the **CFG_ACTIVECLUSTER** configuration inputs at the last system power-on reset.

3.21.7 LED Control Register, DCS_LED

The DCS_LED Register can control up to eight platform LEDs, or other platform notification or status outputs.

A platform might not necessarily use all eight available bits. All LEDs are general-purpose, so the common platform specification assigns no specific meaning to any LED. The following table shows the bit assignments of the DCS_LED register.

Table 3-36 DCS_LED Register bit assignments

Bits	Name	Type	Description
[7:0]	LED	RW	Each bit controls an individual LED or status output. Reads back as the last value written to the register.

3.21.8 Switch Status Register, DCS_SW

This read-only register reads the value of up to eight general-purpose platform switches.

A platform might not necessarily use all eight switches. The following table shows the bit assignment of the DCS_SW register.

Table 3-37 DCS_SW Register bit assignments

Bits	Name	Type	Description
[7:0]	SW	RO	Each bit reads the value of a general-purpose switch.

3.21.9 Dual Cluster System Auxiliary Platform ID Register, DCS_AID

The read-only DCS_AID register returns information about the platform to the Versatile Express platform configuration controller.

It also contains other status fields that software or users of the platform can use. The following table shows the bit assignments.

Table 3-38 DCS_AID Register bit assignments

Bits	Name	Type	Description
[31:24]	BUILD	RO	A serial build number for all DCS releases, starting at zero. The build number uniquely identifies each platform.
[23:14]	-	RO	Reserved. RAZ.

Table 3-38 DCS_AID Register bit assignments (continued)

Bits	Name	Type	Description
[13:12]	INTGEN_INTS	RO	Number of interrupts that the interrupt generation trickbox controls: `0b00` 32 interrupts. `0b01` 64 interrupts. `0b10` 96 interrupts. `0b11` 128 interrupts.
[11]	INTGEN_PRESENT	RO	Reads as: **1** If the interrupt generation trickbox is present. **0** Otherwise.
[10]	SW_ENABLE	RO	Reads as: **1** If the platform supports reading switch or configuration inputs from the DCS_SW register. **0** Otherwise.
[9]	LED_ENABLE	RO	Reads as: **1** If the platform supports setting LEDs or status information through the DCS_LED register. **0** Otherwise.
[8]	-	RO	Reserved. RAZ.
[7:0]	CFGREGNUM	RO	Reads as zero to indicate that no additional Versatile Express user-defined configuration commands exist.

The BUILD field identifies a specific release of a dual cluster system platform implementation. No two releases of a platform implementation have a same build number, regardless of the version of the system specification that it implements. Build numbers are only unique within a particular implementation, and there is no relationship between build numbers of different platform implementations. For example, an FPGA implementation with a build number of `0x8'h05` is not necessarily based on the same dual cluster system source version as build number `0x8'h05` of a software model.

3.21.10 Dual Cluster System Platform ID Register, DCS_ID

The read-only DCS_ID register provides version information for the dual cluster system implementation.

The following table shows the bit assignments.

Table 3-39 DCS_ID Register bit assignments

Bits	Name	Type	Description
[31:24]	IMPLEMENTER	RO	Reads as `0x8'h41` to indicate ARM Limited.
[23:20]	VARIANT	RO	Identifies the implementation target. Reads as: `0x4'h0` For Versatile Express. `0x4'h1` For software model.
[19:16]	ARCHITECTURE	RO	Identifies the DCS architecture: `0x4'h0` Reserved. `0x4'h1` Cortex-A15 plus Cortex-A15 system. `0x4'h2` Cortex-A15 plus Cortex-A7 system.

Table 3-39 DCS_ID Register bit assignments (continued)

Bits	Name	Type	Description
[15:4]	-	RO	Reserved.
[3:0]	REVISION	RO	RAZ

3.21.11 Interrupt Generator Control Register, INT_CTRL

The INT_CTRL register starts and stops the generation of interrupts from the interrupt generation trickbox.

See *3.22 Reset architecture* on page 3-67 for a complete description of how to set up and use trickbox interrupts. The following table shows the bit assignments for the INT_CTRL register. If the interrupt generator is not present, then this register is RAZ/WI.

Table 3-40 INT_CTRL Register bit assignments

Bits	Name	Type	Description
[1]	TIMER_EN	RW	Enables automatic generation of interrupts at regular intervals: **0** Do not generate timer-based interrupts. Interrupts are not generated automatically, but you can still generate them manually. **1** Generate regular interrupts using the timer, if ENABLE is HIGH.
[0]	ENABLE	RW	Enable interrupts from the interrupt trickbox: **0** The trickbox does not generate interrupts, neither automatically nor explicitly-requested. **1** The trickbox can generate interrupts.

The timer that controls automatic assertion of interrupts runs when all the following are true:

- INT_CTRL.ENABLE is HIGH.
- INT_CTRL.TIMER_EN is HIGH.
- Additional interrupts are available for generation, indicated by INT_NUMER.SATURATED being LOW.

If any of the above conditions are not met, then the timer does not run and interrupts are not asserted automatically. When the timer is paused because INT_CTRL.ENABLE is LOW or INT_CTRL.TIMER_EN is LOW, it resumes from its previous position when the appropriate enable is set HIGH.

If the timer is halted because no additional interrupts are available for generation, the timer can only start again when all the interrupts are acknowledged using the INT_ACK register.

3.21.12 Interrupt Generator Interrupt Frequency Register, INT_FREQ

The INT_FREQ register controls the frequency at which the interrupt generator automatically asserts interrupts, if this functionality is enabled in the INT_CTRL register.

The following table shows the bit assignments for the INT_FREQ register.

Table 3-41 INT_FREQ Register bit assignments

Bits	Name	Type	Description
[9:0]	FREQUENCY	RW	Controls the frequency of timer-generated interrupts, when INT_CTRL.TIMER_EN is HIGH. Set this field to n to generate interrupts every n+1 cycles when the timer is enabled. This means that a value of: **0** Generates an interrupt every clock cycle. **1** Generates an interrupt every second clock cycle. **2** Generates an interrupt every third clock cycle. **...** And so on.

Writing any value to the INT_FREQ register causes the internal interrupt timer to be cleared, but it automatically restarts if it is enabled. The INT_FREQ register can be updated at any time, even if interrupt generation is currently enabled. If the interrupt generation trickbox is not present, then this register is RAZ/WI.

3.21.13 Interrupt Generator Interrupt Type Registers, INT_TYPEx

The INT_TYPEx registers configure whether the interrupt generator trickbox generates level-sensitive or edge-sensitive interrupts for each of the 128 supported interrupts:

INT_TYPE0 Configures interrupts 0-31, one bit for each interrupt.
INT_TYPE1 Configures interrupts 32-63, one bit for each interrupt.
INT_TYPE2 Configures interrupts 64-95, one bit for each interrupt.
INT_TYPE3 Configures interrupts 96-127, one bit for each interrupt.

Each register consists of 32 read and write bit fields, where each bit field configures an interrupt. Bit n of INT_TYPEx configures interrupt number 32x + n, and can take the following values:

0 Generate a level interrupt for this line. An interrupt on this line pulls the interrupt signal HIGH and keeps it HIGH until it is acknowledged.

1 Generate pulse, edge-triggered, interrupts on this line. An interrupt on this line causes the interrupt signal to go HIGH for one clock cycle.

It is recommended to only update this register prior to starting interrupt generation, when INT_CTRL.ENABLE is LOW and interrupts from any previous generation runs have been acknowledged. If the interrupt generator is configured to support fewer than 128 interrupts, then the registers corresponding to unsupported interrupts are RAZ/WI.

3.21.14 Interrupt Generator Generate Register, INT_GENERATE

The INT_GENERATE register causes an interrupt to be generated on request.

The generated interrupt is the next interrupt from the interrupt sequence list. If all possible interrupts have been generated, then requesting an additional interrupt has no effect. Writing to this register has no effect if INT_CTRL.ENABLE is LOW.

The following table shows the bit assignments for the INT_GENERATE register.

Table 3-42 INT_GENERATE Register bit assignments

Bits	Name	Type	Description
[0]	GENERATE	WO	Write 1 to this field to generate the next interrupt

This register is RAZ/WI if the interrupt generator is not present.

3.21.15 Interrupt Generator Interrupt Number Register, INT_NUMBER

Read the INT_NUMBER register to determine the sequence number of the next interrupt that the interrupt generator trickbox is to generate.

The following table shows the bit assignments for the INT_NUMBER register.

Table 3-43 INT_NUMBER Register bit assignments

Bits	Name	Type	Description
[7]	SATURATED	RO	Reads as: **0** If additional interrupts can be generated. **1** If all possible interrupts have been generated.
[6:0]	NUMBER	RO	Returns the sequence number for the next interrupt, starting at `0x0`. When all possible interrupts have been generated, every bit in this field is HIGH, regardless of the configured maximum number of interrupts.

Bits [7:0] of this register can be used together to determine how many interrupts have been generated:

`0x8'hFF` All possible interrupts have been generated.

Any other value a n interrupts have been generated.

The number of possible interrupts depends on the build configuration of the interrupt generator. If the interrupt generator is not present, then this register is RAZ/WI.

3.21.16 Interrupt Generator Sequencing Registers, INT_SEQx

The interrupt generator sequencing registers configure the sequence for trickbox-generated interrupts.

You must program these registers with a valid sequence before enabling the interrupt generator. There are up to 128 INT_SEQx registers, one for each supported interrupt.

If the interrupt generator supports fewer than the maximum 128 interrupts, then registers corresponding to unavailable interrupts are RAZ/WI.

If the interrupt generator is not present, all registers are RAZ/WI.

Each INT_SEQx register has the bit assignment that the following table shows. INT_SEQx configures the interrupt line that is asserted when the xth interrupt is requested.

The following table shows the bit assignments for the INT_SEQx register.

Table 3-44 INT_SEQx Register bit assignments

Bits	Name	Type	Description
[6:0]	NUMBER	RW	Configures the interrupt line that is to be asserted for this position in the sequence. If the interrupt generator is configured to support fewer than 128 interrupts, then the higher-order bits of this field are RAZ/WI. The width of the field is $\log_2$(Number of supported interrupts).

Before generation starts, the NUMBER field of every INT_SEQx register must be unique, otherwise results are UNPREDICTABLE. Writing to these registers when interrupt generation is enabled, or with unacknowledged interrupts, gives UNPREDICTABLE behavior. If the interrupt generator is configured to support fewer than 128 interrupts, then the registers corresponding to the unavailable interrupts are RAZ/WI.

3.21.17 Interrupt Generator Acknowledge Register, INT_ACK

The INT_ACK register acknowledges all level-sensitive interrupts that have been generated, and resets the INT_NUMBER register to zero.

It also resets the internal timer that is used for automatic interrupt generation. After acknowledging all interrupts using this register, the interrupt generator can be used again in another generation run.

The following table shows the bit assignments for the INT_ACK register.

Table 3-45 INT_ACK Register bit assignments

Bits	Name	Type	Description
[0]	ACK	WO	Write 1 to acknowledge all interrupts and reset the INT_NUMBER register to zero.

If the interrupt generator is not present, then this register is RAZ/WI.

3.21.18 Debug Reset Control and Reset Schedule Registers

The debug reset control and schedule registers are used together to reset some or all of the system.

The following tables show the bit assignments of the Debug Reset Control and Reset Schedule registers.

Table 3-46 RST_CTRL Register bit assignments

Bits	Name	Type	Description
[31]	SYS_DBGRST	RW	Schedule a debug system reset when 1.
[30]	SYS_PORESET	RW	Schedule a system-wide power-on reset when 1.
[29]	-	SBZP	Reserved.
[28]	CLUSTER1_SCURESET	RW	Schedule Cortex-A7 core resets. These fields exhibit the same behavior as the Cortex-A15 reset fields.
[27:24]	CLUSTER1_DBGRESET	RW	
[23:20]	CLUSTER1_CORERESET	RW	
[19:18]	-	SBZP	Reserved.
[17]	CLUSTER0_L2RESET	RW	Schedule Cortex-A15 resets. When a field is 1, a reset pulse is scheduled for when the RST_SCHED timer expires. **Note** If the Cortex-A15 cluster is configured to contain fewer than four cores, then the high-order bits of each field corresponding to unimplemented cores are RAZ/WI.
[16]	CLUSTER0_PRESET	RW	
[15:12]	CLUSTER0_DBGRESET	RW	
[11:8]	CLUSTER0_CXRESET	RW	
[7:4]	CLUSTER0_CORERESET	RW	
[3:0]	CLUSTER0_CPUPORESET	RW	

Table 3-47 RST_SCHED Register bit assignments

Bits	Name	Type	Description
[8]	DISABLE	RW	Disables the reset scheduler timer. This bit auto-sets when the TIMER field reaches zero.
[7:0]	TIMER	RW	Reset schedule timer. This field auto-decrements if the DISABLE field is 0. When the timer reaches zero, the interrupts configured in the RST_CTRL register are pulsed. Writing 0 to this field, and 0 to the DISABLE field pulses the interrupts immediately.

To use the registers:

- Write 1 to each bit of the RST_CTRL register that corresponds to the reset that is to be pulsed.
- Write a value to the TIMER field of the RST_SCHED register. If the DISABLE bit is written as 0, or if it is already 0, the timer automatically begins to decrement. When the TIMER reaches zero, any

resets that are scheduled in the RST_CTRL register are asserted, and the DISABLE field resets to 1 to prevent subsequent resets.
- Writing 0 to the TIMER field causes the resets to be pulsed immediately, if the DISABLE field is 0.

If the TIMER field of the RST_SCHED register is written together with a value of 0 in the DISABLE field in a single transaction, then the reset scheduler activates immediately. Otherwise, the TIMER only decrements when 0 is written to the DISABLE field later.

3.22 Reset architecture

The platform layer defines several levels of reset.

The following levels of reset exist, in increasing order:

Core reset
Core reset, including NEON and VFP.
Core power-on reset
Core, NEON, VFP, and CPU debug.
Cluster reset
Entire core cluster including L2 and interrupt controller.
System reset
Cluster reset for all clusters plus dual cluster system.
Power-on reset
Entire system and platform including the debug subsystem.

Debug resets

The processors contain the following debug-related reset signals:

nDBGRESET[n:0]
Where n is the number of cores in the cluster.
nPRESETDBG
In the Cortex-A7, this reset only exists at the integration layer, that integrates the CoreSight subsystem. Conversely, the Cortex-A15 processor integrates the CoreSight components at the Cortex-A15 level and therefore contains the **nPRESETDBG** signal.

nDBGRESET[n:0] resets the debug logic for each core, including breakpoint and watchpoint logic. **nPRESETDBG** resets the CoreSight debug subsystem, including the CTIs, CTMs, and debug APB. A core power-on reset, cluster reset, and system reset assert the **nDBGRESET** lines to the appropriate cores, but only a power-on reset asserts **nPRESETDBG**. All components in the CoreSight debug subsystem are in the same **nPRESETDBG** domain and cannot be reset independently of each other.

Each successive reset level from the list above is a superset of the previous reset level, so a reset level also resets everything in the levels below it.

For example, a core power-on reset includes everything that is reset by a core reset, and a system reset includes a cluster reset, core power-on reset, and core reset for each cluster. A memory-mapped register can explicitly control platform resets, but the reset controller also manages platform resets. The reset controller is closely coupled with the power controller.

See *3.21.1 Reset hold registers, RST_HOLD0 and RST_HOLD1* on page 3-52 and *3.21.2 Software Reset Register, SYS_SWRESET* on page 3-54 for information about the reset registers.

At power-on, the dual cluster system platform issues a complete power-on reset to reset:
- Core clusters.
- Interconnect.
- Debug.
- Peripherals.

When the power-on reset sequence is complete, a static configuration option determines whether the Cortex-A7 cluster or the Cortex-A15 cluster, or both, exit reset.

3.23 Interrupt Generation Trickbox

The interrupt generation trickbox is an extension of the *Dual Cluster System Configuration Block* (DCSCB) that enables interrupts to be generated either on demand, by writing to a control register, or automatically at regular intervals, using a programmable timer.

The trickbox controls various interrupt lines and asserts, at most, one interrupt per cycle, in an order that you can configure. Interrupts can be either:

- Level-sensitive.
- Edge-sensitive.

3.15 Bus consistency messages on page 3-43 describes the DCSCB that contains registers to configure and control the interrupt generation trickbox. The interrupt generation trickbox can control up to 128 interrupt lines.

When interrupt generation starts:
1. The index in the INT_SEQ0 register provides the first interrupt that the trickbox asserts.
2. The index in the INT_SEQ1 register provides the second interrupt that the trickbox asserts.

It is a prerequsite that you have already programmed the INT_SEQx registers with the required sequence before you start the interrupt generation.

This section contains the following subsections:
- *3.23.1 Using the Interrupt Generation Trickbox* on page 3-68.

3.23.1 Using the Interrupt Generation Trickbox

You can generate interrupts either: on demand, by writing to a control register, or automatically at regular intervals, using a programmable timer.

To set up and use the interrupt generation trickbox:

Procedure

1. Program the INT_SEQx registers with integers in the range 0-(INTGEN_INTS-1).
   These numbers refer to the interrupt line that fires each time the trickbox generates an interrupt. The value in each INT_SEQx register must be unique.
2. Program the INT_TYPEx registers to configure whether each interrupt is level-triggered or edge-triggered, as required.
3. If the trickbox must automatically generate interrupts at regular intervals, set INT_FREQ to the required frequency.

   The values are as follows:

   0 A value of 0 means that an interrupt is generated on every clock cycle.

   1 A value of 1 means that an interrupt is generated every other clock cycle.

   The timer does not start until interrupt generation is globally enabled.
4. Set INT_CTRL.ENABLE to 1 to enable the generation of interrupts.
   Interrupts start to fire automatically if INT_CTRL.TIMER_EN is 1. Otherwise, manual generation of interrupts is enabled.
5. Run the code of interest across which the interrupts are to be strobed.

Option	Description
Interrupts are to be generated manually.	Write 1 to INT_GENERATE at the required points in the code to fire the next interrupt.
Interrupts are to be timer-generated.	You can pause generation at any time by clearing INT_CTRL.TIMER_EN.

Option	Description
Additional interrupts can be generated manually	If INT_CTRL.ENABLE is HIGH.

6. Write 0 to INT_CTRL.ENABLE to stop the generation of subsequent interrupts.
7. Read INT_NUMBER to determine how many interrupts fired.
   If this value is `0x8hFF`, then all possible interrupts fired, as the INTGEN_INTS option configures. If the value is not `0x8hFF`, then it provides the actual number of interrupts that fired.
8. Write 1 to INT_ACK to clear all level-sensitive interrupts and to reset INT_NUMBER back to zero. The interrupt generator is now ready to be reconfigured or restarted.

Note

Writing to INT_GENERATE when INT_CTRL.ENABLE is 0 has no effect. Generation is UNPREDICTABLE if an interrupt number appears in more than one INT_SEQx register.